

SilkCentral Test Manager 12.0



API Hilfe

Micro Focus
575 Anton Blvd., Suite 510
Costa Mesa, CA 92626

Copyright © 2012 Micro Focus. Alle Rechte vorbehalten. SilkCentral Test Manager enthält Derivatezeugnisse von Borland Software Corporation, Copyright © 2004-2009 Borland Software Corporation (eine Micro Focus-Gesellschaft).

MICRO FOCUS und das Logo von Micro Focus sind u.a. Markenzeichen oder eingetragene Markenzeichen von Micro Focus IP Development Limited oder deren Tochtergesellschaften bzw. Konzerngesellschaften in den Vereinigten Staaten, Großbritannien und anderen Ländern.

BORLAND, das Logo von Borland und SilkCentral Test Manager sind Markenzeichen oder eingetragene Markenzeichen der Borland Software Corporation oder deren Tochtergesellschaften bzw. Konzerngesellschaften in den Vereinigten Staaten, Großbritannien und anderen Ländern.

Alle anderen Markenzeichen sind Eigentum der jeweiligen Inhaber.

2012-03-08

Inhalt

Einführung	5
Erstellen von Plug-Ins	6
Integration Codeabdeckung	7
Erstellen Ihres eigenen Codeabdeckungs-Plug-Ins	7
Test Manager-Codeanalysefluss für Codeabdeckungs-Plug-In	8
Beispielprofilklasse	9
Codeabdeckung XSD	11
XML-Beispielsdatei	12
Beispielschnittstellenklasse für Codeanalyse	13
Installation des Codeanalyse Systems in einer Linux-AUT-Umgebung	14
Integration der Versionsverwaltung	16
Schnittstelle der Versionsverwaltungsintegration	16
Konventionen für die Integration der Versionsverwaltung	17
Integration der Fehlerverfolgung	18
Webdienst-Schnittstelle	18
WSDL-Spezifikation	21
Konfigurieren eines Webdienst-Profiles	27
Java API-Schnittstelle	27
Anforderungsverwaltungs-Integrationen	29
Java API-Schnittstellen	29
Integration von Drittanbieter-Testtypen	30
Plug-In-Implementierung	30
Komprimierung	30
Übergabe von Parametern an das Plug-In	31
Struktur der API	31
Beispielcode	32
XML-Konfigurationsdatei	35
Plug-In-Metadaten	35
Allgemeine Metadaten für Eigenschaften	35
Metadaten für String-Eigenschaften	35
Metadaten für Dateieigenschaften	36
Benutzerdefinierte Symbole	36
Testumgebung	37
Angaben von Start und Ende von Videoaufnahmen	37
SilkCentral-Webdienste	39
Webdienste – Kurzanleitung	39
Voraussetzungen	39
Einführung in Webdienste	40
Webdienst-Client – Übersicht	40
Beispiel für einen Anwendungsfall: Hinzufügen einer Anforderung	43
Sitzungen	44
Verfügbare Webdienste	45
Services Exchange	46
reportData-Schnittstelle	46
TMAttach-Schnittstelle	47
Schnittstelle createTestPlan	48
Schnittstelle exportTestPlan	50
Schnittstelle updateTestPlan	51
Schnittstelle createRequirements	52
Schnittstelle exportRequirements	53

Schnittstelle updateRequirements	54
Schnittstelle updateRequirementsByExternalID	55
Schnittstelle createExecutionDefinitions	56
Schnittstelle exportExecutionDefinitions	59
Schnittstelle updateExecutionDefinitions	60
Schnittstelle createLibraries	61
Schnittstelle exportLibraryStructure	63
Schnittstelle exportLibraryStructureWithoutSteps	64
Schnittstelle getLibraryInfoByName	65
Demo Client für Webdienste	65

Einführung

In dieser Hilfe finden Sie Informationen darüber, wie Plug-Ins erstellt und verteilt werden, um Tools von Drittanbietern in SilkCentral® Test Manager™ (Test Manager) zu integrieren. Sie enthält Webdienst-Spezifikationen und API-Beschreibungen und erläutert, wie Plug-Ins in Test Manager integriert werden.



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

Übersicht

Test Manager stellt eine API für die Integration von Anwendungen von Drittanbietern bereit. Mit den APIs von SilkCentral können Sie vorhandene Tools zur Versionsverwaltung, Fehlerverfolgung und Anforderungsverwaltung integrieren, indem Sie Test Manager-Plug-Ins konfigurieren. Test Manager wird mit einer Reihe von Beispiel-Plug-Ins ausgeliefert.

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in [Javadoc](#). Wenn der Link nicht funktioniert, klicken Sie im Test Manager-Menü auf **Hilfe > Dokumentation > Test Manager API-Spezifikation**, um Javadoc zu öffnen.

Test Manager– Integrations-Plug-Ins

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

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

Erstellen von Plug-Ins

Übersicht

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

Plug-In-Art

Test Manager stellt mehrere Plug-In-APIs bereit. Jede API stellt eine *Art* dar.

Compilierung

In den Versionshinweisen zu Test Manager finden Sie die zum Entwickeln und Kompilieren von Plug-Ins geeignete Java-Version. Dies ist für die Kompatibilität mit dem JRE von Test Manager wichtig.

Testumgebung

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

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

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



Hinweis: Damit das neue Plug-In in Test Manager verfügbar wird, müssen Sie den Anwendungsserver und den Front-End-Server neu starten. Nähere Informationen über das Neustarten der Server finden Sie in der *Hilfe zum SilkCentral-Verwaltungsmodul*.

Verteilung

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

Integration Codeabdeckung

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

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

- SilkCentral Java Codeanalyse (Java Codeanalyse Agent)
- DevPartner Studio .NET Codeanalyse (Windows Codeanalyse System)



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

Falls die zwei vorher genannten Tools nicht ausreichend sind, können Sie Ihre eigene Integration zur Codeabdeckung erstellen und verteilen. Siehe *Erstellen Ihres eigenen Codeabdeckungs-Plug-Ins*.



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

Erstellen Ihres eigenen Codeabdeckungs-Plug-Ins

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



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



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

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

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

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



Hinweis: Die Methoden in den folgenden Schritten, die Sie schreiben werden, werden von SilkCentral abgerufen, wenn sie benötigt werden. Das heißt, Sie werden diese nicht direkt abrufen.

5. Code `getBaseline`. Diese Methode sollte eine XML-Datei erzeugen, die alle Namensräume/Pakete/Klassen/Methoden der Anwendung enthält. Details über das Format der Datei finden Sie in der

Themendatei *XML-Beispieldatei*. Überprüfen Sie die XML unter Verwendung der XSD-Beispieldatei. Details über XSD finden Sie im Thema *Codeabdeckung XSD*.

Diese Funktion wird aufgerufen, bevor mit der Abdeckung begonnen wird, und wird entweder durch das Starten eines Testlaufs des SilkCentral-Ausführungsserver ausgelöst oder durch die SilkCentral **Manual Test Client Start**-Schaltfläche, um die Codeanalyse zu starten und alle abzudeckenden Objekte darzustellen. Die Ausgabe muss unter Verwendung des im XML-Schema spezifizierten Formats, welches im Installationsorder CA-Framework enthalten ist, in XML konvertiert werden.

6. Code `startCoverage`. Dieser Aufruf sollte dem Codeabdeckungs-Tool mitteilen, mit der Datensammlung zu beginnen. Antworten Sie *Wahr*, wenn es startet.

Dies wird durch das SilkCentral-Codeabdeckungssystem aufgerufen, nachdem die Methode `getBaseLine()` abgeschlossen ist. Jetzt sollten Sie das Codeabdeckungs-Tool zum Sammeln von Codeabdeckungsdaten starten.

7. Code `stopCoverage`. Dieser Aufruf sollte dem Codeabdeckungs-Tool mitteilen, die Datensammlung zu beenden. Antworten Sie *Wahr* für erfolgreich.

Dies wird nach `startCoverage` aufgerufen, und wird entweder durch das Beenden eines Testlaufs des SilkCentral-Ausführungsserver ausgelöst oder über die SilkCentral **Manual Test Client Stopp**-Schaltfläche, um die Codeanalyse anzuhalten.

8. Code `getCoverage`. Dies erzeugt eine XML-Datei mit den gesammelten Daten von den zwischen den Methoden `startCoverage` und `stopCoverage` gesammelten Daten. Details über das Format der Datei finden Sie in der Themendatei *XML-Beispieldatei*. Überprüfen Sie die XML unter Verwendung der XSD-Beispieldatei. Details über XSD finden Sie im Thema *Codeabdeckung XSD*.

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

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

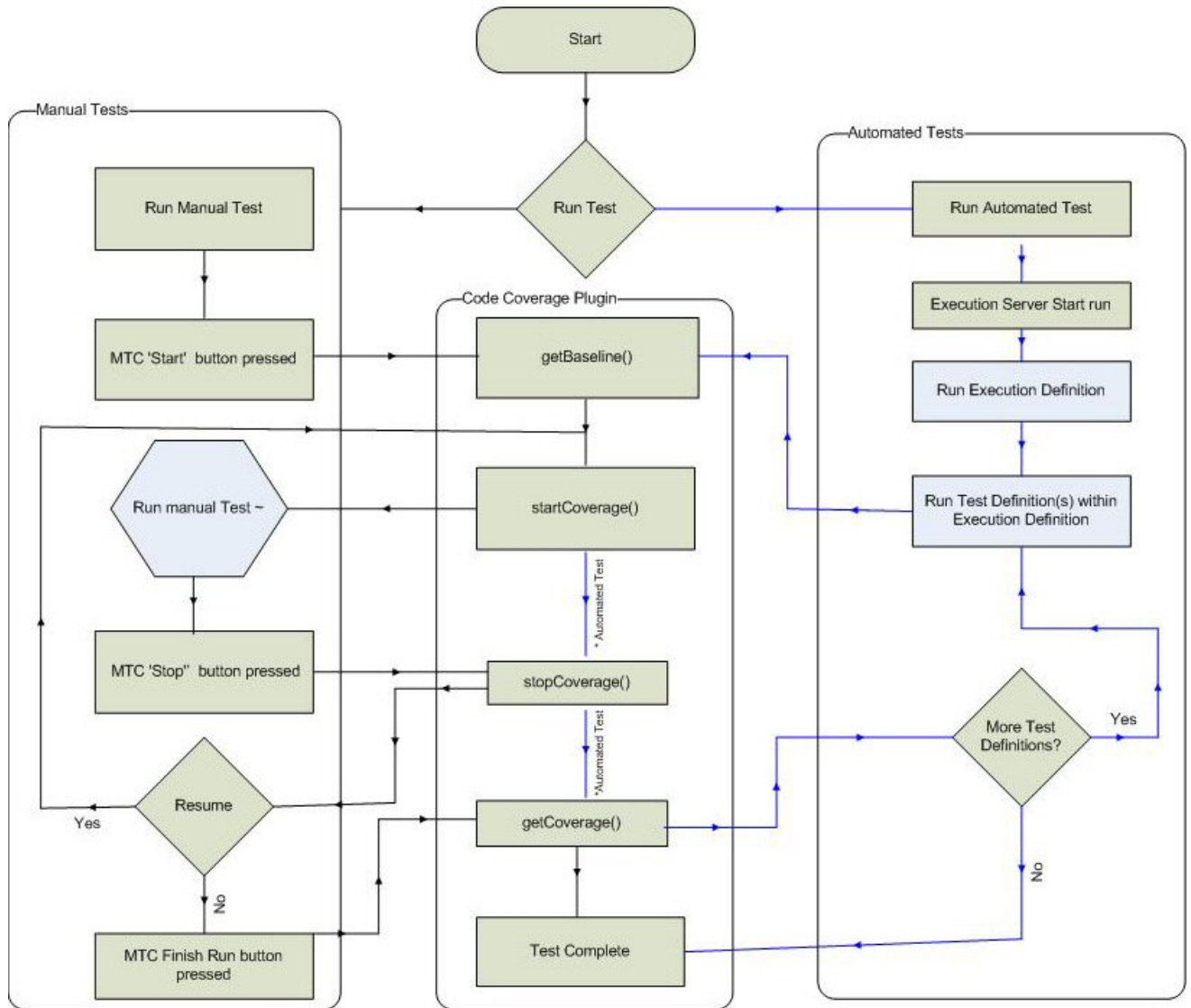
Dies wird zuerst durch das SilkCentral-Codeabdeckungssystem aufgerufen. Der Name des Plug-Ins wird in der Codeabdeckungs-Dropdown in angezeigt Der Name des Plug-Ins wird in der Codeabdeckungs-Dropdown in SCTM angezeigt.

10. Integrieren Sie das Plug-In in ein Jar und platzieren Sie das Jar in einer ZIP-Datei.

11. Verteilen Sie Ihr Plug-In auf die folgenden Servergruppen:

- Im Verzeichnis `Plugins` des SilkCentral-Installationsordners.
- Im Verzeichnis `Plugins` der CA-Framework-Installation.

Test Manager-Codeanalysefluss für Codeabdeckungs-Plug-In



Beispielprofilklasse

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

```
//Add the library scc.jar to your classpath as it contains the interfaces that
//must be extended. The JAR file can be found in the lib directory of the
Test
//Manager installation directory.
//
//make sure to include these imports after adding the scc.jar external
reference
import com.segue.scc.published.api.codeanalysis.CodeAnalysisProfile;
import com.segue.scc.published.api.codeanalysis.CodeAnalysisProfileException;
import com.segue.scc.published.api.codeanalysis.CodeAnalysisResult;

public class myProfileClass implements CodeAnalysisProfile{

    // This function is called first by the SCTM Code Coverage framework
    // The name of the plug-in is displayed in the code coverage drop down in
```

```

SCTM
@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 either the SCTM Execution Server starting a test run
// or the SCTM MTC 'Start' button 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 SCTM 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 either the SCTM Execution Server finishing a test run
// or the SCTM MTC 'Stop' button 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(),
// this 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 SCTM needed xml
    ...
    return xmlString;
}
}

```

Codeabdeckung XSD

Im folgenden finden Sie die Codeabdeckung XSD, welche Sie verwenden sollten, um die von Ihrem Codeabdeckungs-Tool erstellte XML zu validieren. Dieses Dokument können Sie unter der folgenden Servergruppe finden: <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>

```

XML-Beispielsdatei

Die Codeabdeckung API erwartet XML im folgenden Format.

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

```

<?xml version="1.0" encoding="UTF-8"?><!-- Generated by 'MyPluginTool' at
'2010-11-05T16:11:09'
-->
<data>
    <class name="ProjectA.ClassA1">
        <sourcefile name="C:\Documents and Settings\TestApp\ProjectA\ClassA1.cs"/>
        <coverage hits="8" total="8" type="method"/>
        <coverage hits="30" total="30" type="line"/>
        <method inv="2" name="ClassA1 : ()V">

```

```

        <coverage hits="3" lines="11:2,12:2,14:2" total="3" type="line"/>
    </method>
    <method inv="2" name="BoolByteMethod : (LSystem/Boolean;LSystem/
SByte;)V">
        <coverage hits="3" lines="17:2,18:2,19:2" total="3" type="line"/>
    </method>
    <method inv="1" name="CharStringMethod : (LSystem/Char;LSystem/
String;)V">
        <coverage hits="3" lines="38:1,39:1,40:1" total="3" type="line"/>
    </method>
    <method inv="2" name="DateMethod : (LSystem/DateTime;)V">
        <coverage hits="3" lines="22:2,23:2,24:2" total="3" type="line"/>
    </method>
    <method inv="1" name="DecimalMethod : (LSystem/Decimal;LSystem/
Single;LSystem/Double;)V">
        <coverage hits="4" lines="27:1,28:1,29:1,30:1" total="4" type="line"/>
    </method>
    <method inv="1" name="IntMethod : (LSystem/Int32;LSystem/Int64;LSystem/
Int16;)V">
        <coverage hits="3" lines="33:1,34:1,35:1" total="3" type="line"/>
    </method>
    <method inv="1" name="passMeArrays : (LSystem/Int32[];LSystem/
Decimal[];)V">
        <coverage hits="6" lines="51:1,52:1,53:1,55:1,56:1,58:1" total="6"
type="line"/>
    </method>
    <method inv="1" name="passMeObjects : (LSystem/Object;LSystem/Object/
ClassA1;)V">
        <coverage hits="5" lines="43:1,44:1,45:1,46:1,48:1" total="5"
type="line"/>
    </method>
</class>
<class name="TestApp.Form1">
    <sourcefile name="C:\Documents and Settings\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>

```

Beispielschnittstellenklasse für Codeanalyse

Das folgende Codebeispiel definiert die Codeanalysebeispielschnittstelle.

```

package com.segue.scc.published.api.codeanalysis;

public interface CodeAnalysisProfile
{
    /**
     * This function is called by the SCTM Code Coverage Framework
     * after the getBaseLine() method is complete
     * this is where you should start your code coverage tool
     * collecting code coverage data
     * @return true if the operation is successful
     * @throws CodeAnalysisProfileException if an error occurs in this operation

```

```

*/
boolean startCoverage() throws CodeAnalysisProfileException;
/**
 * This function is called after startCoverage,
 * This is triggered by either the SCTM Execution Server finishing a test
run
 * or the SCTM MTC 'Stop' button to stop code analysis
 * Call to your code coverage tool to stop collecting data here.
 * @return true if the operation is successful
 * @throws CodeAnalysisProfileException if an error occurs in this operation
 */
boolean stopCoverage() throws CodeAnalysisProfileException;
/**
 * This function is called before starting coverage.
 * This is triggered by either the SCTM Execution Server starting a test run
 * or the SCTM MTC 'Start' button to start code analysis
 * @return all of the objects to be covered. The output needs to be
 * converted into xml using the format specified in the XML schema
 * CCoverage.xsd included in the CA-Framework installation folder
 * @throws CodeAnalysisProfileException if an error occurs in this operation
 */
String getBaseline() throws CodeAnalysisProfileException;
/**
 * This function is called after stopCoverage().
 * @return all the coverage data collected. The output needs to be
 * converted into xml using the specified in the XML schema
 * CCoverage.xsd included in the CA-Framework installation folder
 * @throws CodeAnalysisProfileException if an error occurs in this operation
 */
String getCoverage() throws CodeAnalysisProfileException;
/**
 * This function is called first by the SCTM Code Coverage framework
 * The name of the plug-in is displayed in the code coverage drop down in
SCTM
 * @return the name of the plug-in tool
 */
String getName();
}

```

Installation des Codeanalyse Systems in einer Linux-AUT-Umgebung

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

1. Das Codeanalyse Systems für Linux ist verfügbar unter **Hilfe > Tools > Linux Codeanalyse System**. Laden Sie dieses herunter und kopieren Sie es in die Stammkonten oder einen anderen Ordner auf der Linux-Maschine.
2. Stellen Sie sicher, dass Java 1.6 oder höher in der Anwendung auf der Testmaschine installiert ist.
3. Entpacken Sie CA-Framework.tar.gz.
4. Legen Sie das Codeanalyse-Plug-In im Ordner<install dir>/SC Test Manager 12.0/Plugins ab.
5. Ändern Sie das Verzeichnis in <install dir>/SC Test Manager 12.0/Codeanalyse.
6. Finden Sie das Shellskript startCodeAnalysisFramework.sh, welches den Prozess CA-Framework ausführt.
7. Führen Sie den folgenden Befehl aus, um die Datei in das Unix-Format zu konvertieren: dos2unix startCodeAnalysisFramework.sh.

8. Führen Sie den folgenden Befehl aus, um die benötigten Berechtigungen einzustellen, die zur Ausführung des Shellskripts notwendig sind: `chmod 775 startCodeAnalysisFramework.sh`.
9. Führen Sie das folgende Shellskript aus, um den CAFramework-Prozess auszuführen: `./startCodeAnalysisFramework.sh`.

Das Codeanalyse System ist zur Verwendung von Test Manager aus bereit.

Integration der Versionsverwaltung

Mithilfe von Versionsverwaltungsprofilen kann Test Manager in externe Versionsverwaltungssysteme integriert werden. Test Manager wird mit Plug-Ins für die folgenden Versionsverwaltungssysteme ausgeliefert:

- StarTeam®
- Serena® Version Manager™ (PVCS®)
- Concurrent Version System (CVS)
- Microsoft® Visual SourceSafe® (MSVSS)
- Universal Naming Convention (UNC) (Dateisystemzugriff)
- Subversion
- Apache Commons Virtual File System (VFS)
- Microsoft Team Foundation Server 2010 (TFS)

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

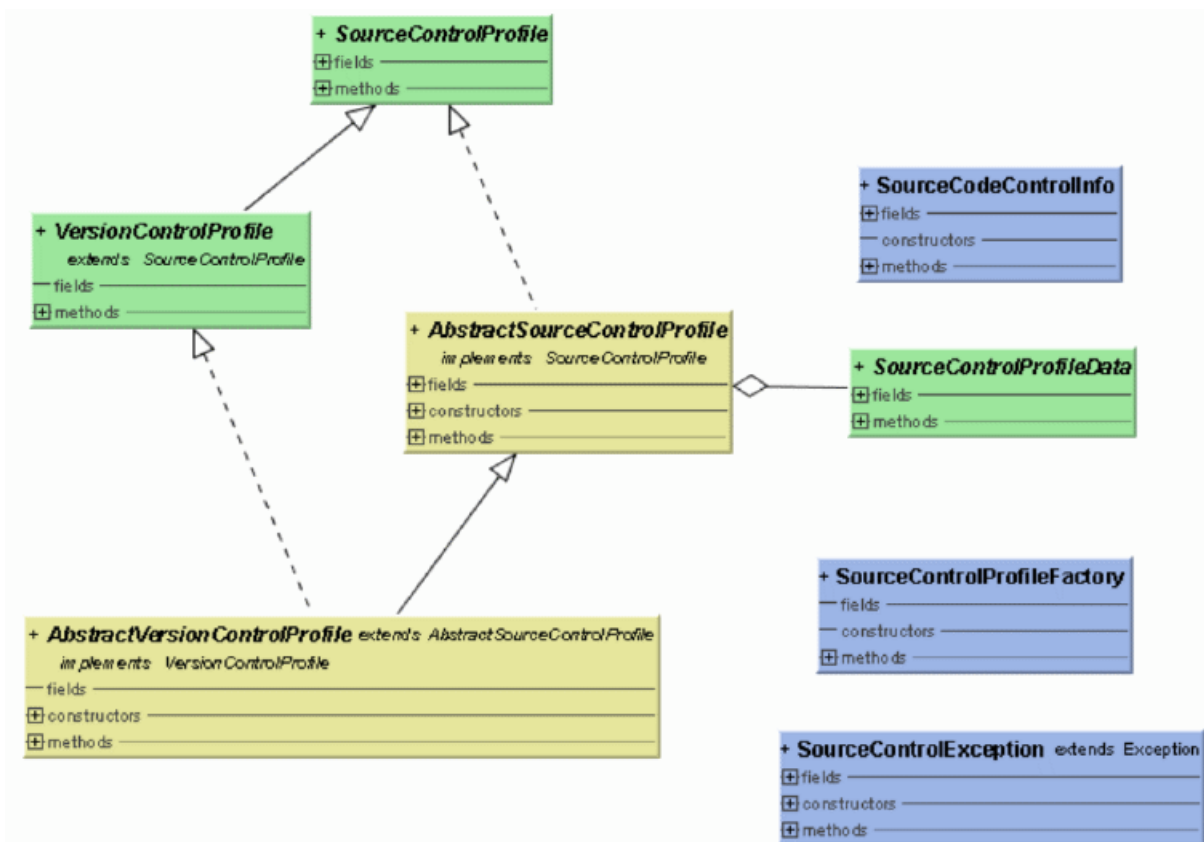
Sie können in Test Manager benutzerdefinierte Versionsverwaltungs-Plug-Ins konfigurieren und festlegen, wo die Ausführungsserver von Test Manager den Programmcode für die Testausführung abrufen sollen. Weitere Informationen hierzu finden Sie in der *SilkCentral Test Manager 12.0-Hilfe*.

Schnittstelle der Versionsverwaltungsintegration

Test Manager unterscheidet zwischen `SourceControlProfile` und `VersionControlProfile`. Der Unterschied besteht darin, dass `SourceControlProfile` im Gegensatz zu `VersionControlProfile` keine Versionsnummern hat.

Die folgenden Test Manager-Schnittstellen werden für die Versionsverwaltungsintegration verwendet:

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



Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in [Javadoc](#). Wenn der Link nicht funktioniert, klicken Sie im Test Manager-Menü auf **Hilfe > Dokumentation > Test Manager API-Spezifikation**, um Javadoc zu öffnen.

Konventionen für die Integration der Versionsverwaltung

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

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

Integration der Fehlerverfolgung

Die in diesem Kapitel besprochenen Schnittstellen werden benötigt, um Plug-Ins für Test Manager zu erstellen, die die Integration eines externen Fehlerverfolgungssystems (Issue Tracking System, ITS) erlauben.

Es gibt zwei Arten der Integration:

- Webdienst-Schnittstelle
- Java API-Schnittstelle

Mit Hilfe von Fehlerverfolgungsprofilen kann Test Manager in externe Fehlerverfolgungssysteme integriert werden. Die folgenden Fehlerverfolgungspakete werden von Test Manager derzeit standardmäßig unterstützt:

- SilkCentral Issue Manager
- IBM® Rational® ClearQuest®
- StarTeam®
- Webdienst zur Fehlerverfolgung
- Bugzilla
- Atlassian JIRA

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

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

Weitere Informationen hierzu finden Sie in der *Test Manager-Hilfe*.

Webdienst-Schnittstelle

Die Schnittstelle, auf der die WSDL basiert, ist weiter unten dargestellt. Sie enthält auch die Dokumentation für die Methoden und Typen. Der folgende Java-Code erläutert die Schnittstelle, was aber nicht bedeutet, dass der Webdienst in Java geschrieben werden muss.



Hinweis: Der Webdienst muss unabhängig von der verwendeten Programmiersprache dieselbe SOAP API bereitstellen, die in der WSDL-Spezifikation beschrieben ist.



Hinweis: Die Webdienst-Schnittstelle erlaubt nur die Erstellung neuer Fehler. Wenn Sie vorhandene Fehler Tests zuordnen möchten, sollten Sie die Java-API verwenden.

Webdienst-Schnittstelle

```
/**
 * This interface is used to generate the WSDL of the
 * Web Service a customer can implement
 * to integrate any, in the first place not supported,
 * Issue Tracking System.
 *
 * The WSDL document will be generated out of this
 * interface using Apache Axis Java2WSDL tool
 *
 */
public interface IssueTrackingSOAPInterface {

    /**
```

```

* This method returns an ID that is used for all
* actions during a session to identify the user. The
* returned value will be passed to the other methods as
* parameter.
* @param user
* @param password
* @return a sessionID which is used for all other
* actions to determine the user
*/
public long logonUser(String user, String password)
    throws IssueTrackingServiceException;

/**
* Will be called when starting to enter a new issue
* @param sessionID
* @return a (temporary) id for the new issue
*/
public String newIssue(long sessionID)
    throws IssueTrackingServiceException;

/**
* Returns all IssueField objects of the (temporary)
* issue with the provided issueID.
* The IssueField objects contain the current values
* of the fields.
* @param sessionID
* @param issueID
* @return
*/
public IssueField[] getFields(long sessionID,
    String issueID)
    throws IssueTrackingServiceException;

/**
* This method is used to synchronize with the current
* states of the external tracking tool.
* @param sessionID
* @param issueIDs array containing the ids of the
* requested issues
* @return state info of requested issues [0] id, [1]
* state, [2] longvalue (changedAt)
*/
public String[][] getStates(long sessionID,
    String[] issueIDs)
    throws IssueTrackingServiceException;

/**
* This method is used to get all possible states an
* issue may have to do a mapping between internal and
* external states.
* @param sessionID
* @return all states an issue may have in this
* profile
*/
public String[] getPossibleStates(long sessionID)
    throws IssueTrackingServiceException;

/**
* This method is called if a field with type SELECT_
* FIELD has dependent fields and was changed.
* e.g.: The field "Product" has a depending field
* "Release". If "Product" changes, the IssueField
* for "Release" has to be returned containing the
* proper options for the selected product.

```

```

* @param sessionID
* @param issueID (temporary) id of the issue
* @param field that has changed
* @return all fields of the issue that have changed
* (eg. other options) due to the change of the submitted
* field
*/
public IssueField[] fieldChanged(long sessionID,
    String issueID, IssueField field)
    throws IssueTrackingServiceException;

/**
* this method stores the issue with the values which
* are submitted in the fields array.
*
* @param sessionID
* @param id the (temporary) id of the issue
* @param fields array of fields containing the values
* to store
* @return new id of the issue after storing into the
* tracking system
*/
public String store(long sessionID, String id,
    IssueField[] fields)
    throws IssueTrackingServiceException;

/**
* This method returns the Http link to the issue with
* the submitted id if available.
* @param sessionID
* @param id
* @return http link to the issue, null if either no
* link is available or the id is invalid
*/
public String getLink(long sessionID, String id)
    throws IssueTrackingServiceException;

/**
* Not used in this version. May be used in a future
* version to speed up the update process.
* @param product only info of issues linked to this
* product will be returned
* @param changedSince timeMillis, only info of
* issues changed after will be returned.
* Can be ignored by
* implementing services if not supported,
* but then all issues of this
* product will be returned (may slow down updates).
* @return array with actual info about the changed
* issues
*/
public IssueInfo[] getChanges(String product,
    long changedSince)
    throws IssueTrackingServiceException;
}

/**
* The ComplexType IssueField corresponds to the following
* definition.
*/

/**
* An IssueField represents one input field in the
* issue tracking dialog.

```

```

* @param name A unique name for the field (ID)
* @param type The GUI type of the field. SelectBox =
* 1, Checkbox = 2, TextField = 3, TextArea = 4 (handled as TextField in this
version)
* @param value The current value of the field.
* @param options The options for a field of type
* SelectBox (type = 1)
* @param label The label for the field shown in the
* dialog
* @param required Defines whether the user has to
* enter a value for this field or if it can be left blank
* @param length The maximum number of characters that
* can be entered in a TextField or TextArea
* @param dependent Array of names of other fields
* whose content depend on the selection of this field.
* Used for SelectBoxes (type = 1)
*/

/**
* The ComplexType IssueInfo is not used in this version as
* the method getChanges is not yet called.
*/

```

WSDL-Spezifikation

Eine WSDL-Datei (Inhalt siehe weiter unten) beschreibt die SOAP API, die implementiert werden muss, damit die Integration stattfinden kann. Ihr Webservice muss diese WSDL implementieren.

Webdienst-Schnittstelle

Hier folgt die Spezifikation der Webservice-Schnittstelle. Die Datei wurde mit dem Apache-Tool *Java2WSDL* generiert.

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://scc.segus.com/issuetracking"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://scc.segus.com/issuetracking"
xmlns:intf="http://scc.segus.com/issuetracking"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types>
    <schema targetNamespace="http://scc.segus.com/issuetracking"
xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      <complexType name="IssueField">
        <sequence>
          <element name="Name" nillable="true" type="xsd:string"/>
          <element name="Type" type="xsd:int"/>
          <element name="Value" nillable="true" type="xsd:string"/>
          <element maxOccurs="unbounded" name="Options"
nillable="true" type="xsd:string"/>
          <element name="Label" nillable="true" type="xsd:string"/>
          <element name="IsRequired" type="xsd:boolean"/>
          <element name="MaxLength" type="xsd:int"/>
          <element maxOccurs="unbounded" name="Dependent"
nillable="true" type="xsd:string"/>
        </sequence>
      </complexType>
      <complexType name="ArrayOfIssueField">

```

```

    <complexContent>
      <restriction base="soapenc:Array">
        <attribute ref="soapenc:arrayType"
wsdl:arrayType="impl:IssueField[]" />
      </restriction>
    </complexContent>
  </complexType>
<complexType name="IssueTrackingServiceException">
  <sequence/>
</complexType>
<complexType name="ArrayOf_xsd_string">
  <complexContent>
    <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]" />
    </restriction>
  </complexContent>
</complexType>
<complexType name="ArrayOfArrayOf_xsd_string">
  <complexContent>
    <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[][]" />
    </restriction>
  </complexContent>
</complexType>
<complexType name="IssueInfo">
  <sequence>
    <element name="Id" nillable="true" type="xsd:string" />
    <element name="Status" nillable="true" type="xsd:string" />
    <element name="Synopsis" nillable="true" type="xsd:string" />
    <element name="Product" nillable="true" type="xsd:string" />
    <element name="Component" nillable="true" type="xsd:string" />
    <element name="Platform" nillable="true" type="xsd:string" />
    <element name="Url" nillable="true" type="xsd:string" />
    <element name="ChangedOn" type="xsd:long" />
    <element name="ChangedBy" nillable="true" type="xsd:string" />
  </sequence>
</complexType>
<complexType name="ArrayOfIssueInfo">
  <complexContent>
    <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType" wsdl:arrayType="impl:IssueInfo[]" />
    </restriction>
  </complexContent>
</complexType>
</schema>
</wsdl:types>
<wsdl:message name="getChangesRequest">
  <wsdl:part name="product" type="xsd:string" />
  <wsdl:part name="changedSince" type="xsd:long" />
</wsdl:message>
<wsdl:message name="fieldChangedResponse">
  <wsdl:part name="fieldChangedReturn" type="impl:ArrayOfIssueField" />
</wsdl:message>
<wsdl:message name="getFieldsResponse">
  <wsdl:part name="getFieldsReturn" type="impl:ArrayOfIssueField" />
</wsdl:message>
<wsdl:message name="getPossibleStatesResponse">
  <wsdl:part name="getPossibleStatesReturn" type="impl:ArrayOf_xsd_string" />
</wsdl:message>
<wsdl:message name="logonUserResponse">
  <wsdl:part name="logonUserReturn" type="xsd:long" />
</wsdl:message>
<wsdl:message name="getStatesResponse">
  <wsdl:part name="getStatesReturn" type="impl:ArrayOfArrayOf_xsd_string" />

```

```

</wsdl:message>
<wsdl:message name="getPossibleStatesRequest">
  <wsdl:part name="sessionID" type="xsd:long"/>
</wsdl:message>
<wsdl:message name="newIssueResponse">
  <wsdl:part name="newIssueReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="getChangesResponse">
  <wsdl:part name="getChangesReturn" type="impl:ArrayOfIssueInfo"/>
</wsdl:message>
<wsdl:message name="logonUserRequest">
  <wsdl:part name="user" type="xsd:string"/>
  <wsdl:part name="password" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="getFieldsRequest">
  <wsdl:part name="sessionID" type="xsd:long"/>
  <wsdl:part name="issueID" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="storeResponse">
  <wsdl:part name="storeReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="getStatesRequest">
  <wsdl:part name="sessionID" type="xsd:long"/>
  <wsdl:part name="issueIDs" type="impl:ArrayOf_xsd_string"/>
</wsdl:message>
<wsdl:message name="getLinkRequest">
  <wsdl:part name="sessionID" type="xsd:long"/>
  <wsdl:part name="id" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="newIssueRequest">
  <wsdl:part name="sessionID" type="xsd:long"/>
</wsdl:message>
<wsdl:message name="IssueTrackingServiceException">
  <wsdl:part name="fault" type="impl:IssueTrackingServiceException"/>
</wsdl:message>
<wsdl:message name="fieldChangedRequest">
  <wsdl:part name="sessionID" type="xsd:long"/>
  <wsdl:part name="issueID" type="xsd:string"/>
  <wsdl:part name="field" type="impl:IssueField"/>
</wsdl:message>
<wsdl:message name="getLinkResponse">
  <wsdl:part name="getLinkReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="storeRequest">
  <wsdl:part name="sessionID" type="xsd:long"/>
  <wsdl:part name="id" type="xsd:string"/>
  <wsdl:part name="fields" type="impl:ArrayOfIssueField"/>
</wsdl:message>
<wsdl:portType name="IssueTrackingPortType">
  <wsdl:operation name="getFields" parameterOrder="sessionID issueID">
    <wsdl:input message="impl:getFieldsRequest" name="getFieldsRequest"/>
    <wsdl:output message="impl:getFieldsResponse" name="getFieldsResponse"/>
    <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
  </wsdl:operation>
  <wsdl:operation name="store" parameterOrder="sessionID id fields">
    <wsdl:input message="impl:storeRequest" name="storeRequest"/>
    <wsdl:output message="impl:storeResponse" name="storeResponse"/>
    <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
  </wsdl:operation>
  <wsdl:operation name="logonUser" parameterOrder="user password">
    <wsdl:input message="impl:logonUserRequest" name="logonUserRequest"/>
    <wsdl:output message="impl:logonUserResponse" name="logonUserResponse"/>
  </wsdl:operation>
</wsdl:portType>

```

```

        <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
    </wsdl:operation>
    <wsdl:operation name="newIssue" parameterOrder="sessionID">
        <wsdl:input message="impl:newIssueRequest" name="newIssueRequest"/>
        <wsdl:output message="impl:newIssueResponse" name="newIssueResponse"/>
        <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
    </wsdl:operation>
    <wsdl:operation name="getStates" parameterOrder="sessionID issueIDs">
        <wsdl:input message="impl:getStatesRequest" name="getStatesRequest"/>
        <wsdl:output message="impl:getStatesResponse" name="getStatesResponse"/>
        <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
    </wsdl:operation>
    <wsdl:operation name="getPossibleStates" parameterOrder="sessionID">
        <wsdl:input message="impl:getPossibleStatesRequest"
name="getPossibleStatesRequest"/>
        <wsdl:output message="impl:getPossibleStatesResponse"
name="getPossibleStatesResponse"/>
        <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
    </wsdl:operation>
    <wsdl:operation name="fieldChanged" parameterOrder="sessionID issueID
field">
        <wsdl:input message="impl:fieldChangedRequest"
name="fieldChangedRequest"/>
        <wsdl:output message="impl:fieldChangedResponse"
name="fieldChangedResponse"/>
        <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
    </wsdl:operation>
    <wsdl:operation name="getLink" parameterOrder="sessionID id">
        <wsdl:input message="impl:getLinkRequest" name="getLinkRequest"/>
        <wsdl:output message="impl:getLinkResponse" name="getLinkResponse"/>
        <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
    </wsdl:operation>
    <wsdl:operation name="getChanges" parameterOrder="product changedSince">
        <wsdl:input message="impl:getChangesRequest" name="getChangesRequest"/>
        <wsdl:output message="impl:getChangesResponse"
name="getChangesResponse"/>
        <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="IssueTrackingSOAPBinding"
type="impl:IssueTrackingPortType">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/
http"/>
    <wsdl:operation name="getFields">
        <wsdlsoap:operation soapAction="" />
        <wsdl:input name="getFieldsRequest">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.segue.com/issuetracking" use="encoded"/>
        </wsdl:input>
        <wsdl:output name="getFieldsResponse">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.segue.com/issuetracking" use="encoded"/>
        </wsdl:output>
        <wsdl:fault name="IssueTrackingServiceException">
            <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/

```

```

encoding/"
namespace="http://scc.seguel.com/issuetracking" use="encoded"/>
  </wsdl:faul>
</wsdl:operation>
<wsdl:operation name="store">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="storeRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguel.com/issuetracking" use="encoded"/>
  </wsdl:input>
  <wsdl:output name="storeResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguel.com/issuetracking" use="encoded"/>
  </wsdl:output>
  <wsdl:faul name="IssueTrackingServiceException">
    <wsdlsoap:faul encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguel.com/issuetracking" use="encoded"/>
  </wsdl:faul>
</wsdl:operation>
<wsdl:operation name="logonUser">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="logonUserRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguel.com/issuetracking" use="encoded"/>
  </wsdl:input>
  <wsdl:output name="logonUserResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguel.com/issuetracking" use="encoded"/>
  </wsdl:output>
  <wsdl:faul name="IssueTrackingServiceException">
    <wsdlsoap:faul encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguel.com/issuetracking" use="encoded"/>
  </wsdl:faul>
</wsdl:operation>
<wsdl:operation name="newIssue">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="newIssueRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguel.com/issuetracking" use="encoded"/>
  </wsdl:input>
  <wsdl:output name="newIssueResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguel.com/issuetracking" use="encoded"/>
  </wsdl:output>
  <wsdl:faul name="IssueTrackingServiceException">
    <wsdlsoap:faul encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguel.com/issuetracking" use="encoded"/>
  </wsdl:faul>
</wsdl:operation>
<wsdl:operation name="getStates">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="getStatesRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguel.com/issuetracking" use="encoded"/>

```

```

        </wsdl:input>
        <wsdl:output name="getStatesResponse">
          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
        </wsdl:output>
        <wsdl:fault name="IssueTrackingServiceException">
          <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
        </wsdl:fault>
      </wsdl:operation>
      <wsdl:operation name="getPossibleStates">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="getPossibleStatesRequest">
          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
        </wsdl:input>
        <wsdl:output name="getPossibleStatesResponse">
          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
        </wsdl:output>
        <wsdl:fault name="IssueTrackingServiceException">
          <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
        </wsdl:fault>
      </wsdl:operation>
      <wsdl:operation name="fieldChanged">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="fieldChangedRequest">
          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
        </wsdl:input>
        <wsdl:output name="fieldChangedResponse">
          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
        </wsdl:output>
        <wsdl:fault name="IssueTrackingServiceException">
          <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
        </wsdl:fault>
      </wsdl:operation>
      <wsdl:operation name="getLink">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="getLinkRequest">
          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
        </wsdl:input>
        <wsdl:output name="getLinkResponse">
          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
        </wsdl:output>
        <wsdl:fault name="IssueTrackingServiceException">
          <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>

```

```

</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="getChanges">
  <wsdlsoap:operation soapAction="" />
  <wsdl:input name="getChangesRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/" namespace="http://scc.seguate.com/issuetracking" use="encoded" />
  </wsdl:input>
  <wsdl:output name="getChangesResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/" namespace="http://scc.seguate.com/issuetracking" use="encoded" />
  </wsdl:output>
  <wsdl:fault name="IssueTrackingServiceException">
    <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/" namespace="http://scc.seguate.com/issuetracking" use="encoded" />
  </wsdl:fault>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="IssueTrackingService">
  <wsdl:port binding="impl:IssueTrackingSOAPBinding"
name="IssueTrackingPort">
    <wsdlsoap:address location="http://issuetracking.endpoint" />
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

Konfigurieren eines Webdienst-Profiles

Sobald Ihr benutzerdefinierter Fehlerverfolgungs-Webdienst läuft, können Sie ein Fehlerverfolgungsprofil konfigurieren.

1. Klicken Sie im Menü **Projekteinstellungen > Fehlerverfolgung**. Auf der Seite **Fehlerverfolgung** werden alle Fehlerverfolgungsprofile angezeigt, die für das System erstellt wurden.
2. Geben Sie einen **Namen** und eine **Beschreibung** für das neue Fehlerverfolgungsprofil ein.
3. Wählen Sie im Listenfeld **Typ** die Option **Webdienst zur Fehlerverfolgung**.
4. Geben Sie im Feld **Webservice-Adresse** die URL Ihres Webdienstes ein.
5. Geben Sie in die entsprechenden Felder einen gültigen **Benutzernamen** und ein **Kennwort** ein.
6. Klicken Sie auf **OK**.

Java API-Schnittstelle

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in [Javadoc](#). Wenn der Link nicht funktioniert, klicken Sie im Test Manager-Menü auf **Hilfe > Dokumentation > Test Manager API-Spezifikation**, um Javadoc zu öffnen.

Build-Umgebung

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

Sie müssen zwei Schnittstellen/Klassen erweitern:

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

Klassen/Schnittstellen



- IssueTrackingProfile
- IssueTrackingData
- Issue
- IssueTrackingField
- IssueTrackingProfileException

Anforderungsverwaltungs-Integrationen

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

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

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

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in [Javadoc](#). Wenn der Link nicht funktioniert, klicken Sie im Test Manager-Menü auf **Hilfe > Dokumentation > Test Manager API-Spezifikation**, um Javadoc zu öffnen.

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

Nähere Informationen hierzu finden Sie in der Hilfe zum *SilkCentral-Verwaltungsmodul*.

Java API-Schnittstellen

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in [Javadoc](#). Wenn der Link nicht funktioniert, klicken Sie im Test Manager-Menü auf **Hilfe > Dokumentation > Test Manager API-Spezifikation**, um Javadoc zu öffnen.

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

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

- `RMAction`
- `RMAttachment`
- `RMDataProvider`
- *Optional:* `RMIconProvider`
- `RMNode`
- `RMNodeType`
- `RMPluginProfile`
- `RMPProject`
- `RMTTest`
- `RMTTestParameter`
- *Optional:* `RMTTestProvider`
- `RMTTestStep`

Integration von Drittanbieter-Testtypen

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

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

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

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in [Javadoc](#). Wenn der Link nicht funktioniert, klicken Sie im Test Manager-Menü auf **Hilfe > Dokumentation > Test Manager API-Spezifikation**, um Javadoc zu öffnen.

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

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

Dieser Abschnitt enthält ein Codebeispiel, das den Teststyp "Process Executor" implementiert. Mit "Process Executor" kann eine beliebige ausführbare Datei aufgerufen werden. Es erweitert die öffentliche Klasse "Process Test Launcher". Weitere Informationen finden Sie im *Beispiel Teststart-Plug-In*, das Sie unter **Hilfe > Tools** herunterladen können, sowie in der Datei `Readme.txt`.

Plug-In-Implementierung

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

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

Komprimierung

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

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

Übergabe von Parametern an das Plug-In

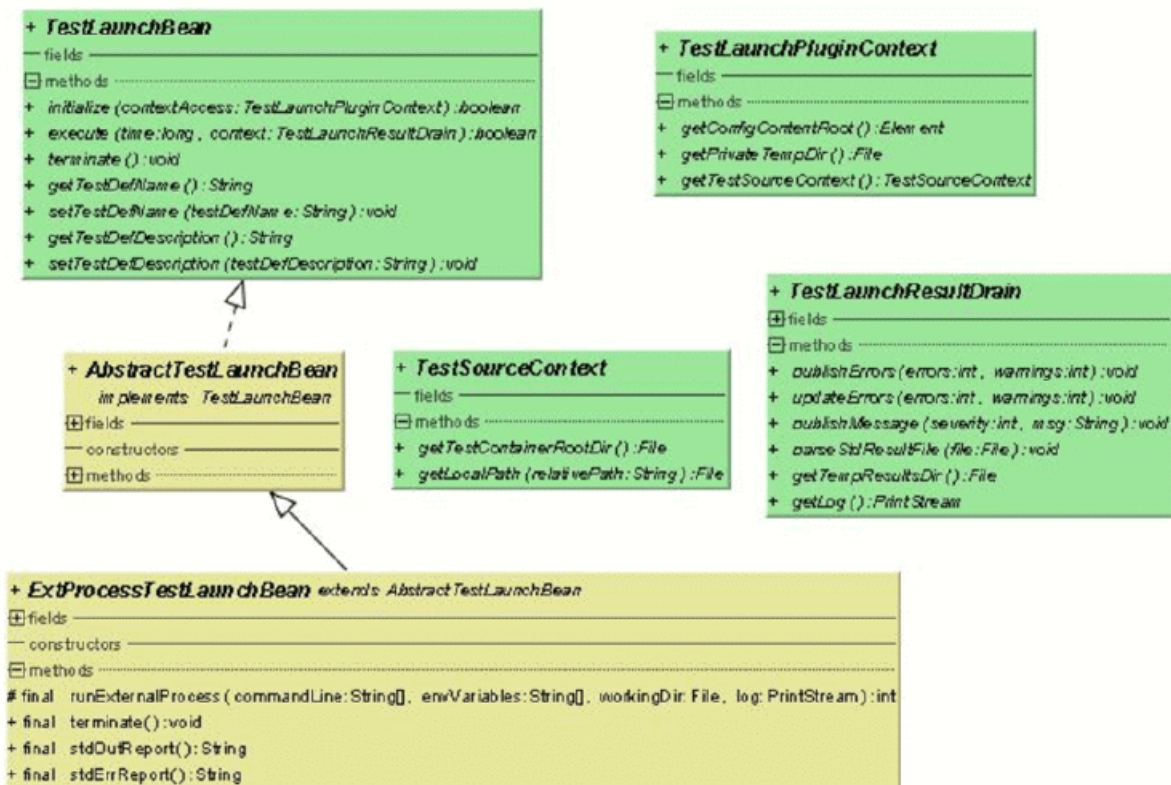
Die Plug-In-Schnittstelle erlaubt den uneingeschränkten Zugriff auf alle benutzerdefinierten Parameter, die im Bereich **Tests** von Test Manager definiert wurden. Für Testtypen von Drittanbietern werden nur benutzerdefinierte Parameter unterstützt. Das Plug-In kann keine vordefinierten Parameter verwenden.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

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

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

Struktur der API

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



Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in [Javadoc](#). Wenn der Link nicht funktioniert, klicken Sie im Test Manager-Menü auf **Hilfe > Dokumentation > Test Manager API-Spezifikation**, um Javadoc zu öffnen.

Verfügbare Schnittstellen

- `TestLaunchBean`
- `ExtProcessTestLaunchBean`
- `TestLaunchPluginContext`

- TestSourceContext
- TestLaunchResultDrain

Beispielcode

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

```
package com.borland.sctm.testlauncher;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

import com.segue.tm.published.api.testlaunch.ExtProcessTestLaunchBean;
import com.segue.tm.published.api.testlaunch.TestLaunchResultDrain;

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

    // test properties that will be set by SCTM 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 SCTM 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 SCTM 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 SCTM 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 Test Manager 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[] getEnvironmentVariables(TestLaunchResultDrain context)
    throws IOException {
    String[] envVariables = {
        "SCTM_EXEC_RESULTS_FOLDER=",
        + context.getTempResultsDir().getAbsolutePath(),
        "SCTM_EXEC_SOURCE_FOLDER="
    }
}

```

```

        + sourceAccess().getTestContainerRootDir().getAbsolutePath(),
    };
    return envVariables;
}

/**
 * Let SCTM parse the standard report xml file (output.xml) if exists.
 * See also Test Manager Web Help - Creating a Test Package. A XSD file
 * can be found in SCTM 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 executed in working folder '%s'",
            workingFolderFile.getAbsolutePath()));
    return workingFolderFile;
}

/**
 * Retrieves the command line arguments specified.
 * @param context the test execution context
 * @return an array of command line arguments
 */
private String[] getCommandArgs(TestLaunchResultDrain context) {
    final ArrayList<String> cmdList = new ArrayList<String>();
    final StringBuilder cmd = new StringBuilder();
    cmdList.add(command);
    cmd.append(command);
    if (arguments != null) {
        String[] lines = arguments.split("[\\r\\n]+");
        for (String line : lines) {
            cmdList.add(line);
            cmd.append(" ").append(line);
        }
    }
}

```

```

    }
  }
  context.publishMessage(TestLaunchResultDrain.SEVERITY_INFO,
    String.format("executed command '%s'", cmd.toString()));
  context.getLog().printf("start '%s'%n", cmd.toString());
  return (String[]) cmdList.toArray(new String[cmdList.size()]);
}
}

```

XML-Konfigurationsdatei

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

Plug-In-Metadaten

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

Typ	Beschreibung
id	Ein interner String, der den Typ identifiziert und in allen installierten Plug-Ins eindeutig ist.
label	Der Text, der in Auswahllisten und Bearbeitungsdiaologfeldern der Benutzeroberfläche für den Typ angezeigt wird.
description	Zusätzlicher Text, der den Teststyp beschreibt.
version	Zur Unterscheidung verschiedener Versionen eines Typs.

Allgemeine Metadaten für Eigenschaften

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

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

Metadaten für String-Eigenschaften

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

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

Metadaten für Dateieigenschaften

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

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

Typ	Beschreibung
openBrowser	Der Wert true bedeutet, dass ein Browser geöffnet werden soll, um die Datei aus den Dateien in einem Container auszuwählen.
isSourceControl	Der Wert true bedeutet, dass die Datei aus einem Versionsverwaltungssystem stammt.
fileNameSpec	Bezeichnet eine Einschränkung für zulässige Dateinamen wie im Standard-Dialogfeld "Durchsuchen" von Windows.

Benutzerdefinierte Symbole

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

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

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

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

Testumgebung

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



Hinweis: Andere Integrationsverfahren werden nicht unterstützt.

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



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

Angeben von Start und Ende von Videoaufnahmen

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in [Javadoc](#). Wenn der Link nicht funktioniert, klicken Sie im Test Manager-Menü auf **Hilfe > Dokumentation > Test Manager API-Spezifikation**, um Javadoc zu öffnen.

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

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

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

Drittanbieter-Tests, die in einem externen Prozess ausgeführt werden

Für diese Tests können Sie einen TCP/IP-basierten Dienst zum Senden von `START`- und `FINISH`-Meldungen an den Port 19138 des Test Manager-Ausführungsservers verwenden. Anhand dieser Meldungstypen wird der Ausführungsserver informiert, dass ein Testfall in dem Test begonnen oder beendet wurde. Die Meldungen müssen im Unicode- (UTF8) oder ASCII-Format verschlüsselt sein.

Meldungstyp Format

START `START <Test Name>, <Test ID> <LF>`, wobei LF den ASCII-Code 10 hat.

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

Wenn die Anfrage erkannt wurde, gibt der Ausführungsserver die Meldung `OK` zurück; andernfalls gibt er eine Fehlermeldung aus. Warten Sie immer erst die Reaktion des Ausführungsservers ab, bevor Sie den

nächsten Testfall ausführen, da das aufgezeichnete Video und der tatsächliche Testfall sich sonst möglicherweise nicht entsprechen.

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

SilkCentral-Webdienste

Webdienste erfordern keine besonderen Einstellungen. Sie werden auf jedem Front-End-Server standardmäßig aktiviert. Angenommen, <http://www.yourFrontend.com/login> ist die URL, über die Sie auf SilkCentral zugreifen, dann sind <http://www.yourFrontend.com/Services1.0/services> und <http://www.yourFrontend.com/AlmServices1.0/services> die Basis-URLs, über die Sie auf die verfügbaren Webdienste zugreifen.

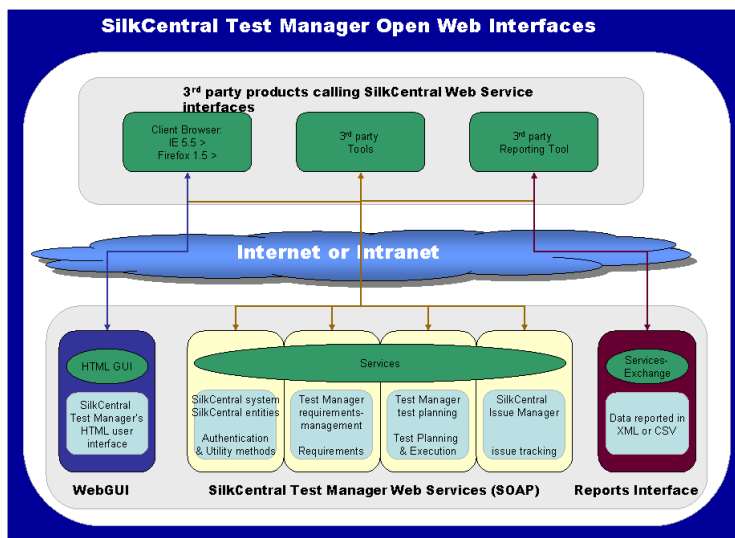
Wenn Sie mit Ihrem Browser auf den Basis-URL zugreifen, wird eine einfache HTML-Liste aller verfügbaren Webdienste angezeigt. Diese Liste wird von Apache Axis bereitgestellt. Der von SilkCentral verwendete SOAP-Stack ist <http://ws.apache.org/axis/>.

Der Basis-URL stellt Links zu XML-Dateien im WSDL (Web Service Definition Language)-Standard bereit, wobei jede Datei die Schnittstelle eines bestimmten Webdienstes beschreibt. Da diese Dateien nur für Programme verständlich sind, werden sie von SOAP-fähigen Clients (z.B. SilkPerformer Java Explorer) gelesen. Auf diese Weise werden die Informationen abgerufen, die erforderlich sind, um Methoden der entsprechenden Webdienste aufzurufen.

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in [Javadoc](#). Wenn der Link nicht funktioniert, klicken Sie im Test Manager-Menü auf **Hilfe > Dokumentation > Test Manager API-Spezifikation**, um Javadoc zu öffnen.



Hinweis: Die vor Version 2008 in Test Manager implementierten Original-Webdienste sind weiterhin unter <http://hostname:port/services> verfügbar.



Webdienste – Kurzanleitung

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

Voraussetzungen

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

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

Einführung in Webdienste

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

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

1. Laden Sie die Axis-Webdienstbibliothek von <http://ws.apache.org/axis/> herunter.
2. Dekomprimieren Sie alle JAR-Archive, und legen Sie sie im Verzeichnis `lib` des Klassenpfades ab. Verwenden Sie dafür die Umgebungsvariable `AXISCLASSPATH`.
3. Führen Sie den folgenden Befehl aus, der auf die WSDL des gewünschten Webdienstes zeigt:

```
java -cp %AXISCLASSPATH%
      org.apache.axis.wsdl.WSDL2Java
      -o . -d Session -p
package.structure.you.want.to.use.for.your.client.yourwebservice
      -t http://url.to.your.service/yourWebService?wsdl
```

4. Suchen Sie nach der automatisch generierten Java-Klasse `YourWebServiceTestCase`. Diese Klasse ist bereits einsatzfähig und kann jede Methode ausführen, die `YourWebService` bereitstellt. Sie brauchen nur noch den leeren Standardvariablen Werte zuzuweisen. Ersetzen Sie z. B. die leeren Strings "Benutzername" und "Kennwort" in dem Aufruf `login(String username, String password)` durch den entsprechenden Benutzernamen und das Kennwort. Diese Klasse ist ein JUnit-Test. Sie können den Code jedoch einfach in eine Java-Klasse kopieren, um einen Client zu erstellen.

Webdienst-Client – Übersicht

Webdienste verwenden normalerweise SOAP anstelle des HTTP-Protokolls. In einem solchen Szenario werden SOAP-Pakete gesendet. Wenn Collections und andere komplexe Objekte in SOAP-Paketen gebündelt werden, ist es kaum möglich, die ASCII-Datenstrukturen zu lesen und zu bearbeiten. Unerfahrene Entwickler sollten nicht versuchen, einen Webdienst-Client durch direkte Bearbeitung von SOAP-Paketen zu erstellen. Erfahrene Entwickler erstellen normalerweise keine Webdienst-Clients auf SOAP-Paket-Ebene. Ein solches Vorgehen wäre mühsam und fehleranfällig. Aus diesem Grund stellen alle wichtigen Programmiersprachen Entwicklungs-Kits für Webdienste bereit. Axis ist das am weitesten verbreitete Java-Kit. SOAP ist das bekannteste Tool für C++ und C. Auch für Perl und Python gibt es derartige Tools.

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

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

Axis folgt diesem Muster. In unseren Beispielen wird das Tool WSDL2Java verwendet, um Client-Rümpfe aus der WSDL zu erstellen. Weitere Informationen über die Verwendung von WSDL2Java finden Sie im [Axis Reference Guide](#). In der obigen Übersicht werden folgende Schalter verwendet:

- `-o`: Das Ausgabeverzeichnis der Client-Rümpfe
- `-d`: Der Gültigkeitsbereich der Webdienst-Verbindung
- `-p`: Package-Struktur für Namespaces verwenden, in diese Package-Struktur verteilen
- `-t`: JUnit-Testfallklasse für den Webdienst ausgeben.

Generieren Sie die JUnit-Testfallklasse. Die Klasse stellt nicht nur Code bereit, der für die Bindung an den Webdienst sorgt, sondern gibt auch Testcode aus, der von jeder einzelnen Methode des Dienstes Gebrauch macht.

Das Tool WSDL2Java generiert mehrere Klassen, die einen Client für den Webdienst unterstützen. Wenn der Name des Dienstes `YourWebService` ist, werden die folgenden Klassen ausgegeben:

- `YourWebService`: Eine Schnittstelle, die `YourWebService` repräsentiert.
- `YourWebServiceService`: Eine Schnittstelle, die den `YourWebService-Locator` repräsentiert.
- `YourWebServiceServiceLocator`: Ein `YourWebService-Locator`, der "`YourWebServiceService`" implementiert.
- `YourWebServiceSoapBindingStub`: Ein Client-Rumpf, der für die Serialisierung der `YourWebService` POJOs verantwortlich ist.
- `YourWebServiceServiceTestCase`: Ein JUnit-Testfall, der von allen Methoden von `YourWebService` Gebrauch macht.
- `Serializeable Objects`: Objekte auf Seiten des Clients, die Objekten entsprechen, die `YourWebService` verwendet.

Aus Gründen der besseren Lesbarkeit empfiehlt es sich, die Schnittstellen und Implementierungsklassen umzubenennen:

- `YourWebServiceService` => wird zu `IYourWebServiceLocator`. Das vorangestellte "I" steht für "Interface", dt. Schnittstelle.
- `YourWebServiceServiceLocator` => wird zu `YourWebServiceLocator` und implementiert `IYourWebServiceLocator`.
- `YourWebService` => wird zu `IYourWebService`.

Der von Axis generierte JUnit-Testfall ist sehr hilfreich. Er stellt einen funktionsfähigen Client dar. Nur die mit voreingestellten Werten initialisierten Variablen müssen so geändert werden, dass sie für den Webdienst gültig sind. Der Testfall stellt auch Bindungscode bereit, den der Locator verwendet, um den SOA-Zyklus "Suchen-Binden-Ausführen" zu starten. Der folgende Auszug stammt aus dem Testfall, der für den Webdienst `tmrequirementsmanagement` bereitgestellt wurde:

```
/* default test automatically generated by WSDL2Java */
public void test2tmrequirementsmanagementLogin() throws Exception {
    com.borland.tm.webservices.tmrequirementsmanagement
        .TmrequirementsmanagementSoapBindingStub binding;
    try {
        binding = (com.borland.tm.webservices.tmrequirementsmanagement
            .TmrequirementsmanagementSoapBindingStub)
            new com.borland.tm.webservices.tmrequirementsmanagement
                .RequirementsServiceLocator().gettmrequirementsmanagement();
    }
    catch (javax.xml.rpc.ServiceException jre) {
        if(jre.getLinkedCause()!=null)
            jre.getLinkedCause().printStackTrace();
        throw new junit.framework.AssertionFailedError("JAX-RPC
            ServiceException caught: " + jre);
    }
    assertNotNull("binding is null", binding);
}
```

```

// Time out after a minute
binding.setTimeout(60000);

// Test operation
try {
    java.lang.String value = null;
    value = binding.login(new java.util.HashMap());
}
catch
(com.borland.tm.webservices.tmrequirementsmanagement.RMServiceException e1) {
    throw new junit.framework.AssertionFailedError("RMServiceException
        Exception caught: " + e1);
}
// TBD - validate results
}

```

Beachten Sie, dass der JUnit-Test den Code für die Bindung und die Anmeldung bereitstellt. Der Entwickler braucht nur die standardmäßige HashMap auszufüllen, um den Anmeldecode fertig zu stellen. Beim Erstellen eines POJO-Webdienst-Clients reicht also einfaches Kopieren, Einfügen und Bearbeiten aus, um funktionsfähige Bindungs- und Anmeldemethoden zu erstellen:

```

public TmrequirementsmanagementSoapBindingStub getBinding()
    throws Exception {
    TmrequirementsmanagementSoapBindingStub binding;
    try {
        binding = (TmrequirementsmanagementSoapBindingStub)
            new RequirementsServiceLocator().gettmrequirementsmanagement();
    }
    catch (ServiceException jre) {
        // Handle error
    }

    if(binding == null)
    {
        // handle error here
        System.err.println("binding is null.");
        System.exit(0);
    }

    // Time out after a minute
    binding.setTimeout(60000);

    return binding;
}

public String login(String username, String password)
{
    // mBinding = getBinding() already obtained
    // Now get mSessionID member variable
    try {
        HashMap map = new HashMap();
        map.put("username", username);
        map.put("password", password);
        mSessionID = mBinding.login(map);
    }
    catch (RMServiceException e1) {
        // Handle error here
    }
    return mSessionID;
}

```

Beispiel für einen Anwendungsfall: Hinzufügen einer Anforderung

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

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

- Sie haben die Schritte für den Webservice *tmrequirementsmanagement* abgeschlossen.
 - Eine funktionierende POJO- oder JUnit-Klasse mit Bindungs-, Anmelde- und weiteren *tmrequirementsmanagement*-Methoden wurde erstellt.
 - Sie haben die anderen Test Manager-API-Hilfethemen gelesen.
1. Stellen Sie mit Hilfe der Anmeldemethode und einer Eigenschaftsliste mit Benutzerinformationen die Bindung zum Dienst her.
 2. Speichern Sie die Sitzungs-ID aus vorherigen Schritt.
 3. Konstruieren Sie ein Anforderungsobjekt, das die gewünschten Daten enthält.
 4. Rufen Sie die Methode `updateRequirement` mit der Sitzungs-ID und dem Anforderungsobjekt auf, das Sie generiert haben.
 5. Speichern Sie die von der Methode `updateRequirement` zurückgegebene Anforderungs-ID.
 6. Erstellen Sie ein `PropertyValue`-Array mit den Anforderungseigenschaften.
 7. Rufen Sie die Methode `updateProperties` mit dem zuvor erzeugten Array auf.

WSDL2Java erstellt die genannten Webservice-Objekte:

- Requirement
- PropertyValue

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

```
/** 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";

// Construct Top Level Requirement
Requirement topLevelRequirement = new Requirement();
topLevelRequirement.setName("tmReqMgt TopLevelReq");
topLevelRequirement.setDescription("tmReqMgt TopLevel Desc");
PropertyValue propRisk = new PropertyValue("1", PROPERTY_RISK, null);
PropertyValue propPriority = new PropertyValue("1", PROPERTY_PRIORITY, null);
PropertyValue[] properties = new PropertyValue[]{propRisk, propPriority};

/* First add requirement skeleton, get its ID
 * mWebService is a binding stub, see above getBinding() snippet
 * mSessionID is the stored session ID, see above login() snippet
 */
String requirementID = mWebService.updateRequirement(mSessionID,
topLevelRequirement, null);
```

```

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

// Add Child Requirement
Requirement childRequirement = new Requirement();
childRequirement.setName("tmReqMgt ChildReq");
childRequirement.setDescription("tmReqMgt ChildLevel Desc");
childRequirement.setParentId(requirementID);
propRisk = new PropertyValue("2", PROPERTY_RISK, null);
propPriority = new PropertyValue("2", PROPERTY_PRIORITY, null);
properties = new PropertyValue[]{propRisk, propPriority};
String childReqID = mWebService.updateRequirement(mSessionID,
    childRequirement, null);
// Now loop through and set properties
for(PropertyValue propValue: properties)
{
    propValue.setRequirementId(requirementID);
    mWebService.updateProperty(mSessionID, childReqID, propValue);
}

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

```

Sitzungen

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

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

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

Die Authentifizierung über Webdienste erfolgt durch den SOAP-Aufruf `logonUser()` des Webdienstes `sccsystem`. Der Methodenaufwurf gibt eine Sitzungs-ID zurück, die zur Identifizierung der auf dem Server generierten Sitzung und zugleich als Schlüssel für den Zugriff auf SilkCentral im Kontext dieser Sitzung dient.

Jeder nachfolgende SOAP-Aufruf, der eine Sitzung zur Ausführung benötigt, übernimmt eine zurückgegebene ID als Parameter, prüft ihre Gültigkeit und wird im Kontext der entsprechenden Sitzung ausgeführt.

Das folgende Java-Codebeispiel zeigt einen einfachen Zugriff auf Test Manager über Webdienste sowie die Verwendung der Sitzungs-ID:

```

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

```

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

Auf der Website <http://sso.borland.com> in dem Abschnitt Test Manager steht ein Client für Demonstrationszwecke zum Download bereit. Dieses Demoprojekt verwendet den Ausführungs-Webdienst von Test Manager, der Sie beim Kennenlernen der Webdienst-Schnittstelle unterstützt.

Verfügbare Webdienste

Die folgende Tabelle zeigt die verfügbaren SilkCentral-Webdienste.



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

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in [Javadoc](#). Wenn der Link nicht funktioniert, klicken Sie im Test Manager-Menü auf **Hilfe > Dokumentation > Test Manager API-Spezifikation**, um Javadoc zu öffnen.



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

Name des Webdiensts (Schnittstelle)	WSDL-URL	Beschreibung
sccsystem (SystemServiceI)	/Services1.0/services/sccsystem?wsdl	Der Stammdienst, der für die Authentifizierung sorgt und einfache Dienstmethoden bereitstellt.
sccentities (MainEntities)	/axislegacy/sccentities?wsdl	Dieser Dienst ermöglicht den Zugriff auf die Entitäten <i>Projekt</i> und <i>Produkt</i> auf der SilkCentral-Plattform.
tmrequirementsmanagement (RMservice)	/Services1.0/services/tmrequirementsmanagement?wsdl	Dieser Dienst ermöglicht den Zugriff auf den Bereich Anforderungen von SilkCentral Test Manager.
tmplanning (IPlanningService)	/Services1.0/services/tmplanning?wsdl	Dieser Dienst ermöglicht den Zugriff auf den Bereich Tests von SilkCentral Test Manager. Außerdem erlaubt er das Starten von Ausführungen und das Abrufen von Ausführungsergebnissen.
tmexecution (IExecutionWebservice)	/Services1.0/services/tmexecution?wsdl	Dieser Dienst ermöglicht den Zugriff auf den Bereich Testausführung von SilkCentral Test Manager.
tmfilters (FilterService)	/Services1.0/services/tmfilters?wsdl	Dieser Dienst ermöglicht das Erstellen, Lesen, Aktualisieren und Löschen von Filtern.
scim (RadarServiceIF)	/axislegacy/scim?wsdl	Dieser Dienst ermöglicht den Zugriff auf Issue Manager. .

Services Exchange

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



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

reportData-Schnittstelle

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

Schnittstellen-URL	Parameter	Beschreibung
/servicesExchange? hid=reportData	sid	Sitzungs-ID: Benutzer-Authentifizierung
	reportFilterID	ID des Berichtsfilters
	type	Format des Antwortrumpfs: (csv oder xml)
	includeHeaders	Berichtskopfzeilen einschließen. (true oder false)
	userName	(Optional) Benutzername: Anstelle von sid
	passWord	(Optional) Kennwort: Anstelle von sid

Beispiel für die reportData-Schnittstelle

```
String sessionID;  
String reportID;  
  
URL report = new URL("http", host, port,  
    "/servicesExchange?hid=reportData" +  
    "&type=csv" +  
    "&sid=" + sessionID +  
    "&reportFilterID=" + reportID +  
    "&includeHeaders=true" +  
    "&execNode_Id_0=1");  
  
BufferedReader in =  
    new BufferedReader(  
        new InputStreamReader(  
            report.openStream(), "UTF-8"));
```

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

```
"&parametername=parametervalue"
```

In diesem Beispiel ist der Parameter `execNode_Id_0` auf den Wert 1 gesetzt.

TMAAttach-Schnittstelle

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

Schnittstellen-URL	Parameter	Beschreibung
/servicesExchange? hid=TMAAttach	sid	Sitzungs-ID: Benutzer-Authentifizierung
	entityType	Typ der Zielentität: (Test, Anforderung oder TestStepParent)
	entityID	ID der Zielentität: (Tests-ID, Anforderungs-ID oder ID der manuellen Test)
	description	Beschreibung des Anhangs. URL-codierter Text, der den Anhang beschreibt.
	isURL	Bei true ist der Anhang ein URL. Bei false ist der Anhang eine Datei.
	URL	Optional - Anzuhängender URL.
	stepPosition	Optional – Der Name des Testschritts. Bezeichnet den Schritt eines manuellen Tests (z. B. 1 für den ersten Schritt). Die Reihenfolge ist obligatorisch, wenn entityType den Wert TestStepParent hat.
	userName	(Optional) Benutzername: Anstelle von sid
	password	(Optional) Kennwort: Anstelle von sid

Beispiel für den Webdienst TMAAttach

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

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

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

String 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) ...

```

Schnittstelle createTestPlan

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

Die folgende Tabelle enthält die Parameter der Schnittstelle `createTestPlan`.

Schnittstellen-URL	Parameter	Beschreibung
/servicesExchange? hid=createTestPlan	sid	Sitzungs-ID – Benutzerauthentifizierung
	parentNodeID	ID des Containers, dem der neue Test in der Testhierarchie hinzugefügt wird.
	userName	<i>Optional:</i> Benutzername – Anstelle von sid
	passWord	<i>Optional:</i> Kennwort – Anstelle von sid

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

Beispiel für den Webdienst createTestPlan

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

```

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

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 xmlFormItem = new StringPart("testPlan", 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());

```

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

Testbeispiel

Der folgende Code zeigt ein Beispiel eines Tests, der mithilfe des Dienstes createTestPlan in Test Manager hochgeladen werden kann.

```

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

  <Folder name="Folder1" id="5438">
    <Description>Description of the folder</Description>
    <Property name="property1">
      <propertyValue>value1</propertyValue>
    </Property>
    <Test name="TestDef1" type="plugin.SilkTest">
      <Description>Description of the test</Description>
      <Property name="property2">
        <propertyValue>value2</propertyValue>
      </Property>
      <Property name="property3">
        <propertyValue>value3</propertyValue>
        <propertyValue>value4</propertyValue>
      </Property>
      <Parameter name="param1" type="string">string1</Parameter>
      <Parameter name="param2" type="boolean">true</Parameter>
      <Parameter name="paramDate"
type="date">01.01.2001</Parameter>
      <Parameter name="paramInherited" type="string"
        inherited="true">
        inheritedValue1
      </Parameter>
      <Step id="1" name="StepA">
        <ActionDescription>do it</ActionDescription>
        <ExpectedResult>everything</ExpectedResult>
      </Step>
      <Step id="2" name="StepB">
        <ActionDescription>and</ActionDescription>
        <ExpectedResult>everything should come</ExpectedResult>
      </Step>
    </Test>
  <Test name="ManualTest1" id="5441" type="_ManualTestType"
    plannedTime="03:45">

```

```

<Description>Description of the manual test</Description>
<Step id="1" name="StepA">
  <ActionDescription>do it</ActionDescription>
  <ExpectedResult>everything</ExpectedResult>
</Step>
<Step id="2" name="StepB">
  <ActionDescription>and</ActionDescription>
  <ExpectedResult>everything should come</ExpectedResult>
</Step>
<Step id="3" name="StepC">
  <ActionDescription>do it now</ActionDescription>
  <ExpectedResult>
    everything should come as you wish
  </ExpectedResult>
</Step>
</Test>
<Folder name="Folder2" id="5439">
  <Description>Description of the folder</Description>
  <Property name="property4">
    <propertyValue>value5</propertyValue>
  </Property>
  <Parameter name="param3" type="number">123</Parameter>
  <Folder name="Folder2_1" id="5442">
    <Description>Description of the folder</Description>
    <Test name="TestDef2" type="plugin.SilkPerformer">
      <Description>Description of the test</Description>
      <Property name="_sp_Project File">
        <propertyValue>ApplicationAccess.ltp</propertyValue>
      </Property>
      <Property name="_sp_Workload">
        <propertyValue>Workload1</propertyValue>
      </Property>
    </Test>
    <Test name="TestDef3" type="JUnitTestType"
      externalId="com.borland.MyTest">
      <Description>Description of the test</Description>
      <Property name="_junit_ClassFile">
        <propertyValue>com.borland.MyTest</propertyValue>
      </Property>
      <Property name="_junit_TestMethod">
        <propertyValue>testMethod</propertyValue>
      </Property>
      <Step id="1" name="StepA">
        <ActionDescription>do it</ActionDescription>
        <ExpectedResult>everything</ExpectedResult>
      </Step>
      <Step id="2" name="StepB">
        <ActionDescription>and</ActionDescription>
        <ExpectedResult>everything should come</
ExpectedResult>
      </Step>
    </Test>
  </Folder>
</Folder>
</Folder>
</TestPlan>

```

Schnittstelle exportTestPlan

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

Schnittstellen-URL	Parameter	Beschreibung
/servicesExchange? hid=exportTestPlan	sid	Sitzungs-ID – Benutzerauthentifizierung
	nodeID	Der Knoten mit dieser ID und rekursiv alle untergeordneten Knoten werden exportiert.
	userName	<i>Optional:</i> Benutzername – Anstelle von sid
	passWord	<i>Optional:</i> Kennwort – Anstelle von sid

Beispiel für den Webservice exportTestPlan

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

```
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().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);
```

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

Schnittstelle updateTestPlan

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

Die folgende Tabelle enthält die Parameter der Schnittstelle `updateTestPlan`.

Schnittstellen-URL	Parameter	Beschreibung
/servicesExchange? hid=updateTestPlan	sid	Sitzungs-ID – Benutzerauthentifizierung
	userName	<i>Optional:</i> Benutzername – Anstelle von sid

Schnittstellen-URL	Parameter	Beschreibung
	passWord	<i>Optional:</i> Kennwort – Anstelle von sid

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

Beispiel für den Webservice updateTestPlan

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

```
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();
```

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

Schnittstelle createRequirements

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

Die folgende Tabelle enthält die Parameter der Schnittstelle `createRequirements`.

Schnittstellen-URL	Parameter	Beschreibung
/servicesExchange? hid=createRequirements	sid	Sitzungs-ID – Benutzerauthentifizierung
	parentNodeID	ID des Containers, dem die neue Anforderung in der

Schnittstellen-URL	Parameter	Beschreibung
		Anforderungshierarchie hinzugefügt wird.
	userName	<i>Optional:</i> Benutzername – Anstelle von sid
	passWord	<i>Optional:</i> Kennwort – Anstelle von sid

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

Beispiel für den Webdienst createRequirements

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

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

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

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

Schnittstelle exportRequirements

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

Schnittstellen-URL	Parameter	Beschreibung
<code>/servicesExchange?hid=exportRequirements</code>	sid	Sitzungs-ID – Benutzerauthentifizierung

Schnittstellen-URL	Parameter	Beschreibung
	nodeID	Der Knoten mit dieser ID und rekursiv alle untergeordneten Knoten werden exportiert.
	userName	<i>Optional:</i> Benutzername – Anstelle von sid
	passWord	<i>Optional:</i> Kennwort – Anstelle von sid

Beispiel für den Webdienst exportRequirements

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

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

long sessionId = mWebServiceHelper.getSessionId();

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

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

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

Schnittstelle updateRequirements

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

Die folgende Tabelle enthält die Parameter der Schnittstelle `updateRequirements`.

Schnittstellen-URL	Parameter	Beschreibung
/servicesExchange? hid=updateRequirements	sid	Sitzungs-ID – Benutzerauthentifizierung
	userName	<i>Optional:</i> Benutzername – Anstelle von sid

Schnittstellen-URL	Parameter	Beschreibung
	passWord	<i>Optional:</i> Kennwort – Anstelle von sid
	NodeID	Der Knoten mit dieser ID und rekursiv alle untergeordneten Knoten werden aktualisiert.

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

Beispiel für den Webdienst updateRequirements

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

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

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

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

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

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

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

Schnittstelle updateRequirementsByExternalID

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

Die folgende Tabelle enthält die Parameter der Schnittstelle `updateRequirementsByExternalID`.

Schnittstellen-URL	Parameter	Beschreibung
/servicesExchange? hid=updateRequirements	sid	Sitzungs-ID – Benutzerauthentifizierung
	nodeID	Die ID des zu aktualisierenden Knotens der Anforderungshierarchie.
	userName	<i>Optional:</i> Benutzername – Anstelle von sid
	passWord	<i>Optional:</i> Kennwort – Anstelle von sid

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

Beispiel für den Webservice `updateRequirementsByExternalID`

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

```
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",
        "updateRequirementsByExternalID",
        sessionID));

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

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

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

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

Schnittstelle `createExecutionDefinitions`

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

Die folgende Tabelle enthält die Parameter der Schnittstelle `createExecutionDefinitions`.

Schnittstellen-URL	Parameter	Beschreibung
/servicesExchange? hid=createExecutionDefinitions	sid	Sitzungs-ID – Benutzerauthentifizierung
	parentNodeID	ID des Knotens, dem die neue Testsuite in der Testsuitehierarchie hinzugefügt wird.
	userName	<i>Optional:</i> Benutzername – Anstelle von sid
	passWord	<i>Optional:</i> Kennwort – Anstelle von sid

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

Beispiel für den Webdienst createExecutionDefinitions

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

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

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());
```

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

Ausführungsplan – Beispiel

Der folgende Code zeigt ein Beispiel eines Ausführungsplans, der mithilfe des Diensts createExecutionDefinitions in Test Manager hochgeladen werden kann. In diesem Fall wird ein benutzerdefinierter Ausführungstermin für eine der Testsuiten

erstellt, und einer Testsuite werden Tests zugeordnet, sowohl manuell als auch mittels Filter. Im Beispiel wird zudem eine Konfigurationssuite mit Konfigurationen erstellt.

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

  <Folder name="Folder1">
    <Description>Description of the folder</Description>
    <ExecDef name="ExecutionDefinition1" TestContainerId="1">
      <Description>Description1</Description>
      <CustomSchedule>
        <start>2009-11-26T21:32:52</start>
        <end>
          <forever>true</forever>
        </end>
        <Interval day="1" hour="2" minute="3"></Interval>
        <adjustDaylightSaving>false</adjustDaylightSaving>
        <exclusions>
          <days>Monday</days>
          <days>Wednesday</days>
          <from>21:32:52</from>
          <to>22:32:52</to>
        </exclusions>
        <definiteRun>2009-11-27T21:35:12</definiteRun>
      </CustomSchedule>
      <ReadFromBuildInfoFile>true</ReadFromBuildInfoFile>
      <Priority>High</Priority>
      <SetupTestDefinition>73</SetupTestDefinition>
      <CleanupTestDefinition>65</CleanupTestDefinition>
      <AssignedTestDefinitions>
        <ManualAssignment useTestPlanOrder="true">
          <TestId>6</TestId>
          <TestId>5</TestId>
        </ManualAssignment>
      </AssignedTestDefinitions>
    </ExecDef>
    <ExecDef name="ExecutionDefinition2" TestContainerId="1">
      <Description>Description2</Description>
      <Build>1</Build>
      <Version>1</Version>
      <Priority>Low</Priority>
      <SourceControlLabel>Label1</SourceControlLabel>
      <DependentExecDef id="65">
        <Condition>Passed</Condition>
        <Deployment>
          <Specific>
            <Execution type="Server" id="1"/>
            <Execution type="Tester" id="0"/>
          </Specific>
        </Deployment>
      </DependentExecDef>
      <DependentExecDef id="70">
        <Condition>Failed</Condition>
        <Deployment>
          <Specific>
            <Execution type="Tester" id="0"/>
          </Specific>
        </Deployment>
      </DependentExecDef>
      <DependentExecDef id="68">
```

```

    <Condition>Any</Condition>
    <Deployment>
      <UseFromCurrentExedDef>true</UseFromCurrentExedDef>
    </Deployment>
  </DependentExecDef>
</ExecDef>

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

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

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

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

```

Schnittstelle exportExecutionDefinitions

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

Schnittstellen-URL	Parameter	Beschreibung
/servicesExchange? hid=exportExecutionDefinitions	sid	Sitzungs-ID – Benutzerauthentifizierung
	nodeID	Der Knoten mit dieser ID und rekursiv alle untergeordneten Knoten werden exportiert.
	userName	<i>Optional:</i> Benutzername – Anstelle von sid
	passWord	<i>Optional:</i> Kennwort – Anstelle von sid

Beispiel für den Webdienst exportExecutionDefinitions

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

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

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

Schnittstelle updateExecutionDefinitions

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

Die folgende Tabelle enthält die Parameter der Schnittstelle `updateExecutionDefinitions`.

Schnittstellen-URL	Parameter	Beschreibung
/servicesExchange? hid=updateExecutionDefinitions	sid	Sitzungs-ID – Benutzerauthentifizierung
	userName	<i>Optional:</i> Benutzername – Anstelle von sid

Schnittstellen-URL	Parameter	Beschreibung
	passWord	Optional: Kennwort – Anstelle von sid

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

Beispiel für den Webservice `updateExecutionDefinitions`

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

```
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();
```

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

Schnittstelle `createLibraries`

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

Die folgende Tabelle enthält die Parameter der Schnittstelle `createLibraries`.

Schnittstellen-URL	Parameter	Beschreibung
<code>/servicesExchange?hid=createLibraries</code>	sid	Sitzungs-ID – Benutzerauthentifizierung

Schnittstellen-URL	Parameter	Beschreibung
	userName	<i>Optional:</i> Benutzername – Anstelle von sid
	passWord	<i>Optional:</i> Kennwort – Anstelle von sid

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

Beispiel für den Webservice createLibraries

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

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

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

Bibliotheken – Beispiel

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

```
<?xml version="1.0" encoding="UTF-8"?>
<LibraryStructure xmlns="http://www.borland.com/TestPlanSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.borland.com/TestPlanSchema
  ../../../../../../TM/wwwroot/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>
    </Step>
    <Step name="Create User">
        <ActionDescription>Create user tester</
ActionDescription>
        <ExpectedResult>User created</ExpectedResult>
    </Step>
    <Step name="Logout">
        <ActionDescription>
            Logout using start menu
        </ActionDescription>
        <ExpectedResult>Logged out.</ExpectedResult>
    </Step>
    </SharedSteps>
</Folder>
</Folder>
<GrantedProjects>
    <ProjectId>0</ProjectId>
    <ProjectId>1</ProjectId>
</GrantedProjects>
</Library>
</LibraryStructure>

```

Schnittstelle exportLibraryStructure

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

Schnittstellen-URL	Parameter	Beschreibung
/servicesExchange? hid=exportLibraryStructure	sid	Sitzungs-ID – Benutzerauthentifizierung
	userName	<i>Optional:</i> Benutzername – Anstelle von sid
	passWord	<i>Optional:</i> Kennwort – Anstelle von sid
	nodeID	Der zu exportierende Bibliotheksknoten oder -ordner in der Bibliothekshierarchie. IDs von Knoten mit gemeinsam verwendbaren Testschritten sind unzulässig.

Beispiel für den Webdienst exportLibraryStructure

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

```

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

```

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

Schnittstelle exportLibraryStructureWithoutSteps

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

Schnittstellen-URL	Parameter	Beschreibung
/servicesExchange? hid=exportLibraryStructureWithoutSteps	sid	Sitzungs-ID – Benutzerauthentifizierung
	userName	<i>Optional:</i> Benutzername – Anstelle von sid
	passWord	<i>Optional:</i> Kennwort – Anstelle von sid
	nodeID	Der zu exportierende Bibliotheksknoten oder -ordner in der Bibliothekshierarchie. IDs von Knoten mit gemeinsam verwendbaren Testschritten sind unzulässig.

Beispiel für den Webdienst exportLibraryStructureWithoutSteps

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

```

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);
```

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

Schnittstelle getLibraryInfoByName

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

Schnittstellen-URL	Parameter	Beschreibung
/servicesExchange? hid=exportLibraryStructureWithoutSteps	sid	Sitzungs-ID – Benutzerauthentifizierung
	userName	<i>Optional:</i> Benutzername – Anstelle von sid
	passWord	<i>Optional:</i> Kennwort – Anstelle von sid
	libraryName	Der Name der Bibliothek

Beispiel für den Webdienst getLibraryInfoByName

Im folgenden Quelltext werden Bibliotheksinformationen mithilfe von Apache HttpClient abgerufen.

```
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",
    "exportLibraryStructureWithoutSteps", sessionID,
    LIBRARY_NAME));

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 response = fileGet.getResponseBodyAsString();
System.out.println(response);
```

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

Demo Client für Webdienste

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

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



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

Index

-Webdienste
Anwendungsfallbeispiel 43

A

Allgemeine Metadaten für Eigenschaften
Drittanbieter-Testtyp-Plug-In 35
Anforderungs-Plug-Ins 29
Apache Axis 39
API-Struktur
Drittanbieter-Testtyp-Plug-In 31
APIs
Integration Codeabdeckung 7
Arten
Plug-Ins 6
Authentifizierung
Webdienst 44

B

Beispielcode
Drittanbieter-Testtyp-Plug-Ins 32
Benutzerdaten 44
Benutzerdefinierte Symbole
Drittanbieter-Testtyp-Plug-In 36
Bereitstellung
Drittanbieter-Testtyp-Plug-In 37
Plug-Ins 6

C

Codeabdeckung
APIs 7
createExecutionDefinitions
Beispiele 56
Schnittstellen 56
createLibraries
Beispiele 61
Schnittstellen 61
createRequirements
Beispiele 52
Schnittstellen 52
createTestPlan
Beispiele 48
Schnittstellen 48

D

Demo Client
Webdienst-Schnittstelle 65
Demo Client für Webdienste 65
Drittanbieter-Testtyp-Plug-In
Allgemeine Metadaten für Eigenschaften 35
API-Struktur 31
Benutzerdefinierte Symbole 36
Implementierung 30

Integration 30
Metadaten für Dateieigenschaften 36
Metadaten für String-Eigenschaften 35
XML-Konfigurationsdatei 35, 37
Drittanbieter-Testtyp-Plug-Ins
Beispielcode 32
Komprimierung 30
Metadaten 35
Übergabe von vordefinierten Parametern 31

E

exportExecutionDefinitions
Beispiele 59
Schnittstellen 59
exportLibraryStructure
Beispiele 63
Schnittstellen 63
exportLibraryStructureWithoutSteps
Beispiele 64
Schnittstellen 64
exportRequirements
Beispiele 53
Schnittstellen 53
exportTestPlan
Beispiele 50
Schnittstellen 50

F

Fehlerverfolgung
Plug-Ins 18

G

getLibraryInfoByName
Beispiele 65
Schnittstellen 65

I

Implementierung
Drittanbieter-Testtyp-Plug-In 30
Integration
Drittanbieter-Testtyp-Plug-In 30
Integration der Anforderungsverwaltung 29
Integration der Fehlerverfolgung
Übersicht 18

J

Java API-Schnittstelle 27

K

Klassen 27
Kompilierung von Plug-Ins 6

Komprimierung
Drittanbieter-Testtyp-Plug-Ins 30

M

Metadaten
Drittanbieter-Testtyp-Plug-Ins 35
Metadaten für Dateieigenschaften
Drittanbieter-Testtyp-Plug-In 36
Metadaten für String-Eigenschaften
Drittanbieter-Testtyp-Plug-In 35

P

Plug-Ins
Anforderungen 29
Anforderungsmanagement 29
Arten 6
Bereitstellung 6
Fehlerverfolgung 18
Kompilierung 6
Übersicht 6
Versionsverwaltung 16
Verteilung 6
Process Executor
Beispielcode 32

R

reportData
Beispiel 46
Schnittstelle 46

S

Schnittstellen
createExecutionDefinitions 56
createLibraries 61
createRequirements 52
createTestPlan 48
exportExecutionDefinitions 59
exportLibraryStructure 63
exportLibraryStructureWithoutSteps 64
exportRequirements 53
exportTestPlan 50
getLibraryInfoByName 65
Java API-Schnittstelle 27
reportData 46
TMAAttach 47
updateExecutionDefinitions 60
updateRequirements 54
updateRequirementsByExternalID 55
updateTestPlan 51
Versionsverwaltung 16
Webdienst 18
Services Exchange 46
Sitzungen 44
SOAP

Pakete 40
Stack 39
Symbole
Benutzerdefiniert 36
Synchronisierung
requirements 29

T

TMAAttach
Beispiel 47
Schnittstelle 47

U

updateExecutionDefinitions
Beispiele 60
Schnittstellen 60
updateRequirements
Beispiele 54
Schnittstellen 54
updateRequirementsByExternalID
Beispiele 55
Schnittstellen 55
updateTestPlan
Beispiele 51
Schnittstellen 51

V

Versionsverwaltung
Integration 16
Plug-Ins 16
Schnittstellen 16
Schnittstellen-Konventionen 17
Verteilung
Plug-Ins 6
Videoaufnahme
Angaben des Starts 37
Angaben des Stopps 37
Vordefinierte Parameter
Übergabe an Drittanbieter-Testtyp-Plug-Ins 31

W

Webdienst
Anforderungsmanagement 29
Konfigurieren von Profilen 27
Schnittstelle 18
Übersicht 39
Verfügbar 45
Voraussetzungen 39
WSDL-Spezifikation 21
Webdienst-Schnittstelle
Kurzanleitung 39
Tutorial 40
WSDL-Spezifikation 21