Webservices mit ADITO4

AID 059 DE

Autor: FA, KN, MW. Version 10.2. Zuletzt geändert 05.09.2017

Version	Änderungen
10.2	Anpassung der Formatierungen
10.1	Hinweis auf Slashes bei Webservices, die von ADITO angeboten werden, Beispielprozess für POST hinzugefügt
10.0	Anpassung an ADITO4.6
7.4	Hinweis, dass bei der Verwendung von REST-Webserivces bei Veröffentlichung von JDito-Prozessen zwingend ein Benutzerkonto angegeben werden muss.
7.3	Kleinere Korrekturen
7.2	Client-Webservice hinzugefügt
7.1	Letzte Version vor Übernahme in Versionshistorie

Inhaltsverzeichnis

1. Webservices in ADITO4

Die ADITO Webservice-Schnittstellen ermöglichen die Kommunikation von über das Internet bereitgestellten Services mit dem ADITO-Server und dem ADITO-Client.

ADITO unterstützt

- **SOAP** Webservices (https://de.wikipedia.org/wiki/SOAP)
- **REST**-Calls (RESTful Webservices, https://de.wikipedia.org/wiki/Representational_State_Transfer)

1.1. SOAP

Bei ADITO4 ist grundsätzlich zwischen ein- und ausgehenden Webservices zu unterscheiden. Bei eingehenden Webservices können Anwendungen mit ADITO4 kommunizieren; hier tritt ADITO4 als Web-Service-Server auf. Bei ausgehenden Webservices kommuniziert ADITO4 mit einer anderen Anwendung, die auf der bestehenden PlugIn-Systematik basiert.

Lesen Sie hierzu auch das AID004-DE - PlugIns für ADITO.

Es gibt drei Möglichkeiten, ADITO4 mit Webservices interagieren zu lassen:

1. Webservice → ADITO, clientseitig. (Mit WSDL-Dateien: „webserviceClient“.)
2. Webservice → ADITO, serverseitig. (Mit WSDL-Dateien, „webserviceServer“.) Diese Webservices sind deprecated, sollten also zukünftig nicht mehr verwendet werden.
3. Webservice → ADITO, serverseitig. (Mit WSDL-Dateien, „jditoWebservice“.)
4. ADITO → Webservice, über ein PlugIn. (WSDL oder ASMX)

Die WSDL-Datei hält alle Informationen vor, die den Webservice auszeichnen. Es handelt sich um eine XML-Struktur. In ihr ist beispielsweise definiert, welche Prozesse im Webservice aufgerufen werden können, welche Eingabeparameter welchen Typs die Prozesse benötigen und dergleichen mehr.

1.2. REST

REST-Calls werden über die JDito-Methoden `net.callRestWebservice*` aufgerufen. ADITO unterstützt dabei die Funktionen `GET`, `POST`, `PUT` und `DELETE`. ADITO kann Webservices auch über REST-Aufrufe anbieten.

1.3. Client-Webservices

Client-Webservices werden wie auch Server-Webservices über `webserviceEnabled` in der Gruppe `Server` aktiviert.

Diese können Sie dort konfigurieren, wo auch Server-Webservices konfiguriert werden:

- JDito-Webservices: NUR Server (SOAP oder REST) Webservices
  (ADITO ist Webservice Provider)
- Server-Webservices: SOAP-Webservices mit run-process, ansprechbar über beliebige URL
- Client-Webservices: SOAP-Webservices mit run-process, ansprechbar über localhost; kann auch Client-Methoden ansprechen

Bei allen Prozessen, bei denen Server- (und NICHT Jdito-) Webservices aktiviert sind, können auch die Client-Webservices aktiviert werden.

Um Client-Webservices zu deaktivieren, muss beim Start des Clients folgender Parameter angeben werden:

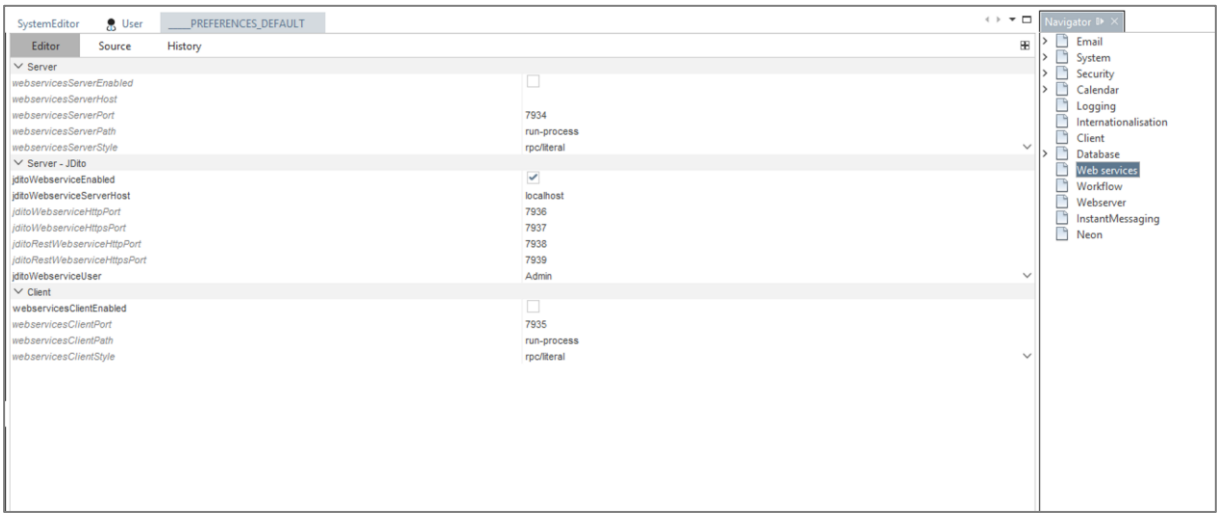
```
-Dadito.client.disable.webservices=true
```

2. SOAP Webservices

2.1. Eingehende Webservices

Bei eingehenden Webservices nehmen Sie sowohl deren server- als auch clientseitige Kommunikation mit ADITO4 vor. Dies geschieht im `Preferences`-Reiter des ADITO4 Designers im System-Editor über die Auswahl `Webservice` im Navigator.

Nachdem Sie diese Eigenschaften angepasst haben, müssen Sie deployen und den ADITO4 Server neu starten!



Welche Maßnahmen zur Konfiguration eingehender Webservices vorgenommen werden müssen, lesen Sie weiter unten.

Damit Webservices mit ADITO4 interagieren können, müssen folgende Voraussetzungen gegeben sein:

webservicesClient

- Der Webservice muss in der Server Config grundsätzlich aktiviert sein.
- Es muss in der `AliasDefinition` im Reiter `User` des Designers ein Benutzer angelegt werden, der als Anmelde-User bei den Webservices dient (nur `webserviceClient` und `webserviceServer`).
- Mindestens ein Prozess unter `Process` muss Client-Webservices zur Verfügung stehen.

webservicesJDito

- Der Webservice muss in der Server Config grundsätzlich aktiviert sein.
- JDito-Webservices können derzeit ausschließlich anonym angesprochen werden.
- Mindestens ein Prozess unter Process muss JDito-Webservices zur Verfügung stehen.

webservicesServer

- Diese Webservices sind deprecated und sollten, wenn möglich, nicht mehr verwendet werden.

Der Unterschied zwischen server- und clientseitigen Webservices ist der, dass Sie auf der Clientseite alle JDito-Methoden zur Verfügung haben, die Ihnen auch in Client-Prozessen zur Verfügung stehen. Damit können Sie beispielsweise einen Frame öffnen oder einen Datensatz aufrufen (`swing.doClientIntermediate`, `swing.doAction` etc.).

2.2. webservicesServer / webservicesClient

2.2.1. Beispiel 1: Mit PHP einen Webservice ansprechen

Das folgende Beispiel kommuniziert nach Eingabe von Daten in einer Website mit ADITO. Nach Verarbeitung in einem ADITO-Prozess werden die Daten zurück an die Webseite übertragen.

2.2.1.1. Vorbereiten der Webservices

Webservices werden in der Serverkonfiguration `server Config` aktiviert. Dazu wechseln Sie im ADITO4 Designer im Navigator auf den Menüpunkt `Web Services`. In diesem Fall benötigen wir einen serverseitigen Webservice:

- `Enable`: Aktiviert oder deaktiviert Webservices.
- `Address`: Legt fest, unter welcher Adresse der Webservice angesprochen werden kann. Diese Einstellung ist nur bei serverseitigen Webservices möglich.
- `Port`: Legt fest, unter welchem Port der Webservice angesprochen werden kann.
- `Path`: Pfad zum Webservice, dieser kann frei gewählt werden.

Nachdem Sie die Webservices aktiviert haben, müssen Sie den Server neu starten. Bei Erfolg gibt der Server beim Start eine entsprechende Meldung an die Log-Ausgabe.

```
H-44-Z-0022-S Web Services gestartet.  [->] Address:
http://127.0.0.1:7937/run-process
```

Über die hier angegebene Adresse können Sie nun auch die Verfügbarkeit des Webservices testen. Wenn Sie die genannte Adresse mit dem Zusatz „`?wsdl`“ in den Webbrowser eingeben, können Sie Einblick in die WSDL-Datei des ADITO-Webservices erhalten.

```
http://127.0.0.1:7937/run-process?wsdl
```

```
<!--
    Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2.4-b01.
-->
<!--
    Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2.4-b01.
-->
<definitions targetNamespace="http://adito.de/webservice/" name="AditoService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://adito.de/webservice/" schemaLocation="http://127.0.0.1:7937/run-process?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="runprocess">
    <part name="name" type="xsd:string"/>
    <part name="parameter" type="xsd:string"/>
    <part name="user" type="xsd:string"/>
    <part name="password" type="xsd:string"/>
    <part name="concurrent" type="xsd:boolean"/>
  </message>
  <message name="runprocessResponse">
    <part name="result" type="xsd:string"/>
  </message>
  <message name="AditoWebserviceException">
    <part name="fault" element="tns:AditoWebserviceException"/>
  </message>
  <portType name="Adito">
    <operation name="runprocess" parameterOrder="name parameter user password concurrent">
      <input wsam:Action="http://adito.de/webservice/Adito/runprocessRequest" message="tns:runprocess"/>
      <output wsam:Action="http://adito.de/webservice/Adito/runprocessResponse" message="tns:runprocessResponse"/>
      <fault message="tns:AditoWebserviceException" name="AditoWebserviceException" wsam:Action="http://adito.de/webservice/Adito
      /runprocess/Fault/AditoWebserviceException"/>
    </operation>
  </portType>
  <binding name="AditoPortBinding" type="tns:Adito">
```

Die WSDL (Web Service Description Language)-Datei gibt alle Funktionen, Daten, Datentypen und Austauschprotokolle eines Webservice an. Über den Aufruf der Datei im Webbrowser wird signalisiert, dass der Webservice grundsätzlich zur Verfügung steht.

2.2.1.2. Anlegen eines Webservice-Benutzers

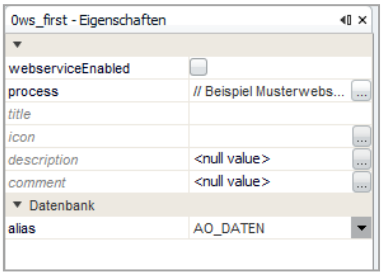
Um den Zugriff auf den Webservice zu ermöglichen, muss sich ein Aufrufer mit Benutzernamen und Passwort anmelden. In ADITO4 legen Sie den Web Service befähigten Benutzer in der `aliasConfig` unter `User` des ADITO4 Designers an.

Sie benötigen hier keinen Mitarbeiter mit Kontaktperson und Zuordnung zu Gruppen. Ein einfacher Benutzer ohne Zuordnung genügt vollkommen.

Um einen so neu angelegten Benutzer mit den Webservices zu verbinden, muss (für ADITO 4) in den erweiterten Eigenschaften des Benutzers `Webservice enabled` auf `Ja` gestellt sein.

In ADITO 4 muss dieser Benutzer die Rolle Webservices besitzen. Diese kann nur im `User`-Editor des Designers vergeben werden.

2.2.1.3. Webservice-fähigen Prozess anlegen



Um einen Prozess aus der Prozessbibliothek für Webservices ansprechbar zu machen, muss die Eigenschaft `Webservice` bestätigt werden. Dies erfolgt über eine Checkbox in den Eigenschaften des Prozesses im ADITO4 Designer.

Als Beispiel soll folgender Prozess wsproc dienen (verwendete Datenbank:SQL-Server):

```
import("system.vars");
import("system.logging");
import("system.db");
import("system.result");

var parm = vars.getString("$local.param");
logging.log("*** Webservice: Parameter erhalten: " + parm);

var orglist = db.array(db.COLUMN, "select orgid from org where
upper(orgname) like upper ('%" + parm + "%')");
logging.log("Länge orglist: " + orglist.length);

var orgs = "";
for ( var i = 0; i < orglist.length; i++)
{
 var sql = "select orgname + '<br/>' + address + ' ' + buildingno +
'<br/>' + zip + ' ' + city "
 + "from org "
 + "join relation on org_id = orgid "
 + "join address on relation_id = relationid "
 + "where pers_id is null and orgname <> 'privat' "
 + "and orgid = '" + orglist[i] + "'";
  orgs += db.cell(sql) + "<br/><br/>";
}
result.string(orgs);
```

Der ADITO Webservice ist in der Lage, einen Eingabeparameter zu verarbeiten. Dieser Parameter wird als einfache Zeichenkette (String) behandelt und steht über die lokale JDito-Variable `$local.param` zur Verfügung.

Wenn Sie ein Array oder mehrstufige Daten an den Webservice senden möchten, können Sie die Daten auch XML-formatiert in der Zeichenkette übergeben.

ADITO kann auch einen Wert an den Webservice zurück liefern. Diesen können Sie einfach über eine der `result.*`-Methoden zurückgeben. Dieser Rückgabewert kann in diesem Beispiel vom PHP-Script weiterverarbeitet werden.

2.2.1.4. Mit dem Webservice kommunizieren

Alle Voraussetzungen für die Kommunikation eines Scripts mit ADITO4 über Webservices sind nun erfüllt. Mit Hilfe einer einfachen HTML-Seite, die PHP-Code enthält, können Sie nun mit ADITO4 kommunizieren:

```
<html>
<head>
<meta content="text/html; charset=iso-8859-1" http equiv="Content-
Type" />
 <title>Webservice</title>
 </head>
 <body>
 <form method="post">
  <input name='orgname' type='text' />
  <button name='okbutton'>OK</button>
 </form>
 </body>


 <?php
 $wsdl="http://127.0.0.1:7937/run-process?wsdl";
 $client=new SoapClient($wsdl, array('encoding'=>'ISO-8859-1'));
 echo $client->runprocess('wsproc', $_POST['orgname'], 'wsuser',
'a', false);
 ?>
</html>
index.php
```

Im PHP-Teil dieser Seite definieren Sie die Kommunikation mit dem Webservice. In jeder Technologie, die mit Webservices kommunizieren kann, stecken folgende Elemente:

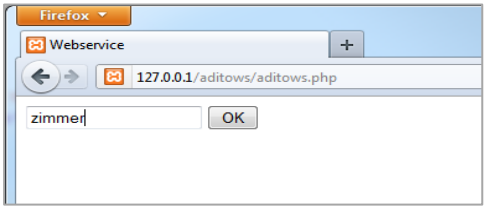
- **Definition der WSDL-Adresse**: Es ist immer notwendig die genaue Adresse anzugeben, unter der die WSDL-Spezifikationen des Webservices zur Verfügung stehen. In diesem Fall wird sie bei der Definition der WSDL-Variable eingetragen:

  ```
  $wsdl="http://127.0.0.1:7937/run-process?wsdl";
  ```

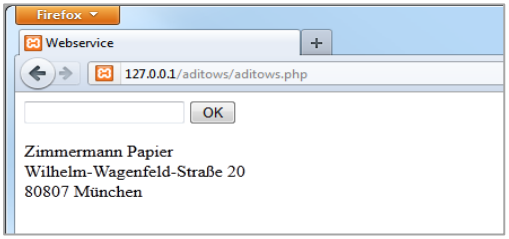
- **Übergabe der Parameter und Aufruf des Prozesses**: Der den Webservice ausführende Prozess hat im ADITO4 System immer den Namen `runprocess` (nicht zu verwechseln mit dem Pfad zur WSDL-Datei, hier `run-process`). Nach Angabe dieses Prozessnamens erwartet die ADITO4 Webservice-Schnittstelle fünf Parameter, hier übergeben als Array:
    - Den Namen des JDito-Prozesses, der aufgerufen wird (`wsproc`).
    - Der Parameter, der im JDito-Prozess unter der Variable `$local.parms` zur Verfügung steht. Hier der Wert aus einem HTML-Formular.
    - Benutzername und
    - Passwort des mit `webservice enabled (4)` angegebenen Benutzers.
    - Der letzte Parameter dient der Rückgabedefinition. Wenn `true` angegeben ist, wartet der Webservice explizit auf einen Rückgabewert des angesprochenen Prozesses. Bei `false` wird auf diese Angabe verzichtet.

2.2.1.5. Ergebnis

Das eben bearbeitete Beispiel ermöglicht es, in einer Website nach Firmenadressen zu suchen und diese in der HTML-Datei anzuzeigen:



Nach Klick auf OK:



Zum Testen der PHP-Seite benötigen Sie einen lokalen Testserver (wie z.B. XAMPP).

2.2.2. Beispiel 2: Einen Webservice mit Java ansprechen

Um einen Webservice mit einer Java-Entwicklungsumgebung anzusprechen, benötigen Sie nur das JDK (Java Development Kit) von Java. Auf Basis der WSDL-Datei (bzw. der URL zur WSDL-Datei) können Sie Proxy-Klassen erstellen, mit deren Hilfe Sie dann mit dem Webservice kommunizieren können.

Um diese Proxy-Klassen zu erstellen, verwenden Sie das mit dem JDK mitgelieferte Tool `wsimport`. Der Aufruf:

```
wsimport -keep http://127.0.0.1:7934/run-process?wsdl
```

Der Parameter `-keep` sorgt dafür, dass die Proxy-Klassen nach dem Aufruf des Tools weiterhin erhalten bleiben. Sie finden danach folgende Klassen vor:

```
Run-process.java
Run-processService.java
```

Wenn Sie diese Klassen in Ihrem Java-Projekt importieren bzw. alle Klassen in einem Java-Package zusammenführen, können Sie nun den Aufruf des Webservices wie im PHP-Beispiel durchführen. Das folgende Beispiel ruft den oben genannten Webservice in einer Java-Klasse auf, die zusammen mit den ADITO-Klassen in einem Projekt liegt:

```
package de.adito.webservice;
public class AditoWS
{

 /**
  * @param args
  */
 public static void main(String[] args)
 {

 AditoService service = new AditoService();
        Adito aditows = service.getAditoPort();

        System.out.println("Start Webservice");


        String res = "zimmer";

        System.out.println("Uebergabe Parameter: " + res);


        try
 {
 System.out.println("Rueckgabe: " +
aditows.runprocess("0ws_test", res, "daWSuser", "a",
```

```
false));
 System.out.println("Ende Webservice");
 }

 catch (AditoWebserviceException e)
 {
  e.printStackTrace();
 }
 }

}
```

Das Ergebnis der Konsolenausgabe ist dann:

```
Zimmermann Papier
Ascherstraße 5 - 14
84144 Geisenhausen
```

2.2.3. Beispiel 3: Webservices von Microsoft Office aus ansprechen

Mit Hilfe von VBA, der Anpassungssprache von Microsoft Office, lassen sich auch Webservices ansprechen. Sie können hiermit beispielsweise eine Schaltfläche anlegen, die nach Eingabe einer Firma oder eines Teils davon die gesamte Adresse in die aktive Zelle einfügt:

```
Sub CallServerWS()
    Dim addIn As COMAddIn
    Dim automationbject As Object
    Set addIn = Application.COMAddIns("AditoWebServicesAddIn")
    Set automationObject = addIn.Object
    automationObject.setUrl ("http://localhost:7770/run-process")
    Dim rueckgabe As String
    rueckgabe =
automationObject.runAditoWebServices("_server_wstest",
    ActiveCell.Value, "admin", "a", False)
    ActiveCell.Value = rueckgabe
End Sub
```

Mit einem speziellen Webservice lassen sich die aus diesem Sub übergebenen Daten auslesen:

```
import("lib_address");
import("system.vars");
import("system.db");
import("system.result");
```

```
var param = vars.getString("$local.param");
var relationid = db.cell("select RELATIONID from RELATION join ORG
on ORGID = ORG_ID and ADDR_TYPE = 1 where UPPER(ORGNAME) like
UPPER('%" + param + "%')");
var ret = param;

if( relationid != "")
{
 ret = new AddrObject( relationid ).formatAddress();
}
result.string(ret);
```

2.3. webservicesJDito

JDito-Webservices sind serverseitige Webservices, die von JDito-Prozessen zur Verfügung gestellt werden. Im Gegensatz zu webservicesServer sind die WSDL-Dateien anders aufgebaut, jede Funktion eines JDito-Prozesses wird hier als eigener Knoten zur Verfügung gestellt.

2.3.1. Beispiel: Einen Webservice mit SoapUI ansprechen.

Folgender Prozess soll per Webservice angesprochen werden. Der Name des Prozesses ist `playgroundpro`:

```
function quak(pInput)
{
    return "Ergebnis: " + pInput;
}
```

2.3.1.1. Vorbereiten des Webservices

Um einen JDito-Webservice zu aktivieren, muss der entsprechende Knoten in den Eigenschaften des Prozesses aktiviert sein. Dieser Prozess muss dabei in einzelne JavaScript-Funktionen aufgeteilt werden.

Folgende Optionen können Sie hier aktivieren:

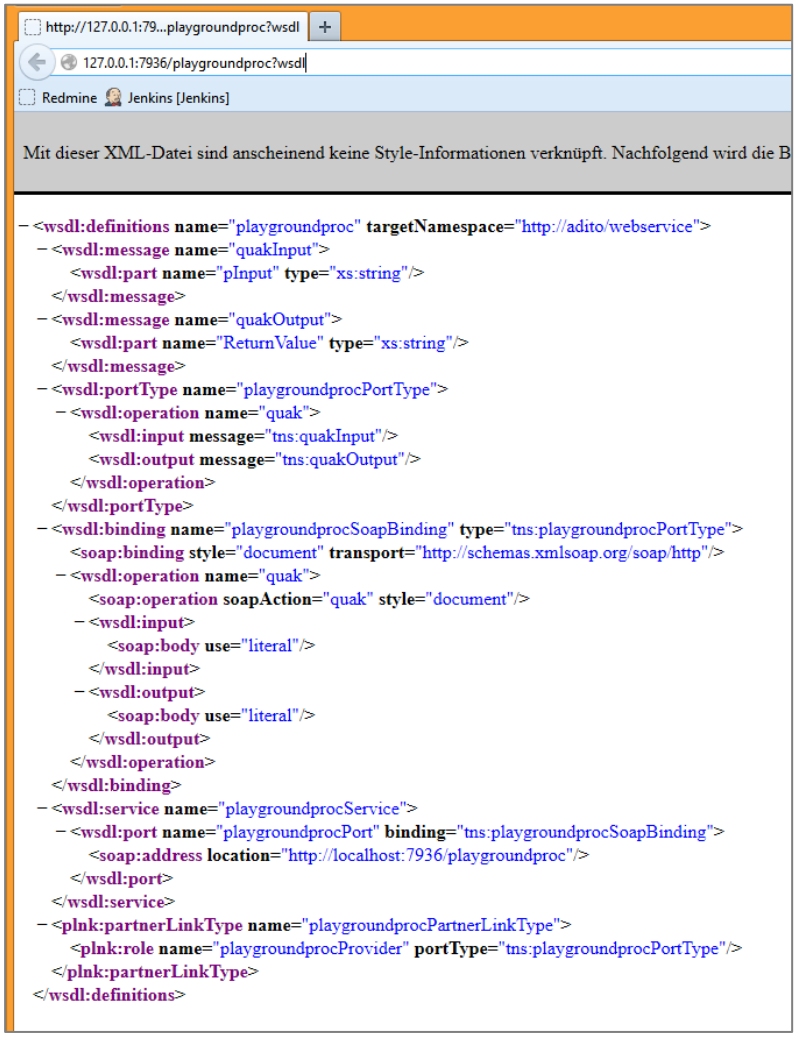
- `publishAsWebservice`: Aktiviert den JDito-Webservice.
- `style`: Verwendet für das SOAP-Binding `rpc`-style, falls aktiviert. Ansonsten wird `document` verwendet.

Der Aufruf erfolgt wieder über URL, z.B.

```
http://127.0.0.1:7936/playgroundproc?wsdl
```

wobei `playground` hier der Name des Prozesses ist.

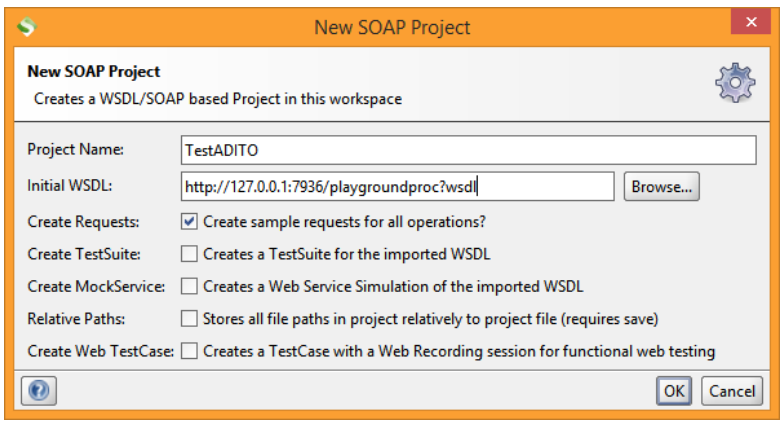
Das Ergebnis sieht wie folgt aus:



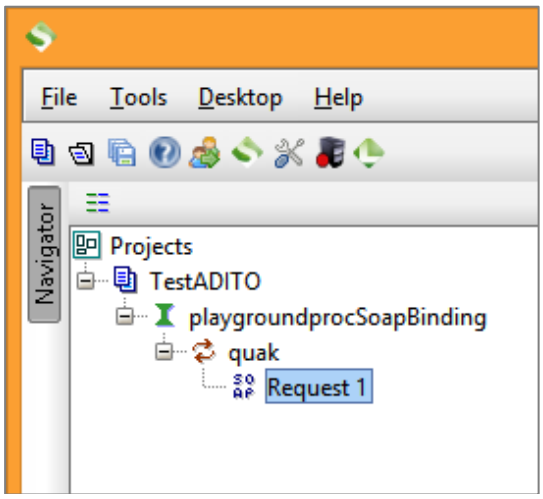
2.3.1.2. Anlegen des Webservices in SoapUI

SoapUI ist ein Werkzeug zum Testen von Webservices über SOAP-Zugriff. Für die Tests wurde die Version 4.6 verwendet.

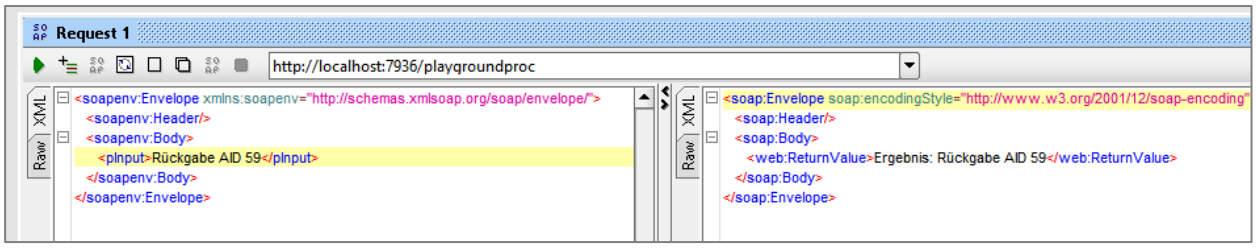
Über `File` → `New Soap Project` kann ein neues Projekt angelegt werden.



Nach Anlage des Projektes steht der Prozess zur Verfügung:

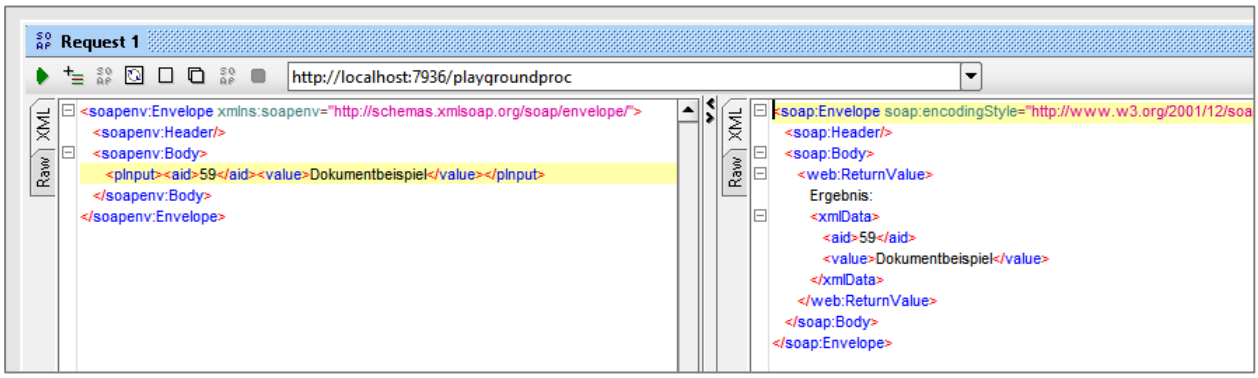


Per Doppelklick auf `Request1` kann ein Wert übergeben werden, im Folgenden „Rückgabe AID 59“:



Wie zu sehen ist, wird der String mit „Ergebnis“ entsprechend wieder ausgegeben.

Übergibt man statt eines Strings XML-Daten, so wird der Knoten `xmlData` vorangestellt.



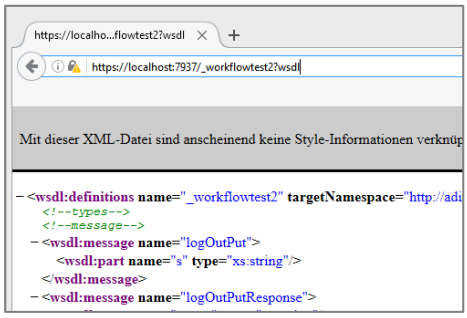
2.3.2. Anbinden von https-Webservices

JDito-Webservices können auch als https veröffentlicht werden. Voraussetzung hierfür ist, dass die JRE des ADITO-Servers im Verzeichnis `lib/security` mit einem gültigen Zertifikat ausgestattet wurde (Details hierzu: https://docs.oracle.com/cd/E19798-01/821-1841/gjrgy/index.html, Abrufdatum 21.06.2016).

Beim Serverstart muss noch ein Parameter mit angegeben werden, damit der Server beim Start auf den Keystore zugreifen kann:

```
-Djavax.net.ssl.keyStorePassword=changeit
```

Wurden diese Schritte berücksichtigt, kann der Server auch über eine https-Verbindung angesprochen werden.

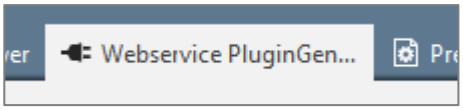


2.4. Ausgehende Webservices

Damit ADITO4 einen Webservice ansprechen kann, wird immer ein Plugin verwendet. Dieses Plugin kann entweder automatisch mit Hilfe des ADITO4 Managers verwendet werden, oder manuell, wenn es sich um spezielle Webservices handelt.

2.4.1. Plugin-Generierung mit Hilfe des ADITO4 Managers

Wenn der Webservice, den Sie ansprechen wollen, über eine WSDL-Datei ansprechbar ist und als Eingabeparameter nur einfache Datentypen benötigt (Boolean, Integer, Double, String, nicht aber Arrays und Objekte), können Sie ein vorgefertigtes ADITO-Plugin über den Manager erstellen lassen.

Den Webservice-Assistenten des ADITO4 Managers rufen Sie auf, indem Sie den Manager öffnen und in der Symbolleiste auf die Schaltfläche 

klicken.

Hier tragen Sie die Informationen ein, mit denen das Webservice-Plugin erstellt wird:

- **Dateipfad / URL**: Gibt eine gültige Webservice-Datei an (im Format WSDL) entweder von der Festplatte oder als URL.
- **Ausgabepfad**: Der Pfad, in dem das Plugin gespeichert wird. Geben Sie beim Dateinamen die Endung „jar“ an!
- **Packagename**: Der Name des Pakets, den das Plugin später besitzen soll (bspw. `de.adito.meinplugin.Webservice`).

Klicken Sie nun auf `Generate`, wird das Plugin erstellt und der für JDito nötige Aufrufcode im Ausgabefenster ausgegeben. Auch Fehler, die bei falschen Einstellungen auftreten, werden hier direkt ausgegeben!

In dieser Ausgabe erhalten Sie JDito-Funktionen, die Ihnen den Code vorbereiten, um gleich in ADITO damit loszulegen.

2.4.2. Beispiel: Zugriff auf Wetterdaten mit ADITO4 über ein generiertes Plugin

Auf Basis des Wetter Web Services von `webservicex.net` können Sie von ADITO aus auf Wetterdaten zugreifen.

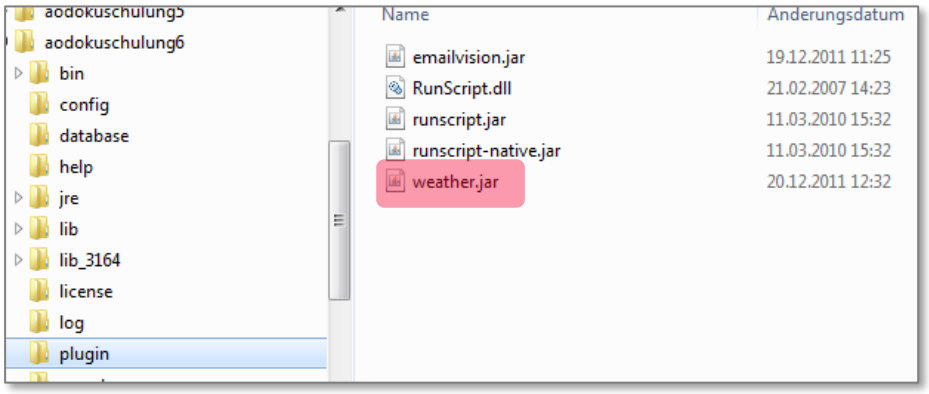
Der Wetter Web Service liegt unter
`http://www.webservicex.net/globalweather.asmx?WSDL`

Dieser Webservice ist so aufgebaut, dass der ADITO4 Manager ein Plugin generieren kann. Dieses steht nach erfolgreicher Erstellung im angegebenen Ausgabepfad zur Verfügung.

Der ADITO4 Webservice Plugin Generator kann nur Server Plugins anlegen!

2.4.2.1. Aufruf des Plugins unter ADITO4

Um auf das erstellte Plugin zugreifen zu können, muss es im Plugin-Verzeichnis des ADITO4 Verzeichnisses vorliegen:



Nun können Sie entweder das Plugin in den Klassenpfad des Servers mit aufnehmen, oder Sie geben beim Plugin-Aufruf den Speicherort der `.jar`-Datei mit an. Wir verwenden das Beispiel der Integration in den Klassenpfad.

Folgende Änderungen wurden in der server.bat vorgenommen. Starten Sie den ADITO4 Server über einen anderen Weg, müssen Sie den Klassenpfad dort entsprechend anpassen. Konsultieren Sie hierfür das Betriebshandbuch.

```
..\..\jre\jre6\bin\java -Djava.awt.headless=true -Xms64m -Xmx1024m -
cp
..\..\system\serverlicense.jar;server.jar;mail.jar;concurrent.jar;cr
ystalclear.jar;cclib.jar;cc-
viewer.jar;jfreechartcc.jar;jcommon.jar;js.jar;jdom.jar;ical4j.jar;c
ommons-codec.jar;commons-httpclient.jar;commons-
logging.jar;exchange.jar;log4j.jar;xbean.jar;commons-
io.jar;notes/notes.jar;notes/ncso.jar;domino.jar;joda-
time.jar;images.jar;flamingo.jar;exchangews.jar;exchangewsstubs.jar;
..\..\plugin\emailvision.jar;..\..\plugin\weather.jar
de.adito.aditoweb.server.Server ../../config/serverconfig.xml
```

Nun kann über JDito auf dieses Plugin zugegriffen werden.

2.4.2.2. Zugriff auf das eingebundene Plugin mit JDito

Über JDito kann ein Serverprozess angelegt werden, welcher auf das Plugin zugreift. Auch wenn der Plugin Generator nur ein Server-Plugin erstellt, so können Sie diesen Prozess auch vom Client heraus starten.

Der Webservice ist einfach gestrickt. Man übergibt ihm ein Land und eine Stadt, und bekommt dann die Wetterdaten per XML.

Die Verarbeitung von XML-Daten in JDito bzw. JavaScript können Sie dem AID071-DE - XML mit JDito und JavaScript, entnehmen.

Folgender Prozess wurde nun erstellt als Mischung aus selbst erstelltem und vom Manager automatisch generiertem Code.

```
import("lib_util");
import("system.question");
import("system.plugin");



var location = chooseCity();
question.showMessage(getWeather(location[0], location[1]));


/**
 * AUTO-GENERATED-CODE!
 *
 * Dokumentation der Methode
 */
function getWeather(CityName, CountryName)
{
 var param = new Array();
 param[0] = CityName;
 param[1] = CountryName;
 var result = plugin.run(null, "de.adito.weather.GetWeather" ,
param); // Sollte nur als Clientplugin laufen
 //var result =
swing.doClientIntermediate(swing.CLIENTCMD_RUNPLUGIN, new
Array(null, "de.adito.weather.GetWeather" , param))
 return result[0];
}
```

```
/**
 * AUTO-GENERATED-CODE!
 *
 * Dokumentation der Methode
 */
function getCitiesByCountry(CountryName)
{
 var param = new Array();
 param[0] = CountryName;
 var result = plugin.run (null,
"de.adito.weather.GetCitiesByCountry" , param); // Sollte nur als
Clientplugin laufen
 //var result =
swing.doClientIntermediate(swing.CLIENTCMD_RUNPLUGIN, new
Array(null, "de.adito.weather.GetCitiesByCountry" , param))
 return result[0];
}


/**
 * Holt sich alle Städte
 */
function getCityList(pCountryName)
{
 if(pCountryName != "")
 {
  var NewDataSet = new XML(cities =
getCitiesByCountry(pCountryName));
  var citylist = [];


  for each (var i in NewDataSet.Table.City)
  {
   citylist.push(i);
  }


  return citylist;
 }
 else
  return new Array();
}
```

```
/**
 * Wählt Stadt aus
 */
function chooseCity()
{
 var countries = ["Germany", "Austria", "Switzerland", "United
Kingdom"];
 var country = question.askQuestion("Bitte Land wählen",
question.QUESTION_COMBOBOX, "|" + countries.join("|"));
 if(country != null)
 {
  var city = question.askQuestion("Bitte Stadt wählen",
question.QUESTION_COMBOBOX, "|" + getCityList(country).join("|"));
  if(city != null)
  {
   return [city, country];
  }
  else return [];
 }
 else return [];
}
```

Führt man nun den Code aus, bekommt man eine Liste aller Städte in der vorausgewählten Stadt.

2.5. Externe Webservices ansprechen mit eigenem Plugin

Die Möglichkeiten des Plugin-Generators sind beschränkt. Mit diesem Generator können Sie einfache Plugins erstellen. Wenn dies nicht möglich ist, dann muss selbst ein Plugin generiert werden.

Um die benötigten Java-Klassen zu bekommen, benötigen Sie eine Java-Entwicklungsumgebung, das JDK (Java Development Kit). Über das Tool `wsimport` können Sie die entsprechenden Schnittstellen erstellen.

3. REST-Webservices

3.1. Eingehende Webservices

Sollen ADITO-Prozesse als RESTful Webservices angeboten werden, gilt die folgende Vorgehensweise.

3.1.1. Vorkonfiguration

Die Konfiguration von Prozessen als RESTful Webservice Provider funktioniert wie die Konfiguration von Webservices Server für SOAP (siehe hier).

Bei der Verwendung von JDito-Webservices über SOAP muss in den `Preferences` bei der Konfiguration der Webservices unbedingt ein Benutzerkonto `jditoWebserviceUser` angegeben werden.

3.1.2. Prozess-Konfiguration

Beim Prozess können die folgenden Eigenschaften konfiguriert werden:

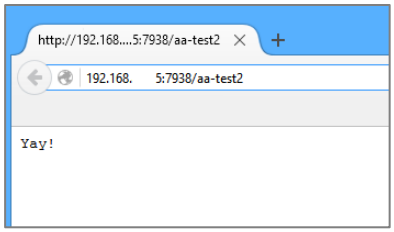
- `publishAsWebservice`: Aktiviert
- `typeForWebservice`: HTTP oder HTTPS (nur bei JDito-Webservice).
- `style`: Hier REST.
- `restAcceptedMimeType`: Mime-Typen, die vom Rest-Webservice akzeptiert werden. Hier text/plain.
- `restDeliveredMimeType`: Mime-Typ, den der Rest-Webservice zurückliefert. Hier text/plain.
- `basicAuthentication`: „Ja“ für Anmeldung mit Benutzername und Passwort.
- `restrictedRoles`: Rollen, die sich nicht am Webservice anmelden können.

Der Prozess muss die Funktion get besitzen, deren Returnwert wird vom Prozess ausgegeben.

3.1.3. Beispielprozess für GET

```
function get(params)
{
    var retcode = "Yay!";
    return retcode;
}
```

Bewirkt bei Aufruf des Webservices über einen Browser folgendes Ergebnis:



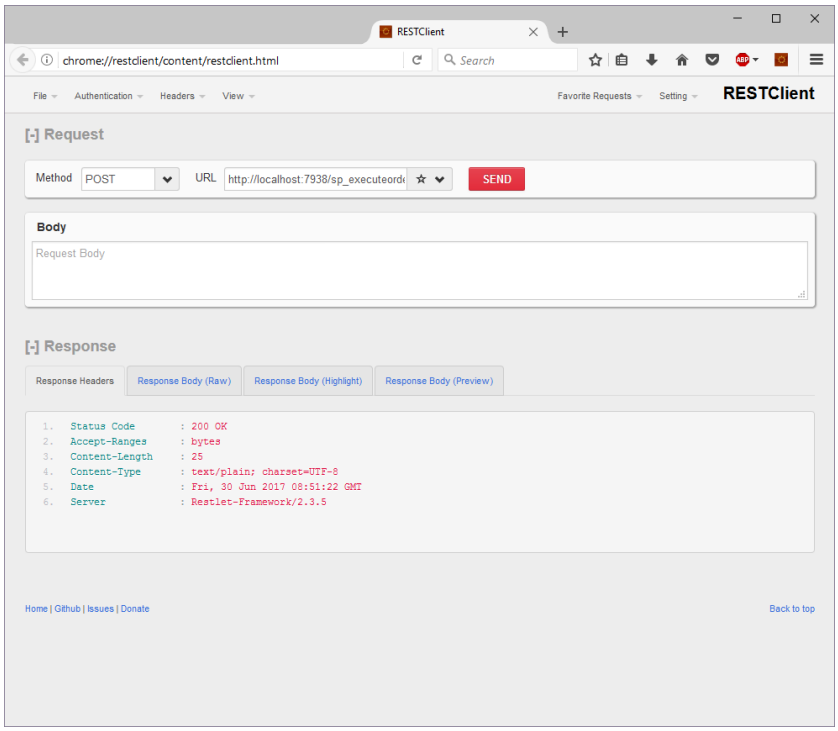
Soll ADITO einen RESTful Webservice anbieten, muss darauf geachtet werden, dass in der URL kein nachfolgender Slash enthalten ist:
`GET www.adito.de/xrm` holt das spezifische Element „xrm“,
`GET www.adito.de/xrm/` holt alle Elemente aus der Gruppe „xrm“.

Siehe hierzu auch:
https://softwareengineering.stackexchange.com/questions/186959/trailing-slash-in-restful-api#18700

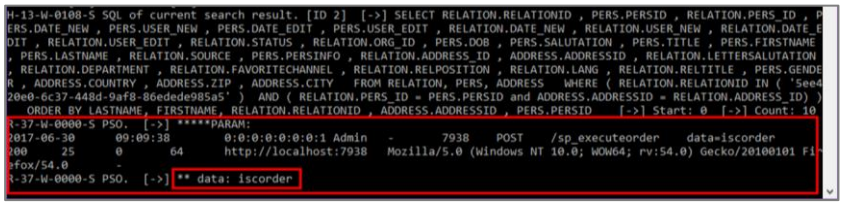
3.1.4. Beispielprozess für POST

```
import("system.logging");
function post(params)
{
        logging.log("*****PARAM: ");
        for ( i in params )
        {
            logging.log("** " + i + ": " + params[i]);
        }
        var retcode = "*** POST HAT FUNKTIONIERT";
        return retcode;
}
```

Anzeige im Firefox RESTClient Add On:



Ausgabe im Serverlog:



3.2. Ausgehende Webservices

Beim Aufruf von RESTful Webservices verwendet ADITO die Jersey-API von Java.

Die Übergabe von Daten kann über **inputValues** (Schlüssel-Wert-Paare) und über den **MessageBody** erfolgen.

3.2.1. Aufruf über inputValues

Folgendes Codebeispiel übergibt Werte an einen RESTful Webservice mit Hilfe von Inputvariablen.

```
// Übergabe des XML an den Webservice
var url = "http://172.23.42.137:8000/webservicecall";
var actionType = "POST";

var txtmsg = [];
txtmsg[0] = "Das ist ein Test";
```

```
// Hier geht's mit den input Variables weiter


var inputValues = new Object(); // "QueryParameter"
inputValues["AditoData"] = txtmsg;


var datatypeAccept = "text/xml";
var datatypeSend = "text/xml";
var datatypeJdito = a.DATA_TEXT;


//var user = "";
//var password = "";


logging.log("**** Offer: VOR Aufruf Webservice!");


var wsret = net.callRestWebserviceNoAuth(url, actionType,
inputValues, datatypeAccept, datatypeSend, datatypeJdito);


question.showMessage("Daten übergeben.")


logging.log("**** Offer: NACH Aufruf Webservice! Antwort: " +
wsret);
```

3.2.2. Aufruf mit messageBody

Sollen Werte über einen `messageBody` übergeben werden, erfolgt dieser Aufruf über die Variable `requestEntity`.

Einfaches Beispiel:

```
var requestentity =
"<resource><ID>4242</ID><NAME>Franz</NAME><PRICE>23</PRICE></resourc
e>";


var s=net.callRestWebserviceNoAuth("http://www.thomas-
bayer.com/sqlrest/PRODUCT/", "POST", null, requestentity,


"TEXT/PLAIN", "TEXT/PLAIN", a.DATA_TEXT, a.DATA_TEXT)
```

Komplexeres Beispiel:

```
var webserviceURL = 'https://webservice.tourenplaner.biz/startjob';


var actionType = "POST";
```

```
  var reqEntity = { "waypoints" : [
          {"address: " : {"locality": "Geisenhausen", "postcode":
"84144", "street":"Gutenbergstrasse 1"}},
          {"address: " : {"locality": "Landshut", "postcode":
"84028", "street":"Altstadt 1"}  }
  ]}



var reqEntityStr =  JSON.stringify(reqEntity);
question.showMessage(reqEntityStr)

var user = "special@user.de";
var password = "hurgaburga";

logging.log("**** VOR Aufruf Webservice!");

try
{
var wsret = net.callRestWebserviceBasicAuth(webserviceURL,
actionType, null, reqEntityStr, "application/json",
"application/json", a.DATA_TEXT, a.DATA_TEXT, user, password);
} catch (ex)
{
     logging.show(ex);
     //logging.log(ex.javaException.getMessage());
}

question.showMessage("Daten an Webservice übergeben.")

logging.log("**** NACH Aufruf Webservice! Antwort: " + wsret);
```