2009

Developer Guide

IMAP Customizing

Dr. Bernd Grimm

Version: 1.0

Inhalt

1 Einleitung

2 Kundenprojekt im Eclipse anlegen

Im Eclipse ist ein dynamisches Web Projekt anzulegen:

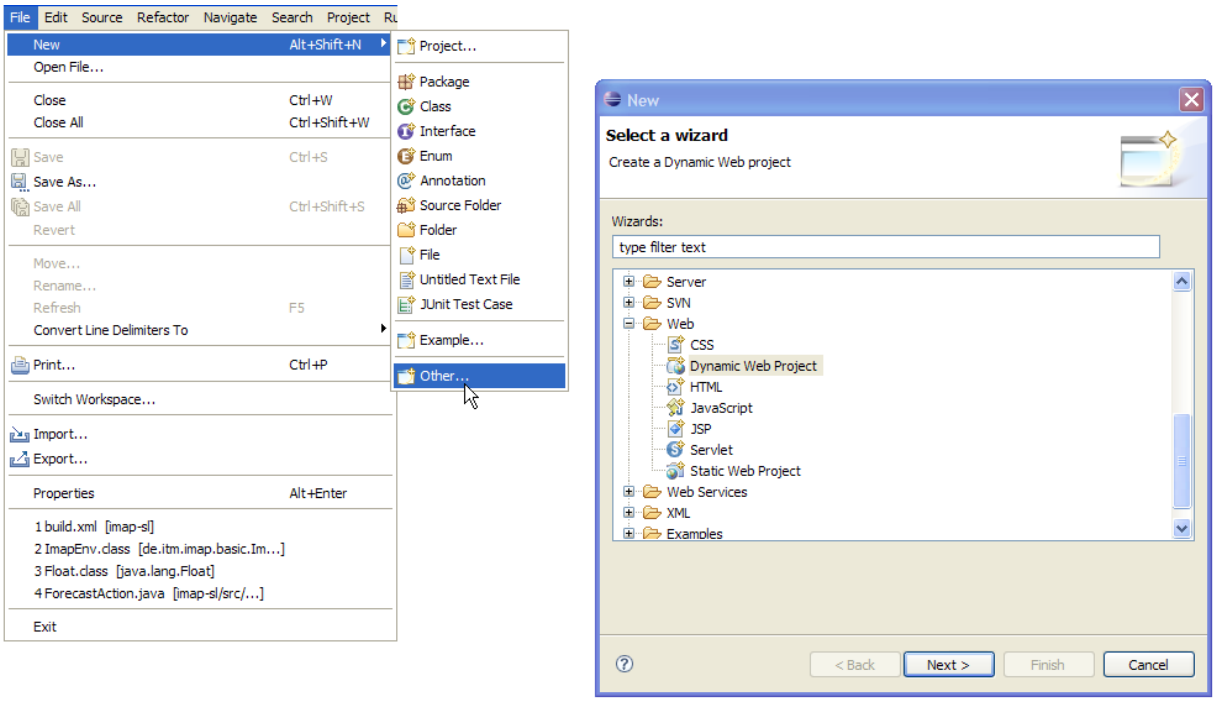


ABBILDUNG 1: DYNAMISCHES WEB PROJEKT ANLEGEN

Im Ordner `WEB-INF` muss die Datei `web.xml` gelöscht werden. Diese Datei wird aus der Basis-IMAP Distribution übernommen.

Als Output-Folder wird der Pfad `WebContent/WEB-INF/classes` eingestellt. Der build-Ordner kann gelöscht werden.

Anlegen des `dist`-Ordners und die Datei `imapBase.war` in diesen Ordner kopieren; dieser Schritt wird auch durch den Ant-Task `get actual ImapBase.war` erreicht.

Anpassung der properties im Ant Build Script:

- imapCustomer.jar
- imapCustomer.war
- local.server

```
<property name="tmp" value="dist/tmp" />
<property name="dist" value="dist" />
<property name="web" value="WebContent" />
<property name="actual.imap.dist" value="${ENV.IMAP.HOME}/dist" />
<property name="imapCustomer.jar"
          value="${actual.imap.dist}/imap-cust.jar" />
<property name="imapCustomer.war"
          value="${actual.imap.dist}/imap-cust.war" />
<property name="imapBase.war" value="${dist}/imapBase.war" />
<property name="local.server"
          value="${ENV.IMAP.SERVER}/webapps/imap-cust" />
```

QUELLCODE 1: PROPERTIES IM ANT BUILD SCRIPT

Folgende Ant-Tasks ausführen:

- get actual ImapBase.war
- get actual ImapEngine.jar
- get actual external libs
- deploy ImapBase.war to local Server

Lokales Home Verzeichnis für die Web Anwendung anlegen:

`${imap.home}/${web.context.path}`

In diesem Verzeichnis ist die Datei `imap.system.properties` anzulegen.

```
imap.db.user = imap
imap.db.password = imap
imap.db.host = localhost:3306
imap.db.schema = imap4Customer
```

QUELLCODE 2: AUSZUG AUS DER DATEI IMAP.SYSTEM.PROPERTIES

In der verwendeten Datenbank ist das Schema „imap4Customer" mit allen Rechten für den User „imap" anzulegen.

2.1 Tomcat Einstellungen

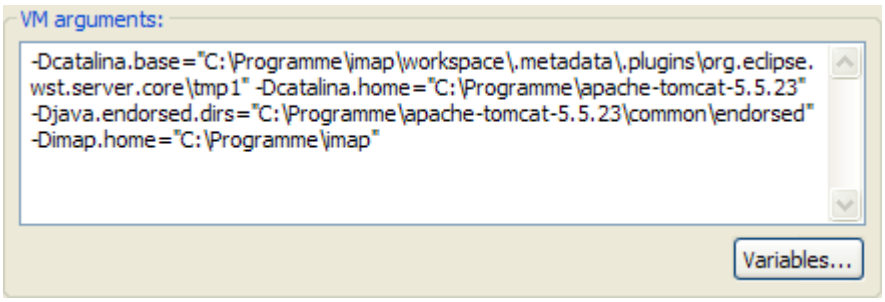


ABBILDUNG 2: VM ARGUMENTE FÜR DEN TOMCAT

2.2 Umgebungsvariablen

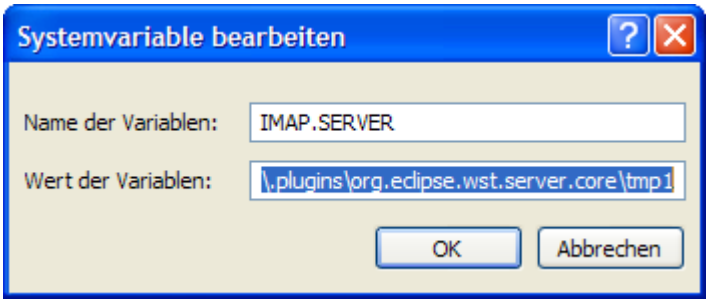


ABBILDUNG 3: UMGEBUNGSVARIABLE IMAP.SERVER = C:/PROGRAMME/IMAP/WORKSPACE/.METADATA/.PLUGINS/ORG.ECLIPSE.WST.SERVER.CORE/TMP1

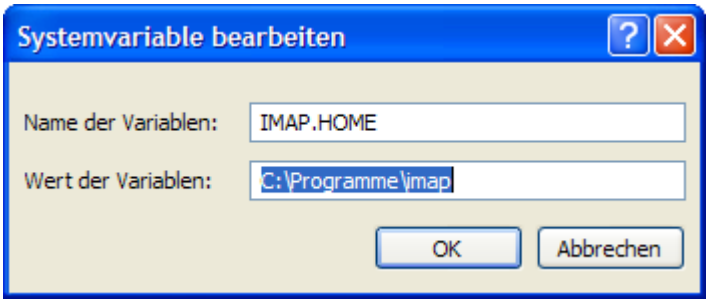


ABBILDUNG 4: UMGEBUNGSVARIABLE IMAP.HOME = C:/PROGRAMME/IMAP

3 Service Implementierung

3.1 SAP Services (JCO 2.1.8)

3.1.1 Voraussetzung

Der SAP-JCO muss auf dem Server System installiert sein.

Installationsanleitung:

http://help.sap.com/saphelp_nw04/helpdata/en/47/80f671ee6e4b41b63c0fe46bd6e4f8/content.htm

3.1.2 Connector API

Folgende Klassen müssen vorhanden sein:

de.itm.imap.connector.sap.SapConnector

de.itm.imap.services.sap.SapBasicService

de.itm.imap.services.sap.SapConnectorService

3.1.3 Verbindung zum SAP herstellen

Definition eines Web Services in der service.xml:

```
<service name="MySapSystem" autodeploy="true">
    <class>de.itm.imap.services.sap.SapConnectorService</class>
    <method name="connect"></method>
    <method name="reconnect"></method>
    <method name="getSystemInfo"></method>
    <property name="Client" description="Mandant" parameter="STRING"
              value="800"/>
    <property name="User" description="Benutzer" parameter="STRING"
              value="sapUser"/>
    <property name="Password" description="Kennwort" parameter="PASSWORD"
              value="sapPassword"/>
    <property name="Host" description="Anwendungserver/Routing"
              parameter="STRING" value="/H/FRODO"/>
    <property name="SystemNo" description="Systemnummer" parameter="STRING"
              value="40"/>
    <property name="Language" description="Sprache" parameter="STRING"
              value="DE"/>
</service>
```

3.1.4 SAP Funktionsbaustein als Web Service bereitstellen

Beispiel: BAPI_MATERIAL_GET_ALL

Implementierung der Service Klasse (Beispiel MaterialService):

```
public class MaterialService extends SapBasicService
```

```
SapConnector sap=null;
try
{
    sap = new SapConnector(getProperty("SapConnectorService"));
    sap.open();
    IFunctionTemplate ft =
        sap.getRepository().getFunctionTemplate(getProperty("BAPI"));

    if (ft!=null)
    {
        JCO.Function fct = ft.getFunction();

        // fill import parameter list

        // fill import tables

        // execute function
        sap.execute(fct);

        // get export parameter list

        // get export tables
    }
}
catch (Exception e)
{
    throw e;
}
finally
{
    if (sap!=null)
        sap.close();
}
```

```
// fill import parameter list
fct.getImportParameterList().setValue(materialNo, "MATERIAL");
```

```
// get export table
JCO.Table table =
        fct.getTableParameterList().getTable("MATERIALDESCRIPTION");
    if (table.getNumRows() > 0)
    {
        for (int i=0; i<table.getNumRows(); i++)
                                {
        table.setRow(i);
        String lang = table.getString("LANGU");
        if (lang.equals(language))
        {
            shortText = table.getString("MATL_DESC");
        }
    }
}
```

Service in der service.xml deklarieren:

```
<service name="MaterialService" use="literal" autodeploy="true">
    <class>de.br.imap.services.sap.material.MaterialService</class>
    <method name="getShortText"></method>
    <method name="transactionStart"></method>
    <method name="transactionCommit"></method>
    <method name="transactionRollback"></method>
    <property name="SapConnectorService" description="SAP System"
        parameter="STRING" value="MySapSystem"/>
    <property name="BAPI" description="Name des RFC Bausteins"
        parameter="STRING" value="BAPI_MATERIAL_GET_ALL"/>
</service>
```

3.2 SAP Services (JCO 3.0.2)

3.2.1 Voraussetzung

Der SAP-JCO muss auf dem Server System installiert sein.

Installationsanleitung:

http://help.sap.com/saphelp_nwpi711/helpdata/de/48/707c54872c1b5ae10000000a42189c/content.htm

3.2.2 Connector API

Folgende Klassen müssen vorhanden sein:

de.itm.imap.connector.sap.SapConnection3
de.itm.imap.connector.sap.MyDestinationDataProvider
de.itm.imap.services.sap.SapBasicService3
de.itm.imap.services.sap.SapConnectorService3

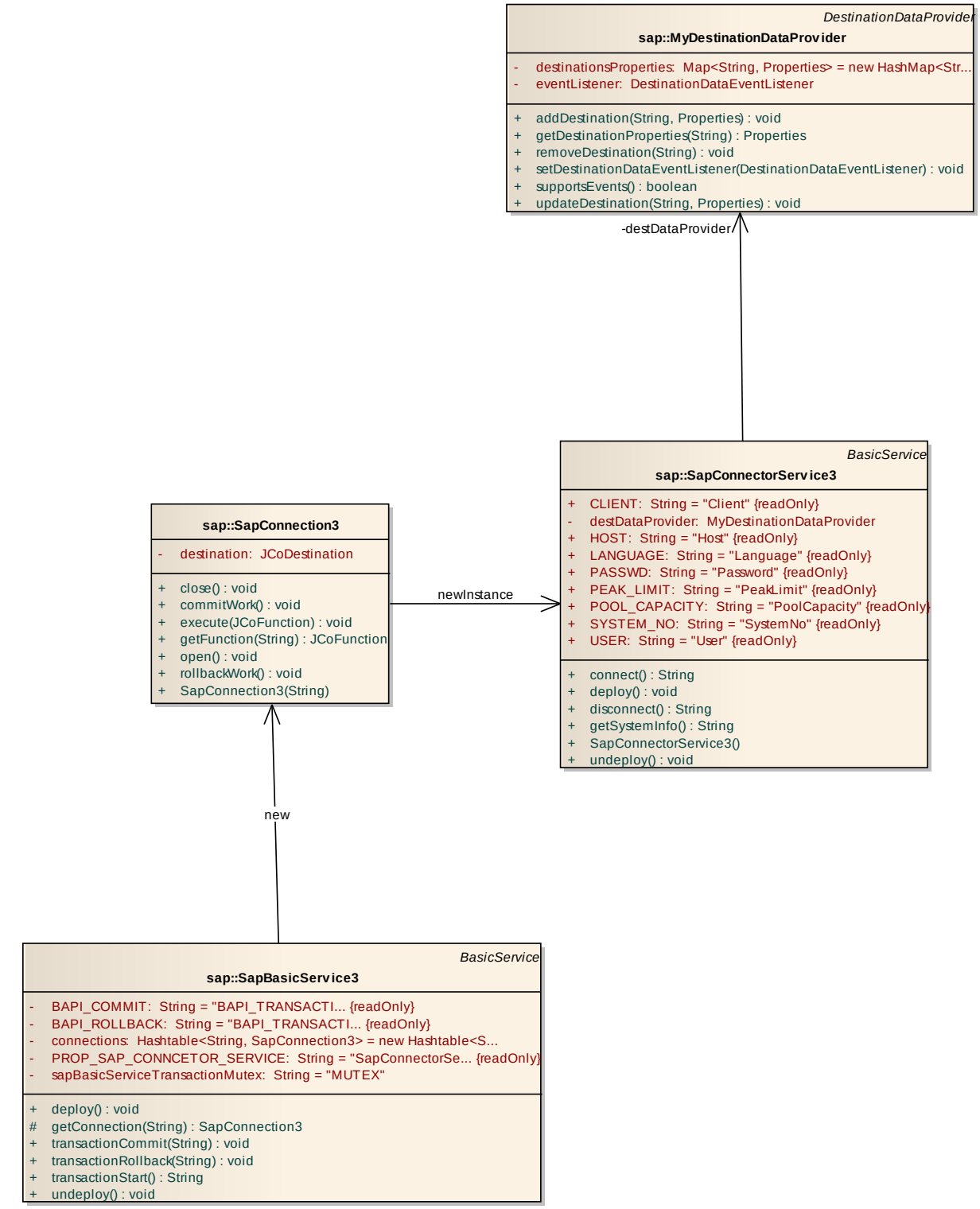


ABBILDUNG 5: KLASSENDIAGRAMM ZUM SAPCONNECTOR

3.2.3 Verbindung zum SAP herstellen

Definition eines Web Services in der service.xml:

```
<service name="IMAP.Standard.SapConncetor3" autodeploy="true">
    <alias>SapConnector3</alias>
    <class>de.itm.imap.services.sap.SapConnectorService3</class>
    <method name="connect"></method>
    <method name="disconnect"></method>
    <method name="getSystemInfo"></method>
    <property name="Client" description="Mandant" parameter="STRING"
              value="800"/>
    <property name="User" description="Benutzer" parameter="STRING"
              value="sapUser"/>
    <property name="Password" description="Kennwort" parameter="PASSWORD"
              value="sapPassword"/>
    <property name="Host" description="Anwendungserver/Routing"
              parameter="STRING" value="/H/FRODO"/>
    <property name="SystemNo" description="Systemnummer" parameter="STRING"
              value="01"/>
    <property name="Language" description="Sprache" parameter="STRING"
              value="DE"/>
    <property name="PoolCapacity"
              description="Maximum number of idle connections kept
                           open to the SAP system"
              parameter="STRING" value="3"/>
    <property name="PeakLimit"
              description="Maximum number of active connections that can be
                           created for the SAP system simultaneously"
              parameter="STRING"
              value="10"/>
</service>
```

3.2.4 SAP Funktionsbaustein als Web Service bereitstellen

Beispiel: BAPI_MATERIAL_GET_ALL

Implementierung der Service Klasse (Beispiel MaterialService):

```
public class MaterialService extends SapBasicService3
```

```
SapConnection3 conn = null;
try
{
    conn = new SapConnection3(getProperty("SapConnectorService"));
    conn.open();
    JCoFunction function = conn.getFunction(getProperty("BAPI"));

    if (function!=null)
    {
        // fill import parameter list

        // fill import tables

        // execute function
        conn.execute(function);

        // get export parameter list

        // get export tables
    }
}
catch (Exception e)
{
    throw e;
}
finally
{
    if (conn!=null)
        conn.close();
}
```

```
// fill import parameter list
function.getImportParameterList().setValue("MATERIAL", materialNo);
```

```
// get export table
JCoTable table =
        function.getTableParameterList().getTable("MATERIALDESCRIPTION");
    if (table.getNumRows() > 0)
    {
        for (int i=0; i<table.getNumRows(); i++)
                                {
        table.setRow(i);
        String lang = table.getString("LANGU");
        if (lang.equals(language))
        {
            shortText = table.getString("MATL_DESC");
        }
    }
}
```

Service in der service.xml deklarieren:

```
<service name="IMAP.Standard.SapMaterial">
    <alias>SapMaterial</alias>
    <class>de.itm.imap.services.sap.examples.SapMaterialService</class>
    <method name="getShortText"></method>
    <method name="transactionStart"></method>
    <method name="transactionCommit"></method>
    <method name="transactionRollback"></method>
    <property name="SapConnectorService" description="SAP System"
        parameter="STRING" value="IMAP.Standard.SapConncetor3"/>
    <property name="BAPI" description="Name des RFC Bausteins"
        parameter="STRING" value="BAPI_MATERIAL_GET_ALL"/>
</service>
```

3.3 SAP RFC Server Service (JCO 3.0.2)

Der RFC Server in IMAP dient zur Verarbeitung von SAP RFC Anfragen. In diesem Abschnitt werden die einzelnen Einstellungsmöglichkeiten und der Aufbau der Verarbeitungs-Klasse (Handler) erläutert.

3.3.1 Voraussetzung

Folgende Klassen müssen vorhanden sein (siehe auch *Kapitel 3.2*):

de.itm.imap.connector.sap.SapRfcServerListener3
de.itm.imap.connector.sap.MyServerDataProvider
de.itm.imap.connector.sap.SapRfcServerFunctionHandler3
de.itm.imap.services.sap.SapBasicRfcServerService3

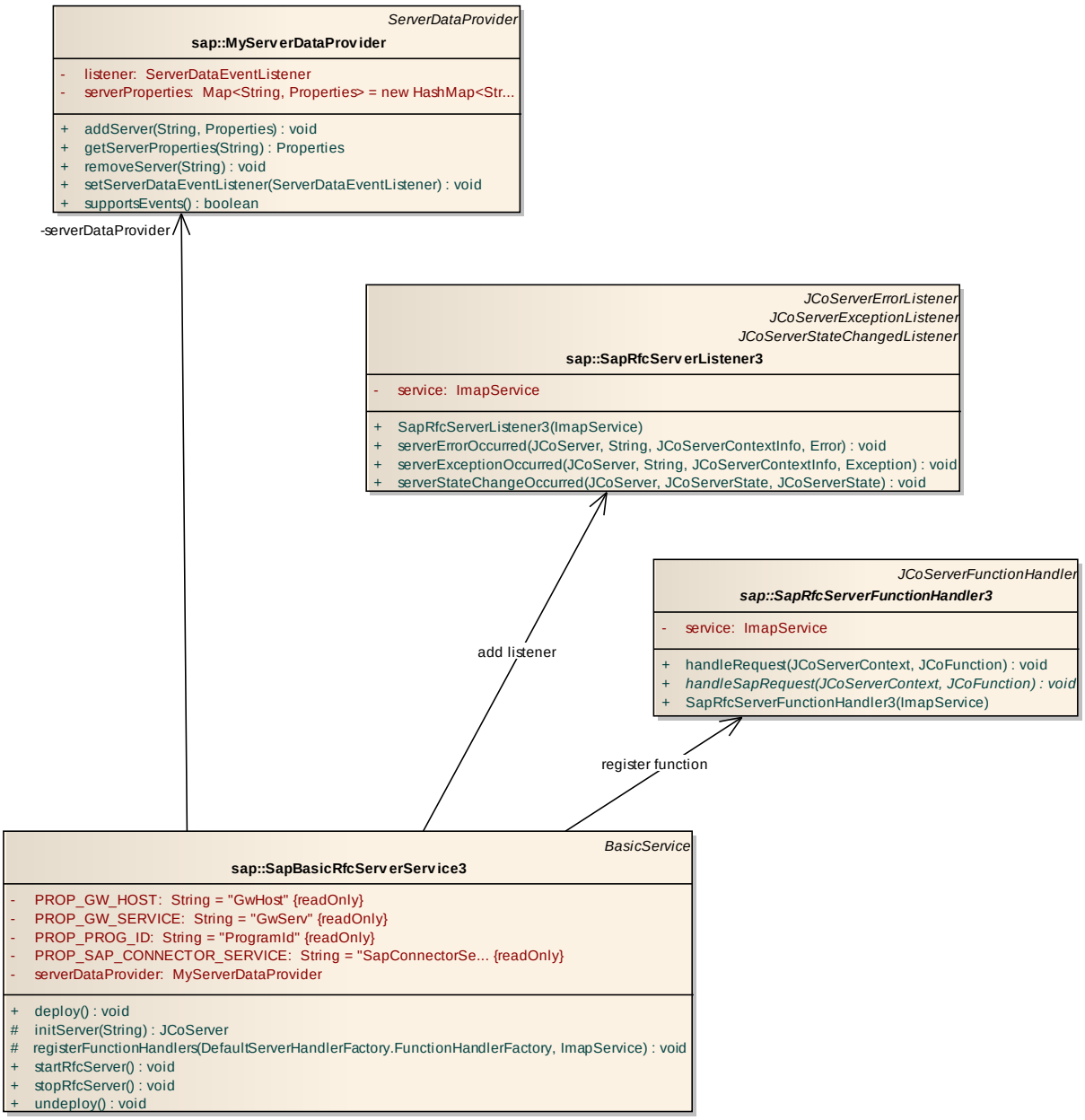


ABBILDUNG 6: KLASSENDIAGRAMM RFC-SERVER

3.3.2 Service Einstellungen

Parameter	Version
GwHost	SAP Gateway Host
GwServ	SAP Gateway Port
ProgramId	Eindeutiger Name, unter der sich der RFC-Server beim SAP Gateway registriert

SapConnectorService Web Service Name des Connector Services (siehe auch *Kapitel 3.2.3*)

3.3.3 RFC Server Service Klasse

Um einen eigenen Server Service bereitzustellen, muss ein Web Service erstellt werden, der vom „SapBasicRfcServerService3" abgeleitet werden muss.

```
public class MySapBasicRfcServerService3 extends SapBasicRfcServerService3
{
}
```

Zusätzlich müssen die gewünschten Handler Klassen registriert werden. Folgender Code Abschnitt zeigt die Registrierung eines Handlers, der aufgerufen wird, wenn ein Aufruf aus dem SAP mit dem Key der Key „STFC_CONNECTION" gestartet wird.

```
protected void registerFunctionHandlers
    (DefaultServerHandlerFactory.FunctionHandlerFactory factory,
     ImapService service)
{
    factory.registerHandler("STFC_CONNECTION",
        new StfcConnectionFunctionHandler(service));

    factory.registerGenericHandler(new GenericFunctionHandler(service));
}
```

3.3.4 Handler Klasse

Um Anfragen des SAP Systems bearbeiten zu können muss eine Handler Klasse implementiert werden. Eine Handler Klasse besitzt folgenden Aufbau und muss von der Klasse „SapRfcServerFunctionHandler3" abgeleitet sein:

```
public class StfcConnectionFunctionHandler extends
    SapRfcServerFunctionHandler3
{
}
```

Constructor:

```
public StfcConnectionFunctionHandler(ImapService service)
{
    super(service);
}
```

Handler Methode:

```
public void handleSapRequest(JCoServerContext serverContext, JCoFunction
function) throws AbapException
{
    String reqTxt = function.getImportParameterList().getString("REQUTEXT");

    function.getExportParameterList().setValue("ECHOTEXT", reqTxt);

    function.getExportParameterList().setValue
        ("RESPTEXT", "it-motive IMAP Version 2.5");
}
```

3.3.5 RFC-Server Service konfigurieren

Der der Service wird in der `service.xml` konfiguriert.

```
<service name="IMAP.Standard.SapRfcServer3">
    <alias>SapRfcServer3</alias>
    <class>de.itm.imap.services.sap.MySapBasicRfcServerService3</class>
    <method name="startRfcServer"></method>
    <method name="stopRfcServer"></method>
    <property name="GwHost" description="Gateway Host"
              parameter="STRING" value="/H/FRODO"></property>
    <property name="GwServ" description="Gateway Service"
              parameter="STRING" value="3301"></property>
    <property name="ProgramId" description="Name of the Server Program"
              parameter="STRING" value="IMAP_RFC_SERVER"></property>
    <property name="SapConnectorService"
              description="Name of the SAP connector service"
              parameter="STRING" value="IMAP.Standard.SapConncetor3">
    </property>
</service>
```

3.3.6 RFC-Verbindung im SAP konfigurieren

Hierzu wird die Transaktion `SM59` benutzt. Es wird eine neue TCP/IP Verbindung angelegt.

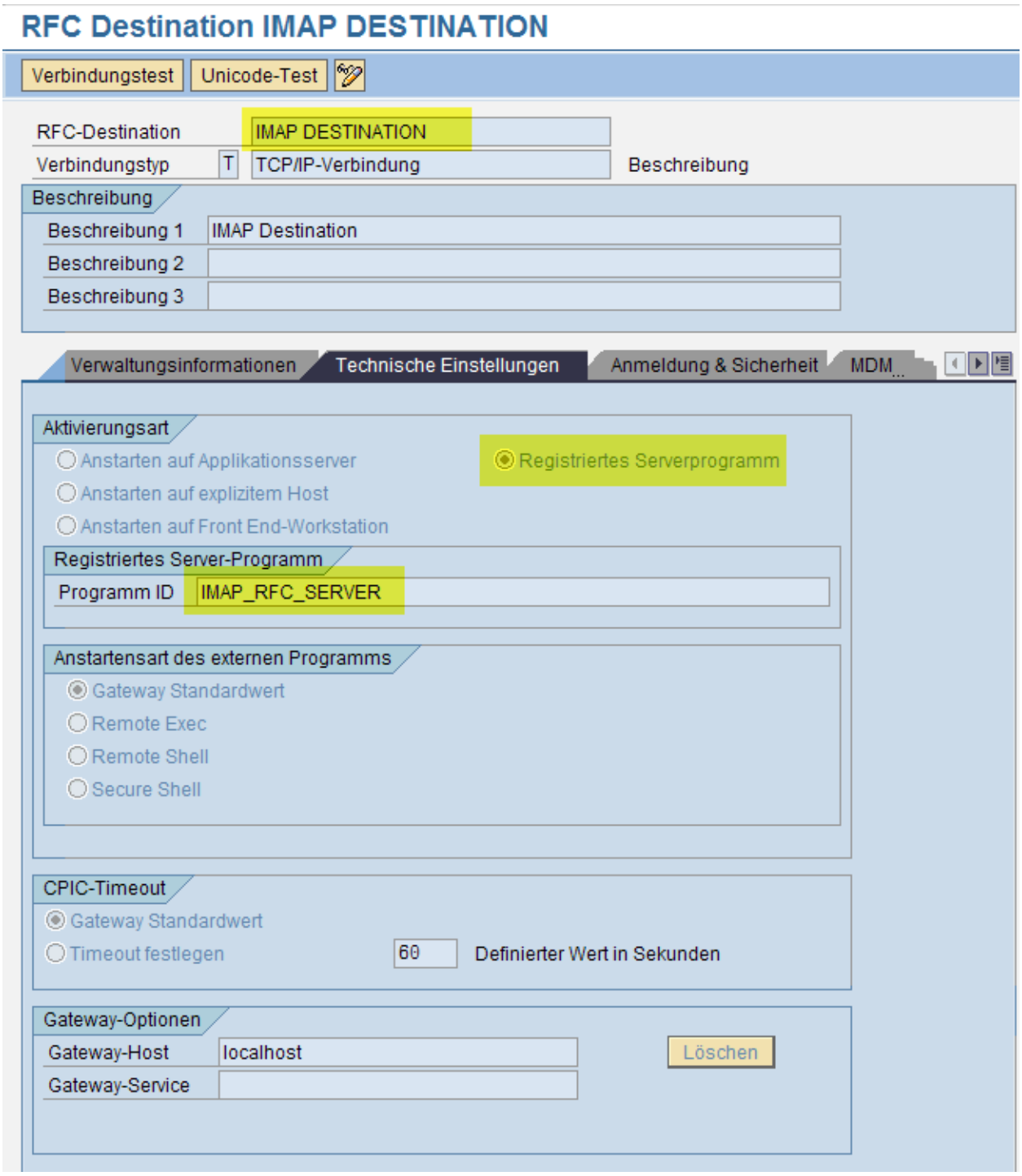


ABBILDUNG 7: RFC-VERBINDUNG MIT DER TRANSAKTION SM59 EINRICHTEN

3.3.7 ABAP zum Testen des RFC Servers

Funktionsbaustein Z_RFC_IMAP_SERVER_TEST aktiv

Eigenschaften | Import | Export | Changing | Tabellen | Ausnahmen | Quelltext

```
FUNCTION Z_RFC_IMAP_SERVER_TEST.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"----------------------------------------------------------------------

  data: REQUTEXT LIKE SY-LISEL,
        RESPTEXT LIKE SY-LISEL,
        ECHOTEXT LIKE SY-LISEL.

  DATA: RFCDEST like rfcdes-rfcdest VALUE 'NONE'.
  DATA: RFC_MESS(128).

  REQUTEXT = 'Hello World'.
  RFCDEST ='IMAP DESTINATION'.

  DO 1 TIMES.

    CALL FUNCTION 'STFC_CONNECTION'
      DESTINATION RFCDEST
      EXPORTING
        REQUTEXT              = REQUTEXT
      IMPORTING
        RESPTEXT              = RESPTEXT
        ECHOTEXT              = ECHOTEXT
      EXCEPTIONS
        SYSTEM_FAILURE        = 1  MESSAGE RFC_MESS
        COMMUNICATION_FAILURE = 2  MESSAGE RFC_MESS.


    IF SY-SUBRC NE 0.
      WRITE: / 'Call STFC_CONNECTION SY-SUBRC = ', SY-SUBRC.
      WRITE: / RFC_MESS.
    ELSE.
      WRITE: / REQUTEXT.
      WRITE: / RESPTEXT.
      WRITE: / ECHOTEXT.
    ENDIF.
  ENDDO.



ENDFUNCTION.
```

ABBILDUNG 8: ABAP ZUM TESTEN DES RFC-SERVERS

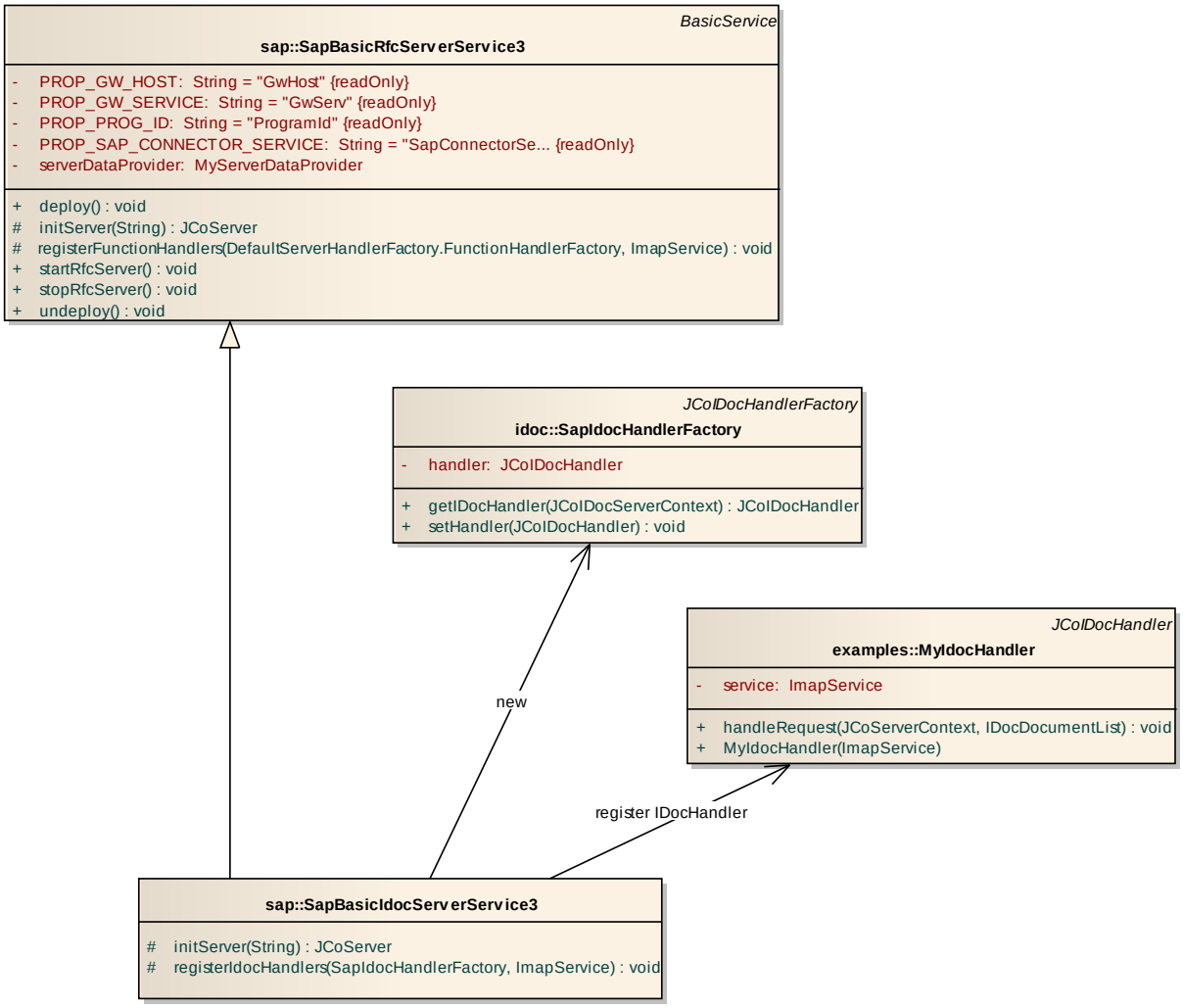
3.4 SAP IDOC Server Service

3.4.1 Voraussetzung

Folgende Klassen müssen vorhanden sein:

de.itm.imap.connector.sap.SapConnection3

de.itm.imap.connector.sap.idoc.SapIdocHandlerFactor
de.itm.imap.services.sap.examples.MyIdocHandler
de.itm.imap.services.sap.SapBasicIdocServerService3



4 Berechtigungen

In diesem Kapitel werden die verschiedenen Berechtigungs-Konzepte, die in IMAP verwendet werden erläutert.

4.1 Service Views

Die Service Views sind verschiedene Sichten im Service Repository Manager. Die verschiedenen Sichten können über die in den IMAP Properties angegebene Datei konfiguriert werden.

```
serviceViews = de/itm/imap/basic/config/serviceViews.xml
```

Die „ServiceViews.xml" hat folgenden Aufbau:

```
<root>
    <service-view>
        <href>/repository.do?actionType=loadServiceDefinition</href>
        <name>Basic</name>
        <role>testuser</role>
    </service-view>
    <service-view>
        <href>/repository.do?actionType=loadServicePropertiesDb</href>
        <name>Properties</name>
        <role>testuser</role>
    </service-view>
    <service-view>
        <name>WSDL</name>
        <href>/repository.do?actionType=showWsdl</href>
        <role>testuser</role>
    </service-view>
    <service-view>
        <name>Execute</name>
        <href>/repository.do?actionType=loadExecutableMethods</href>
        <role>testuser</role>
    </service-view>
</root>
```

Jedes „ServiceView"-Element besteht aus:

- name, dieser wird in der Oberfläche angezeigt
- href, der Link der aufgerufen wird, wenn die View ausgewählt wird.
- role, die Rolle die ein Benutzer benötigt um diese View zu sehen

Die Views werden beim Start für jeden Benutzer in der User Engine geladen. Hier wird überprüft, welche Views der Benutzer sehen darf. **Diese werden ihm dann einmalig zugeordnet**.

5 GUI Implementierung

5.1 Komponenten

Im IMAP Framework gibt es vorgefertigte Komponenten, die zur Entwicklung der Oberflächen benutzt werden können. Um diese Komponenten zu benutzen muss die „imap.tld" in die JSP-Seite eingebunden werden. In den folgenden Unterkapiteln werden diese Komponenten erläutert.

Legende für Attribut-Tabelle:

Normal = Pflichtfeld
Kursiv = Kannfeld

5.1.1 Suchhilfe

Das IMAP Framework bietet die Möglichkeit dem Benutzer eine Suchhilfe für bestimmte Eingabefelder anzubieten. Der Benutzer gibt seine Suchkriterien in das Eingabefeld ein, drückt auf einen Button und erhält eine Tabelle mit den gewünschten Werten, die für dieses Eingabefeld in Frage kommen.

Die Suchhilfe wird über den Tag „matchcode" aus der IMAP-Taglibrary zur Verfügung gestellt und bietet folgende Attribute:

Attribut	Werte	Beschreibung
mcClass	Beliebig	Der qualifizierte Name der Klasse, die die Suchfunktion zur Verfügung stellt
method	Beliebig	Der Methoden-Name der Suchmethode
form	Beliebig	Der Name der Form in der Jsp
field	Beliebig	Der Name des Eingabefeldes in der Jsp
type	single | multi	Single: es kann nur ein Suchergebnis ausgewählt werden Multi: es können beliebig viele Suchergebnisse ausgewählt werden
width	*Zahl*	*Breite des Fensters mit den Suchergebnissen*
height	*Zahl*	*Höhe des Fensters mit den Suchergebnissen*
parameter	*Beliebig*	*Beliebige Zeichenkette, die als Parameter an die Suchmethode übergeben wird.*
maxItems	*Zahl*	*Maximal Anzahl der Suchergebnisse*

Der Java Methodenaufbau sieht wie folgt aus:

```
public void methodenName(MatchcodeForm matchcode, String pattern, String pa-
rameter)throws Exception
{
```

```
        matchcode.getItems().add(new MatchcodeItem("Wert", "Beschreibung"));
}
```

Das Objekt „matchcode" enthält einen Vektor („items") mit den Suchergebnissen, die in einer Tabelle angezeigt werden sollen.

5.1.2 Kalender

Für eine Datumsauswahl bietet IMAP einen Kalender an. Dieser wird über den Tag „calender" angeboten. Folgende Einstellungen sind möglich

Attribut	Werte	Beschreibung
field	Beliebig	Der Name des Eingabefeldes in der Jsp, in das das ausgewählte Datum geschrieben wird
standardDate	*true / false*	*Standard: false* *Gibt an, ob das Standarddatum 31.12.9999 als Button im Kalender zur Verfügung gestellt wird*

Funktionen und Aussehen können in folgenden Dateien angepasst bzw. erweiter werden:

- WebContent/en/basic/datepopup_cal.jsp
- WebContent/res/script/kalender.js

5.1.3 Tabellen

5.1.4 Pager

5.1.5 PrettyPrint

5.2 Mehrsprachigkeit

5.2.1 Sprache definieren

Sprachen werden in eine XML-Datei definiert. Diese XML-Datei muss in den „imap.properties" unter dem Schlüssel „languagesXml" bekannt gegeben werden.

```
# Language File
languagesXml = de/itm/imap/basic/config/languages.xml
```

Der Aufbau der Sprach Xml Datei sieht wie folgt aus:

```
<languages>
    <language>
        <name>en</name>
        <description>English</description>
    </language>
    <language>
        <name>de</name>
        <description>Deutsch</description>
    </language>
</languages>
```

Der Name gibt den Schlüssel an, der für die Sprachdateien benötigt wird (siehe auch 5.2.2). Die Description gibt eine Beschreibung der Sprache an.

5.2.2 Sprachdateien anlegen und veröffentlichen

Im IMAP werden für alle Module eigene Sprachdateien definiert. Die Sprachdateien werden über die im „Struts" zur Verfügung gestellten Funktionen benutzt (siehe auch 5.2.3). Die Sprachdateien werden in der „struts.config" wie folgt veröffentlicht:

```
<!-- Message Resources Configuration -->
<message-resources parameter="de.itm.imap.basic.config.ImapResources"/>
<message-resources key="de.itm.imap.manager.Sidenavigation"
    parameter="de.itm.imap.manager.Sidenavigation"/>
<message-resources key=" de.itm.imap.security.Resources"
    parameter=" de.itm.imap.security.Resources"/>
```

In den „ImapResources" stehen funktionale Schlüssel, die für die Beschriftung der Oberflächen Elemente nicht relevant sind. In der „Sidenavigation" stehen die Beschriftungen für die Seitennavigation. Diese Property-Datei steht im Klasssenpfad „de.itm.imap.manager" mit dem Dateinamen „Sidenavigation.properties". Soll diese Datei jetzt übersetzt werden kopiert man diese Datei in den gleichen Klassenpfad und benenn sie in „Sidenavigation_XX" um, wobei XX für den in Kapitel 5.2.1 definierten Sprachennamen steht (beispielsweise „Sidenavigation_de" für deutsch.

5.2.3 Sprachdateien benutzen

In IMAP können die Sprachdateien wie folgt genutzt werden.

5.2.3.1 In JSPs benutzen

In JSP-Seiten werden die standerdmäßigen Struts-Tags für Beschriftungstexte benutzt.

```
<bean:message bundle="de.itm.imap.security.Resources"
    key="manager.login.label.user" />
```

```
<bean:message bundle="de.itm.imap.security.Resources"
    property="beanProperty" name="bean"  />
```

5.2.3.2 In Klassen benutzen

In Java Klassen benutzt man Beschriftungstexte wie folgt:

```
MessageResources messages = ImapEnv.getMessageResources(request,
    "de.itm.imap.security.Resources");
Locale locale = (Locale)request.getSession()
    .getAttribute("org.apache.struts.action.LOCALE");

String text = messages.getMessage(locale ,"manager.login.label.user");
```

Zuerst wird die gewünschte Beschriftungstextdatei geladen. Anschließend liest man die aktuelle Sprache aus dem Request aus. Und zum Schluss ließt man den gewünschten Wert aus der Sprachdatei aus.

6 Testverfahren

Dieses Kapitel beschreibt die Testverfahren, die in der IMAP Entwicklung empfohlen und eingesetzt werden.

6.1 Grundlagen

6.2 JUnit Tests

In JUnit wird für jede Klasse eine eigene Testklasse angelegt. Diese liegt im gleichen Paket und hat den gleichen Name wie die Klasse mit dem Zusatz Test.

Beispiel:

Klasse:
de.itm.imap.HelloWorld

JUnit Testklasse:
de.itm.imap.HelloWorldTest

In dieser Klasse werden die einzelnen Testfälle für die zu testende Klasse definiert. Zusätzlich werden in allen Paketen, die Testfälle besitzen sogenannte Testsuites eingerichtet, die alle Tests in einem Paket ausführen. Diese Klassen werden AllTests genannt. Diese AllTest Klassen kann man auf jeder Hierarchiestufe einrichten, um größere Pakete testen zu können. Auf der obersten Stufe hätte man dann eine Testklasse die alle JUnit Tests des Projektes aufruft.

6.2.1 Eclipse Plugin

Ab der Eclipse Version 3.2 ist das JUnit Framework bereits integriert. Um einen Test zu erstellen wählt man einfach den Menüeintrag File -> New -> Other -> JUnit Testcase. Nun wählt man die gewünschten Einstellungen für seinen Testfall.

Um den Test durchzuführen wählt man die Testklasse bzw. Suite und wählt den Menüpunkt Run -> Run As -> JUnit Test. Es wird eine neue View geöffnet, der die Auswertung der Testfälle zeigt.

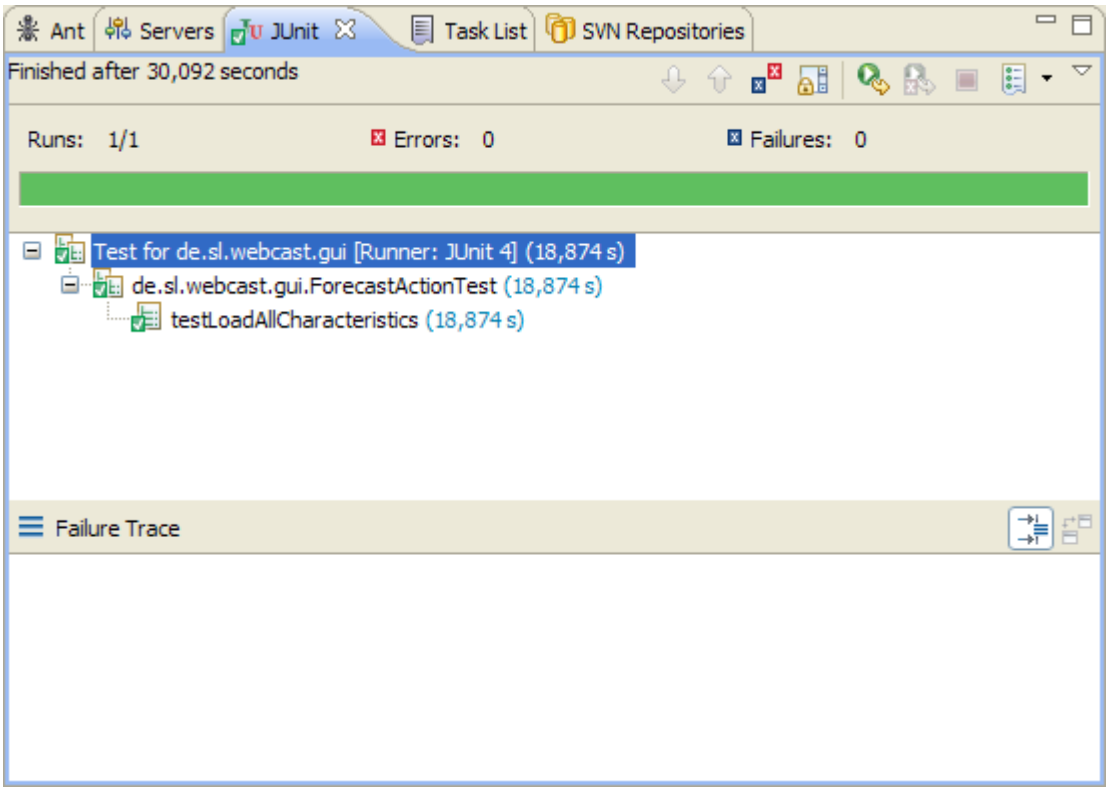


ABBILDUNG 9: JUNIT VIEW

Wichtig: Die JUnit Bibliotheken müssen vor den Web Binlitheken eingebunden werden. Dies kann man im Eclipse in den Projekteinstellungen machen.

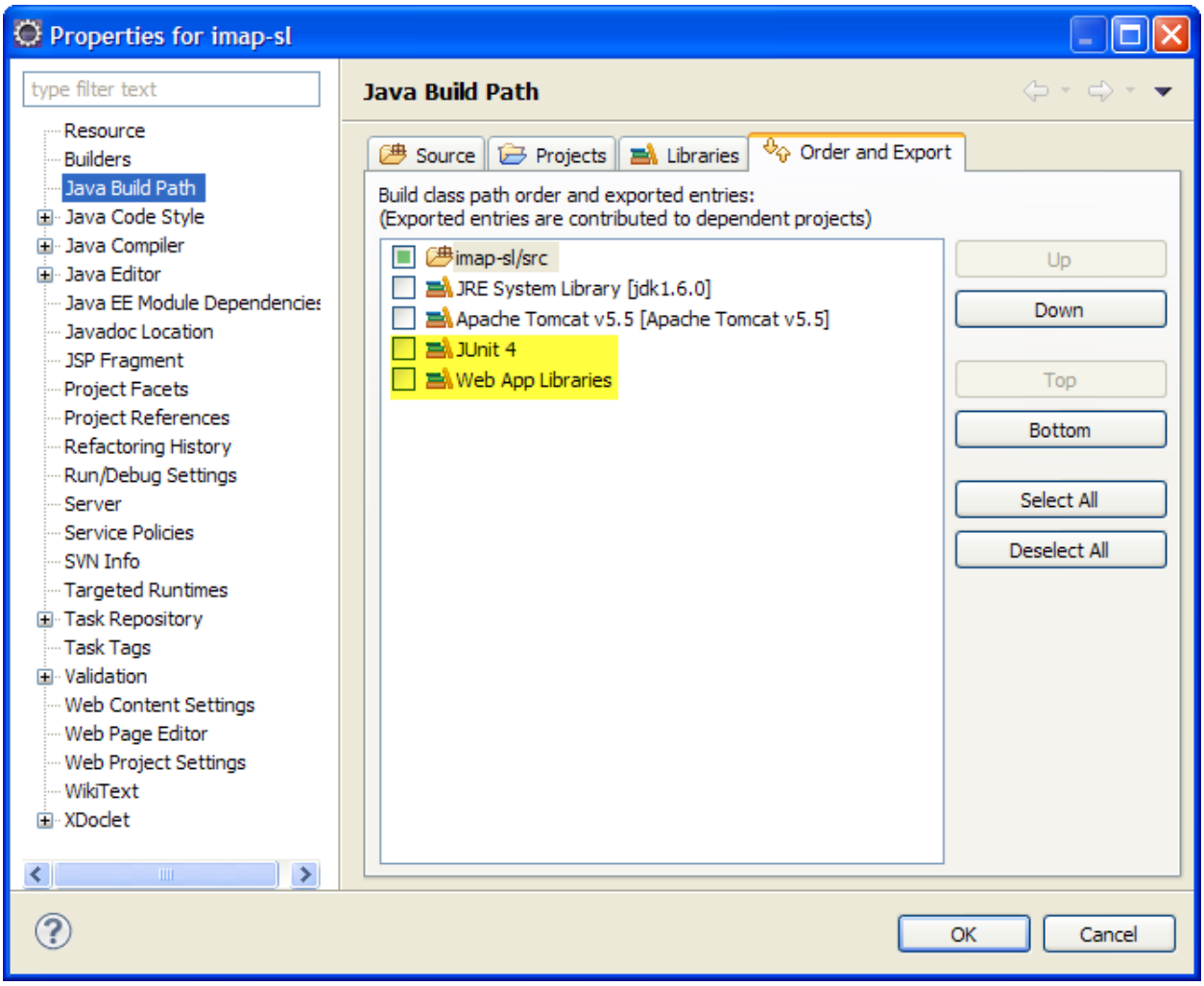


ABBILDUNG 10: JUNIT PROJEKT EINSTELLUNGEN

6.2.2 Testklasse

Eine Testklasse hat immer folgenden Aufbau:

- Klassen-Initialisierungsmethoden (optional)
- Methoden-Initialisierungsmethoden (optional)
- Testmethoden

Die Klasseninitialisierungsmethoden werden beim Start und beim Beenden der Testklasse aufgerufen. Sie eignen sich hervorragend für die Erstellung und Vernichtung von Testdaten. Sie werden wie folgt in die Klasse eingebunden:

```
public static junit.framework.Test suite()
{
    return new JUnit4TestAdapter(HelloWorld.class);
}
```

```
@BeforeClass
public static void setUpBeforClass()throws Exception
{
}


@AfterClass
public static void tearDownAfterClass() throws Exception
{
}
```

Die Methodeninitialisierungsklassen werden vor und nach jedem Aufruf einer Testmethode durchlaufen. Sie eignen sich gut für das neuorganisieren der Testdaten. Diese Methoden besitzen folgenden Aufbau:

```
@Before
public void setUp() throws Exception
{
}

@After
public void tearDown() throws Exception
{
}
```

Zu guter Letzt gibt es noch die eigentlichen Testmethoden. Diese werden ganz normal Durchlaufen und führen die gewünschten Tests durch. Im Falle, das ein Test in dieser Methode nicht erfolgreich war, wird der Durchlauf abgebrochen, der Test wird als nicht erfolgreich markiert und die und die oben genannte Methodeninitialisierungsklasse „After" wird aufgerufen. Die Testmethoden haben folgenden Aufbau:

```
@Test
public void testHelloWorld() throws Exception
{
}
```

Die Annotation „@Test" gibt an, dass die Methode in den Testdurchlauf einbezogen werden soll. Wird dieser Eintrag gelöscht, wird diese Methode nicht mehr als Testmethode erkannt und ausgelassen. Die Benennung der Methode sollte folgenden Aufbau haben:

public void test + „Name der zu testenden Methode(hier helloWorld)"()

In die Testmethoden werden durch asserts verschiedene Tests ausgeführt.

Besipiele:

assertTrue(3+5 == 8);	Test okay!
assertTrue(3+5 == 9);	Test fehlgeschlagen!
assertFalse(3+5 == 9);	Test okay!

Durch asserts können eine Menge verschiedener Tests durchgeführt werden. Hier gibt es eine Übersicht über alle assert-Befehle:

http://junit.org/apidocs/org/junit/Assert.html

6.2.2.1 XML Tests

Im IMAP gibt es eine zusätzliche Bibliothek (XMLUnit), die es erlaubt, in die Testfälle XML Vergleiche einzubauen. Hierzu muss man lediglich die Importe anpassen:

```
//import static org.junit.Assert.*;
import static org.custommonkey.xmlunit.XMLAssert.*;
```

Durch die Änderung des Imports sind alle von jUnit zu Verfügung gestellten assert-Methode weiterhin verfügbar. Es kommen lediglich welche für Vergleiche zwischen XML-Dateien hinzu.

Der folgende Code vergleich Zwei XML-Dateien auf ihre Gleichheit.

```
InputSource isOrg = new InputSource(new FileInputStream(orgFile));
InputSource isComp = new InputSource(new FileInputStream(compFile));

assertXMLEqual(isOrg, isComp);
```

6.2.3 Testsuite

Eine Testsuite kann sowohl verschiedene Testklassen beinhalten, als auch andere Testsuites. Hierdurch lassen sich beliebig viele verschiedene Testsuites auf allen möglichen Paketebenen abbilden. Eine Testuite hat folgenden Aufbau:

```
public class AllTests
{
    public static Test suite()
    {
        TestSuite suite = new TestSuite("Test for de.itm.imap");
        //$JUnit-BEGIN$

        suite.addTest(HelloWorld.suite());
        …
        …

        //$JUnit-END$
        return suite;
    }
}
```

Diese Testsuite wäre eine typische Testklasse für das Paket „de.itm.imap". Zusätzlich könnte man jetzt noch eine Testsuite in der obersten Paketebene erstellen, die wiederum diese Testsuite und eventuelle andere einbindet.

```
public class AllTests
{
    public static Test suite()
    {
        TestSuite suite = new TestSuite("Test for Imap");
        //$JUnit-BEGIN$

        suite.addTestSuite(de.itm.imap.AllTests.class);
        …
        …

        //$JUnit-END$
        return suite;
    }
}
```

7 Kapitel

Text

TABELLE 1: LISTE ALLER VORGÄNGE

7.1 Unterkapitel (Ebene 2)

Text

ABBILDUNG 11: IMAP LOGO

7.1.1 Unterkapitel (Ebene 3)

Wie in (Grimm, 1997) beschrieben. Siehe auch (Rath, 2007)

7.1.1.1 Unterkapitel (Ebene 4)

Text

A Literaturverzeichnis

Grimm, B. (1997). *Simultaion* .... Düsseldorf: Springer.
Rath, T. (2007). *BPEL* . Dortmund: Diplomarbeit.
Red Hat Middleware. (2009). *Hibernate*. Von Hibernate: http://www.hibernate.org/ abgerufen
Sun Microsystems, Inc. (2009). *Java SE Downloads*. Von http://java.sun.com/javase/downloads/index.jsp abgerufen
The Apache Software Foundation. (2009). *Apache Tomcat*. Von http://tomcat.apache.org/download-55.cgi abgerufen

B Abbildungsverzeichnis

C Tabellenverzeichnis

D Änderungsgeschichte

Datum	Name	Beschreibung
25.05.2009	Tobias Rath	Initialisierung und Suchhilfe hinzugefügt
12.06.2009	Dr. Bernd Grimm	Kapitel SAP Services
22.06.2009	Dr. Bernd Grimm	Kapitel Kundenprojekt im Eclipse anlegen
23.06.2009	Tobias Rath	Kapitel Mehrsprachigkeit hinzugefügt
24.06.2009	Tobias Rath	SAP Voraussetzung und Kalender-Tag hinzugefügt
11.08.2009	Tobias Rath	SAP JCO 3.0.2 hinzugefügt
17.08.2009	Tobias Rath	SAP RFC Server hinzugefügt
18.08.2009	Tobias Rath	Berechtigungen
18.08.2009	Grimm	Kapitel „SAP RFC Server Service" überarbeitet Kapitel „SAP IDOC Server Service" hinzugefügt
25.08.2009	Grimm	Kapitel „SAP Services (JCO 3.0.2)" überarbeitet
14.09.2009	Tobias Rath	Kapitel Testverfahren hinzugefügt