

PDA-based Result Submission and Validation with WebServices

PATRICK GLEICHMANN AND REINHOLD SCHAEFER
FACHHOCHSCHULE WIESBADEN, UNIVERSITY OF APPLIED SCIENCES
KURT-SCHUMACHER-RING 18, 65197 WIESBADEN, GERMANY
e-mail: patrick@feedface.com, schaefer@informatik.fh-wiesbaden.de - internet: http://www.wicil.org

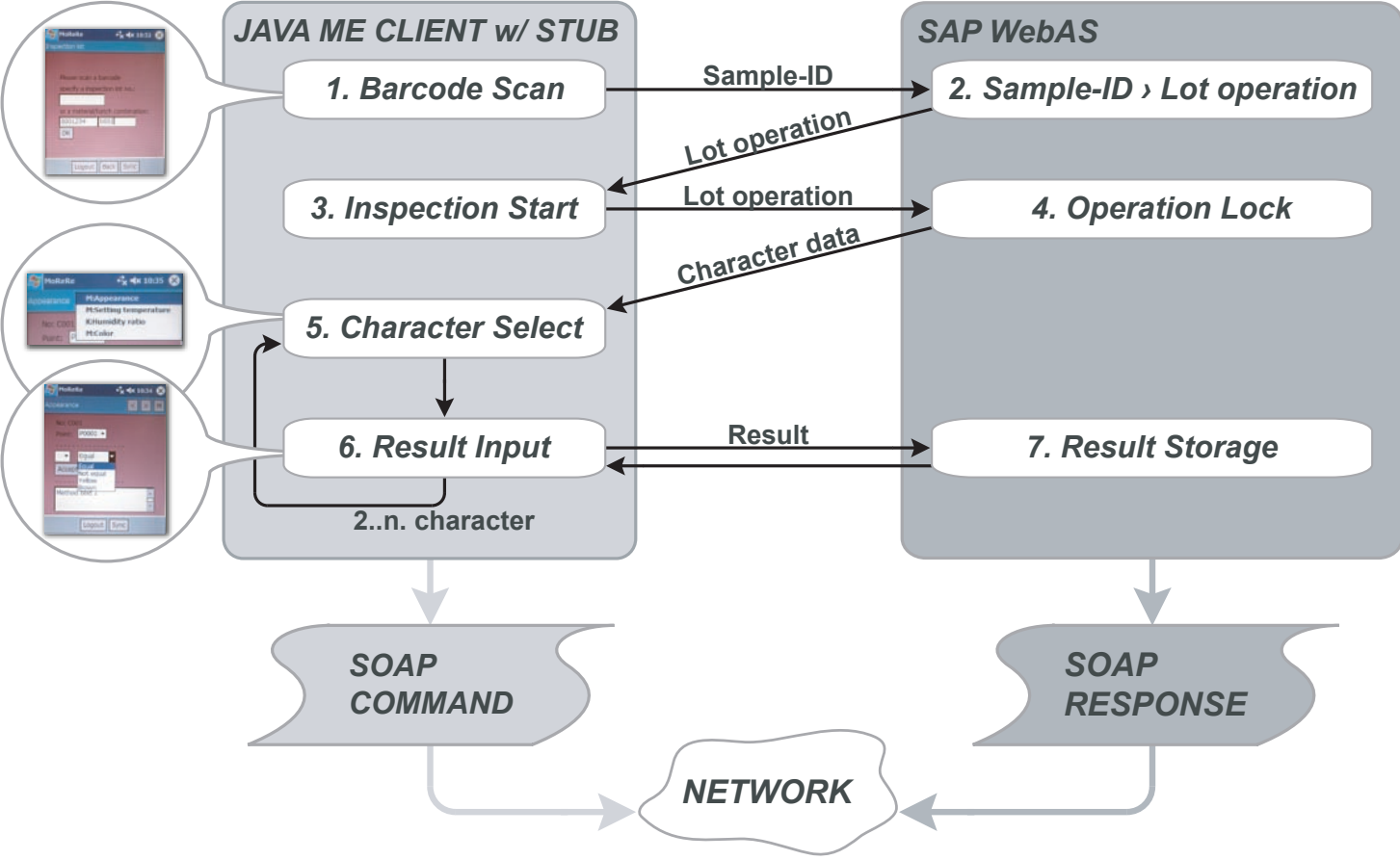


Figure 1: Message exchange & workflow (2.1, page 2)

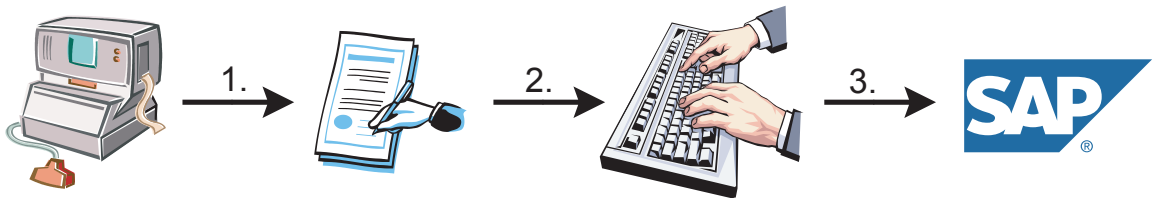


Figure 2: Actual state (2.2, page 2)

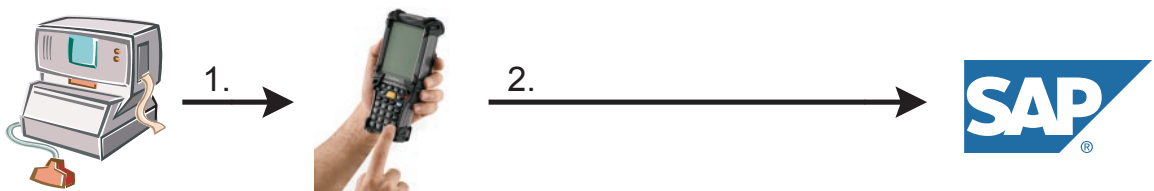


Figure 3: Target state (2.3, page 3)

Content

1. Problems, Goals & Solutions
2. Workflow & States
3. Architecture
4. A SOAP function with SAP: Step by Step
5. Web links

1. Problems, Goals & Solutions

1.1. Problems

- Manual result acquisition slow and error-prone.
- Risk mixing samples up.
- Need to verify groups of results and sign only once.
- Sample tracking in the laboratory complex unless results are entered.
- Necessity to guaranty completeness of documentation incl. raw and result files
> formal procedures like pagination of documents cost time.
- Paper based long term archival laborious.
- Need for extensive employee training due to complicated paper forms.

1.2. Goals

- Minimize and assist manual intervention.
- Prevent mix-ups of samples using positive sample identification.
- Replace paper by electronic acquisition forms.
- Simplify archiving.
- Introduce electronic traceability within the lab.

1.3. Solutions

- Use a PDA with wireless connection to SAP/QM as well as local off-line dialog support and information buffering.
- Introduce barcode labels for sample identification.
- Implement intuitive user guidance with predetermined dialog steps.
- Establish extensive plausibility checks for quantitative results and selection lists for qualitative results.
- Provide digital signatures for all results and comments.
- Replicate status information frequently with the central SAP system.

2. Workflow & States

2.1. Workflow

1. The user scans the sample barcode. The sample-ID then is transferred.
2. SAP looks up the matching inspection lot operation and transfers its number.
3. The user decides to start the inspection of this specific operation.
4. SAP locks the lot operation to prevent duplicated recording. This may fail if the lot operation is already in process.
5. The user selects a character out of the character list.
6. The user enters a result value and confirms it with his electronic signature.
7. SAP stores the result.

2.2. Actual state

1. Manual result acquisition on a paper form
2. Transcription of the result values in a SAP form
3. Confirmation and saving of the SAP form

2.3. Target state

1. Manual result input in a electronic form on the PDA
2. Automatic result transfer to SAP

3. Architecture

3.1. Software:

- SUN Java2ME & SUN Wireless Toolkit (WTK) [1, 2]
- SAP Web Application Server 6.20 [3]

3.2. Webservices

consist of 3 components:

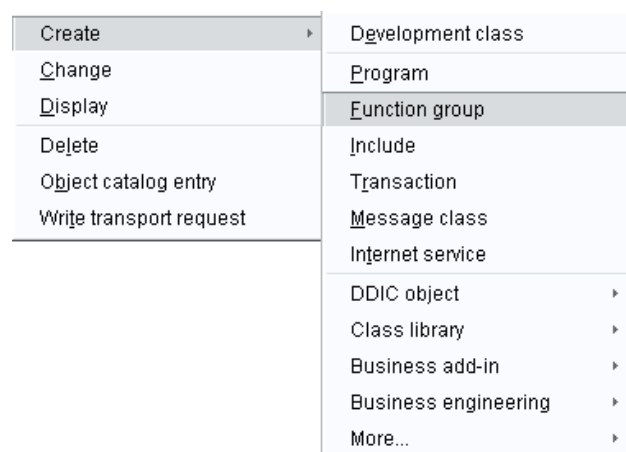
- *WSDL* (Web Services Description Language, [5]) describes the interface of a service end point in a machine-processable format.
A stub compiler normally generates programming language dependent code. The actual access is made by this stub. The *WSDL* document is delivered by the service provider.
- *SOAP* (Simple Object Access Protocol, [4]) is a XML-based protocol for information exchange.
It is a replacement for the outdated RPC mechanism.
- *UDDI* (Universal Description, Discovery and Integration, [6]) is a registry service to manage web services.

4. A SOAP function with SAP: Step by step

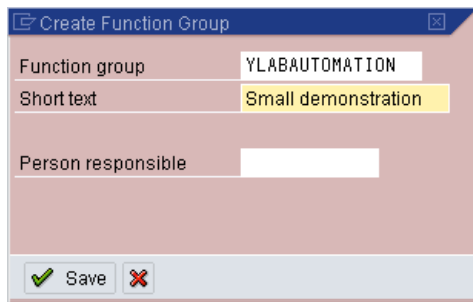
The intention of this chapter is to give you a small overview how easy it is to develop SOAP functions w/ SAP. At the end you'll have a remote accessible SAP function module and a Java2ME client that makes use of the function.

4.1. The function module

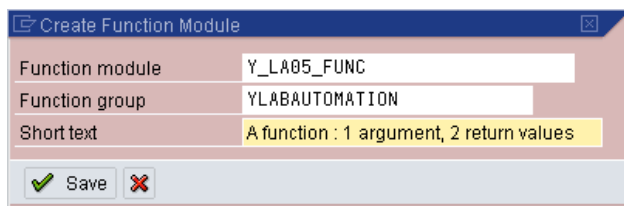
First of all we have to create a *Function group* that will contain our function module. Create it by selecting a *Development class* (e.g. "\$TMP") and a right-click on the class name:



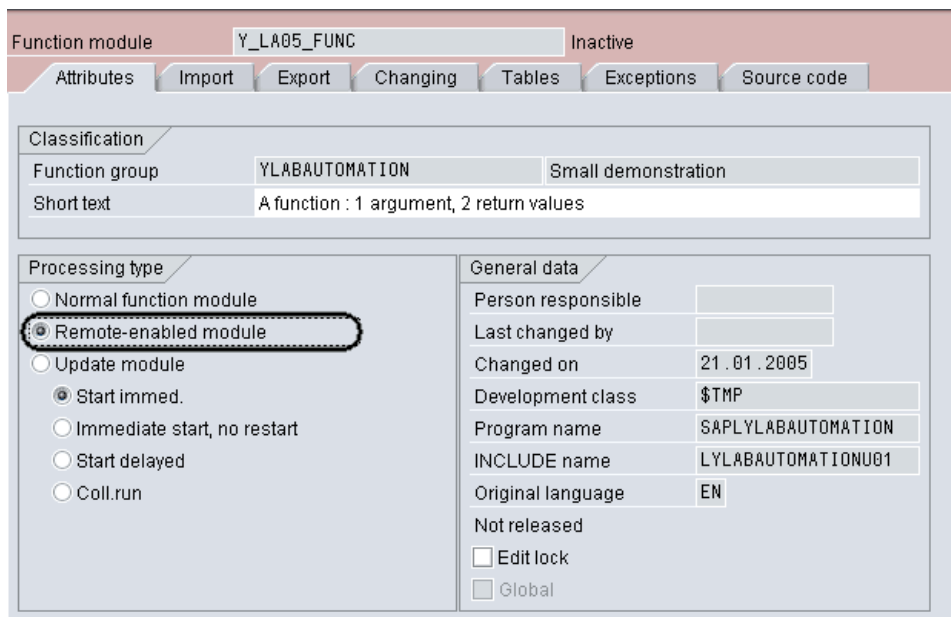
Edit the *Function group* name and the *Short text* according to the screenshot:



Create a *Function module* using the context-sensitive menu as before:



Now comes the most important step: Set the *Processing type* to "Remote-enabled module" - this makes the function external callable:



Open the *Import* tab and insert the following import parameter:

Name	Type/Like	Type	Pass value
I_NUMBER	TYPE	I	✓

Next are the *Export* parameters:

E_MESSAGE	TYPE	STRING	✓
E_SQUARE	TYPE	I	✓

Type can be any data type that's defined in the Data-Dictionary (DDIC).

Our demo function only calculates the square of the input parameter and sets a message-text:

```

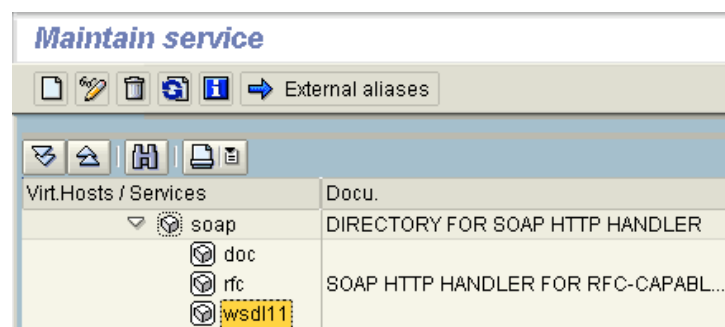
FUNCTION Y_LA05_FUNC.
**-----
** Local interface:
** IMPORTING
**   VALUE(I_NUMBER) TYPE I
** EXPORTING
**   VALUE(E_MESSAGE) TYPE STRING
**   VALUE(E_SQUARE) TYPE I
**-----
*/ Check argument
IF I_NUMBER = SPACE.
  E_MESSAGE = 'Error: No value specified'.
  E_SQUARE = 0.
ELSE.
  E_MESSAGE = 'Successful'.
  E_SQUARE = I_NUMBER * I_NUMBER.
ENDIF.
ENDFUNCTION.

```

Save and activate it and you're done with the function.

4.2. WSDL & Java stub

Make sure that the SAP SOAP and WSDL handlers are actually activated – SAP transaction “SICF”:



Time to fetch the WSDL document for our previously created function module. If you are lucky then you can use the „Web Service Browser“ by simply entering the following URL in your web browser:

```
http://<IP/HOSTNAME>/sap/bc/bsp/sap/webservicebrowser
```

Unlucky? Then use this URL to access the WSDL exporter directly:

```
https://<IP/HOSTNAME>:1443/sap/bc/soap/wsdl11?services=Y_LA05_FUNC
```

Enter your SAP user name and password. Save the resulting file as „Y_LA05_FUNC.wsdl“.

WSCompile can't handle a plain WSDL document, it requires a configuration file – lets call it “Y_LA05_FUNC.xml”:

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl location="Y_LA05_FUNC.wsdl" packageName="labautomation05"/>
</configuration>

```

Basically this means that the “Y_LA05_FUNC.wsl” is in the same directory as the configuration file, and the package of the generated files should be “labautomation05”.

Execute WSCompile with the following command – please note that wscompile has to be in your command path:

```
wscompile -gen Y_LA05_FUNC.xml
```

The „-gen“ option is the short form of „-gen:client“ and generates the client stub including all import and export-classes.

The resulting files are (.java & .class):

- Y_LA05_FUNC.java
- Y_LA05_FUNCPortType_Stub.java
- Y_LA05_FUNCPortType.java
- Y_LA05_FUNCResponse.java – returned object (= Exports)

4.3. The client application

```
1 package labautomation05;
2
3 import javax.xml.rpc.Stub;
4 import java.rmi.RemoteException;
5
6 public class LA05FuncDemo {
7     public static void main(String[] args) {
8         // Create a new stub
9         Y_LA05_FUNCPortType_Stub stub = new Y_LA05_FUNCPortType_Stub();
10
11         // Set the SAP authentication data
12         stub._setProperty(Stub.USERNAME_PROPERTY, "john");
13         stub._setProperty(Stub.PASSWORD_PROPERTY, "doe");
14
15         // Call the SAP function
16         try {
17             Y_LA05_FUNCResponse resp = stub.y_LA05_FUNC(4);
18             System.out.println("Message: "+resp.getE_MESSAGE());
19             System.out.println("Result: "+resp.getE_SQUARE());
20         } catch (RemoteException re) {
21             re.printStackTrace();
22         }
23     }
24 }
25
```

Build and execute it – make sure that you include the “j2me-ws.jar” (Wireless Toolkit, [2]) in your class-path. Some J2ME foundation frameworks don’t contain a Base64 class, which is required for the login encoding. You can extract it from the “midpapi10.jar” (also WTK [2]).

The output has to be:

```
Message: Successful
Result: 16
```

4. Web links

- [1] Java 2 Micro Edition (ME):
<http://java.sun.com/j2me/index.jsp>
- [2] Java Wireless Toolkit (e.g. WSCompile):
<http://java.sun.com/products/j2mewtoolkit/index.html>
- [3] SAP Web Application Server:
<http://www.sap.com/solutions/netweaver/webappserver/index.aspx>
- [4] XML Protocol Working Group – SOAP specification:
<http://www.w3.org/2000/xml/Group/>
- [5] Web Services Description Working Group – WSDL spec.:
<http://www.w3.org/2002/ws/desc/>
- [6] UDDI specification:
<http://www.uddi.org>