Models
Exercise

Chapter: **Models, Web Services**

Theme: **Using Web Dynpro to access a Web Service**

At the end of this Exercise, you are able to:

- Access a Web service from Web Dynpro

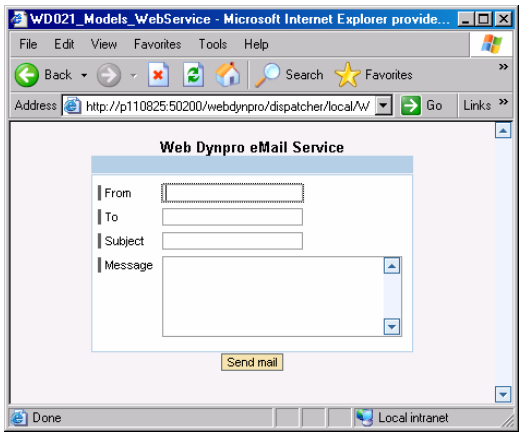
1 Development Objectives

Accessing an e-mail Web service

In this exercise, you will develop a Web Dynpro application for sending an e-mail message, using an e-mail Web service provided by an external service provider. The user interface of this Web application will consist of a simple input form for editing the addresses of senders and recipients, the subject, and the actual e-mail message, and a button for sending the message. A message will be displayed in the Web browser to tell the user whether or not the e-mail message was successfully sent. The use of the e-mail Web service is enabled by an appropriate **model** (auxiliary and communication classes) generated by the Web Dynpro tools. At runtime, the data entered by the user of the application is passed to the model through the data binding between the input fields and the context elements, and through the model binding of these context elements. The model communicates with the Web service through a client stub (a Java object that acts as a proxy for the Web service).

Note: SAP AG does not accept any responsibility regarding the availability and quality of the external e-mail service used in this exercise.

2 Result

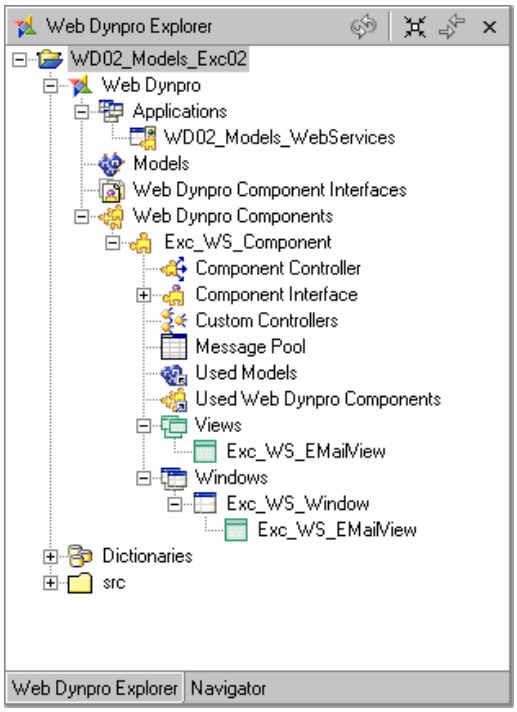


By the end of this exercise, you will be able to:

- Create a model to be used for connecting an external Web service from within the Web Dynpro project.
- Design a simple view layout for sending an e-mail message
- Perform the implementation for availing of the e-mail Web service used.

3 Prerequisites

You have launched the SAP NetWeaver Developer Studio.
You have selected the Web Dynpro perspective.
You have opened the project *WD02_Models_Exc02.*



For your convenience, you can start developing with a predefined Web Dynpro application.

The graphic on the left shows the predefined project structure of this exercise.

4 Overview: Developing

4-1 Create a Web service model.
To create a model that is based on a certain Web service, you first require the URL address through which the corresponding WSDL description can be accessed. If the address is known, you can then easily create an appropriate Web Dynpro model easily.

4-1-1 Generate a Model from the WSDL Description, which can be obtained via the following URL
http://webservices.matlus.com/scripts/emailwebservice.dll/wsdl/IemailService

4-1-2 Make HTTP Proxy Settings. Ask your trainer for the settings appropriate for your network environment.

4-2 Create the binding: Component Controller Context and Model
Each Web Dynpro component is supplied with an associated Component Controller. This controller is responsible for retrieving the data required by the Email Web service to send the e-mail. Accordingly, it must be able to map the corresponding input and output structures of the e-mail model. To do this, you need to bind the context of the component controller to the created Web service model. You can declare this model binding between the controller context and the model with the Data Modeler, available as one of the Web Dynpro tools.

4-2-1 Add the Model defined in Step 4-1 to the Web Dynpro Component using the Data Modeler.

4-2-2 Bind the component controller context to the Web service. Rename the Context Model node to *WebServiceEmail*.

4-3 Map the View Context Elements to the Component Context Elements.
In the last section, a structure for context model elements was created in the context of the component controller. This structure is bound to generated model classes. These model classes contain the data required for sending the e-mail as well as the return data belonging to the response of the Web service.
To be able to access this context structure even outside of a view context, we apply the concept of Context Mapping.

4-3-1 Defining a Context Mapping in the Data Modeler.

4-4 Update the View *Exc_WS_EMailView*

4-4-1 Define the Data Binding between the value of the input field of the View *Exc_WS_EMailView* and the corresponding context model attributes.

4-4-2 Create the Action *SendEmail*.
To trigger sending the email message from the view *Exc_WS_EMailView* using the Web service, you need an associated Action.

4-4-3 Bind the action *SendEmail* to the *onAction* event of the UI element *SendButton*.

4-5 Implement the source code for sending the Email via the Web Service Connection.

4-5-1 Implement the Generic Event Handler *wdDoInit()* of the View Controller.

- Create an instance of the appropriate model adapter class (Type *Request_IEmailService_sendMail*).
- Bind this instance (*req*) to the context model node (Method *wdContext.nodeWebServiceEmail().bind(req)*).

The model object *req* passes its data to the suitable Java proxy, which then communicates with the actual Web service.

4-5-2 Implement the action event handler *onActionSendEmail()*.

- First create a reference (*msgMgr*) to the components message manager (Type *IWDMessageManager*). This is necessary to report messages.

Because errors can occur, the following instruction must be located inside a *try - catch* block.

- Next, the Request has to be send, using the method *wdContext.currentWebServiceEmailElement(). modelObject().execute().*
- Invalidate the model node *response.*
- The response of the service call can be obtained by the method *wdContext.currentResponseElement().getResult();*
- Convert the response to a String.
- If a response is received report the success (method *msgMgr.reportSuccess(…)*), otherwise report a warning (*msgMgr.reportWarning(…)*).
- If errors occurred during communication, report the message (method *msgMgr.reportException(…)*).

The actual Web service is now called using the **execute()** method of the model object currently stored in the context model node. This already contains the reservation data entered by the user (through *data binding* and *context mapping*). The data stored in the component controller context is a copy of the data stored in the model, that is, the one does not directly reference the other. Therefore, the view context bound through context mapping also does not yet contain the returned results of the Web service call executed previously and stored in the model.

As an application developer, you therefore need to explicitly *invalidate* the model node **response.** The response data most recently stored in the model is then transmitted to the corresponding context node element.

The returned result (in the example application this is just a single integer value) is then displayed in an appropriate message text in the *message bar* of the Web Dynpro application.

5 Overview: Building, Deploying, and Running

Deploy and run the Web Dynpro application.