Using an Email Web Service in Web Dynpro

Release 646

HELP.BCJAVA_START_QUICK_ADVANCED

Copyright

Icons in Body Text

Icon	Meaning
⚠	Caution
💬	Example
💡	Note
🧭	Recommendation
syn	Syntax

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see *Help on Help* → *General Information Classes and Information Classes for Business Information Warehouse* on the first page of any version of *SAP Library*.

Typographic Conventions

Type Style	Description
Example text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation.
`Example text`	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

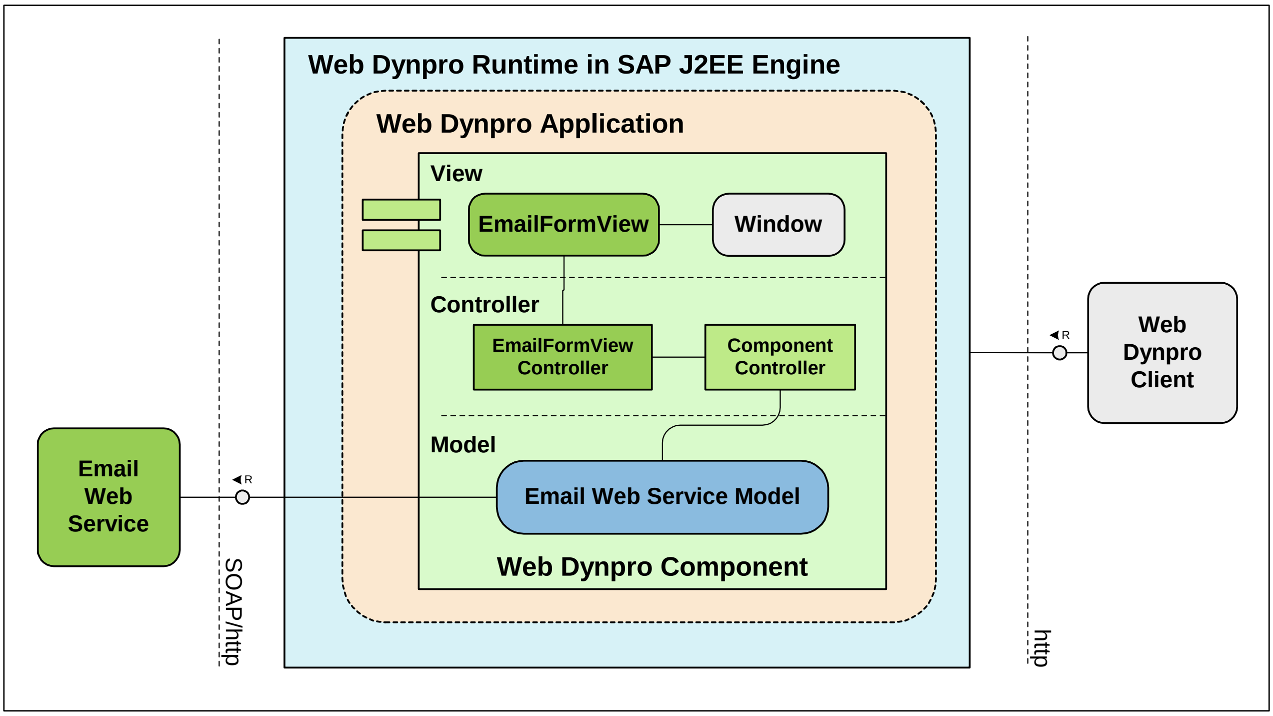
Using an Email Web Service in Web Dynpro

The Task

In this tutorial, 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.

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.



The rapid implementation of this Web Dynpro application is characterized by a completely declarative development process, in which the required implementation workload can be reduced to just a few lines of Java code.

Objectives

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

- ✓ Create a model to be used for connecting an external Web service from within the Web Dynpro project
- ✓ Declare a context node in the component controller and create a connection to the model.
- ✓ Perform context mapping between the view controller and the component controller
- ✓ Design a simple view layout for sending an e-mail message

- ✓ Create an action and implement an event handler
- ✓ Create the data binding from UI elements to context attributes
- ✓ Display the message texts in the Web Browser using a UI service provided by the Web Dynpro runtime environment
- ✓ Perform the implementation for availing of the e-mail Web service used.

Prerequisites

Systems, Installations, and Authorizations

- You have launched the SAP NetWeaver Developer Studio.
- You have access to the J2EE Engine (Release 6.30)
- You can connect to a database instance of the SAP DB.
- You have an open Internet connection.
- The service provider e-mail Web service used in the tutorial is available.

> ⚠
>
> SAP AG cannot guarantee the availability of the third-party Web service used in this tutorial and does not assume any responsibility for the quality of this service.

Knowledge

- You have acquired some basic experience with Web Dynpro applications - for example, by working through the *Welcome Quickstart Guide* (Creating Your First Web Dynpro Application [Extern]).
- Basic knowledge of Java would be an advantage.

Next step:

Creating a Project Structure [Seite 7]

Creating a Project Structure

You can now start creating the project structure that is the basis for the Web service example application. To include all the development objects, first create a new Web Dynpro project. Add a new Web Dynpro component to this project that can be displayed in the Web Browser using the corresponding Web Dynpro application.

Procedure

> The following description does not list every single step in detail, since you have already become familiar with a similar procedure in the Welcome Quickstart Guide [Extern].

Execute the following steps so that you can continue afterwards with actually developing the Web Dynpro example application (view layout, connection to the public Email Web service, data binding, context structures, model binding, and so on).

Creating a Web Dynpro Project

1. Choose File → New Project → Web Dynpro Project:
   The Web Dynpro Project wizard appears.
2. Enter the project name **`WebDynpro_EmailWS`** and leave all the standard settings for the project unchanged.

Creating a Web Dynpro application with a Web Dynpro component

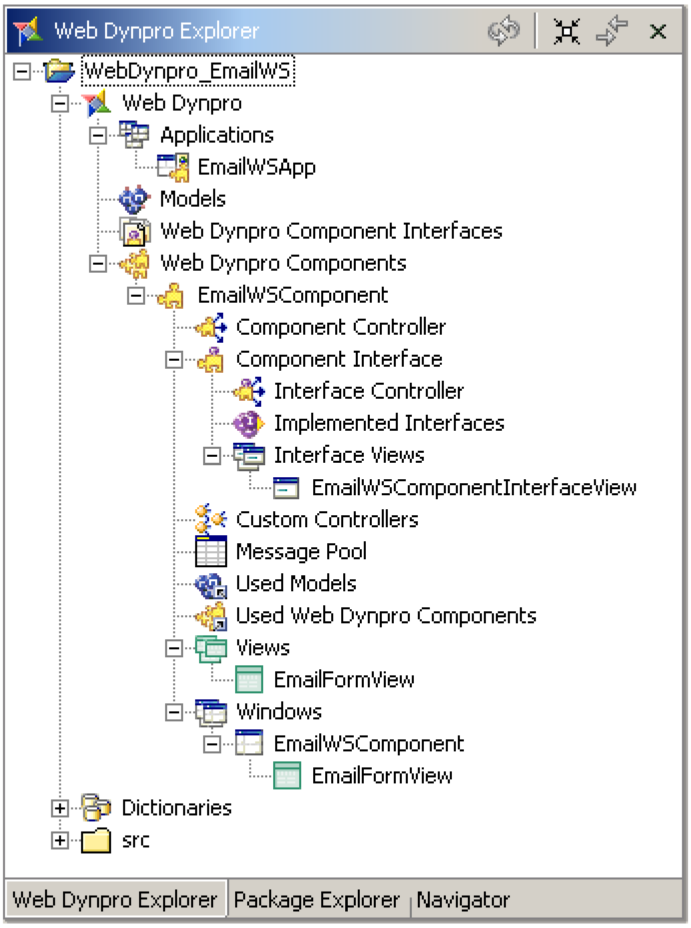
1. To open the wizard, open the context menu for the *Applications* node and choose *Create Application*.
2. Enter the name **`EmailWSApp`** for the Web Dynpro application and the package name **`com.sap.tc.webdynpro.tutorials.emailws`**.
3. Accept the default setting (that is, *Authentication* is deselected) and *Next*.
4. Select the radio buttion *Creat a new component* and choose *Next*.
5. Enter **`EmailWSComponent`** as the name for this Web Dynpro component. Leave the package name **`com.sap.tc.webdynpro.tutorials.emailws`** and all the other default settings unchanged.
6. Choose *Finish*.

Embedding the Web Dynpro View in the Window

1. In the *Web Dynpro Explorer*, expand the *WebDynpro* node and open the context menu for *Web Dynpro Components – EmailWSComponent – Windows - EmailWSComponent*.
2. Choose Embed View and, in the wizard, select the option Embed New View. Choose Next.
3. Enter the name **`EmailFormView`** for the view you are about to create and choose Finish.

Result

You have now created the basic project structure for the new Web Dynpro project **`WebDynpro_EmailWS`**. In the *Web Dynpro Explorer*, the following nodes are displayed for the structure:



You have now made all the preparations necessary to develop the specific functions of the example application.

Next step:

Creating a Web Service Model [Seite 9]

Creating a Web Service Model

Before you can use the e-mail Web service provided by an external provider within this Web Dynpro example application, you need to create the appropriate **model**. You will do this in the next step. At runtime, this model performs the data exchange between a Web Dynpro component and the Web service end point.

Within a Web Dynpro component, business data is stored in separate context structures (consisting of context nodes, context node elements, and context attributes). In this example, *Subject*, *E-mail Text*, *Sender Address*, and *Recipient Address* are the context attributes for the context node *WebServiceEmail*. To transfer the email information entered by the user to the Web service and to receive the response, you need a model that provides the necessary auxiliary classes and communication classes.

You can use the Web Dynpro tools to generate such a model for a particular Web service description. The model mainly consists of Java proxies (*client stubs* for communication with the Web service), as well as special model classes that you can use to link context structures to the model.

The following is a description of how to generate a Web Dynpro model from the WSDL description of the e-mail Web service. (WSDL is an XML format used to describe network services.) This WSDL description is made available by the service provider. If your HTTP connection is linked using a proxy host, you need to make some additional settings.

⚠

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

Prerequisites

□ The service provider can load the WSDL description of the e-mail Web service. The Web service described functions correctly.

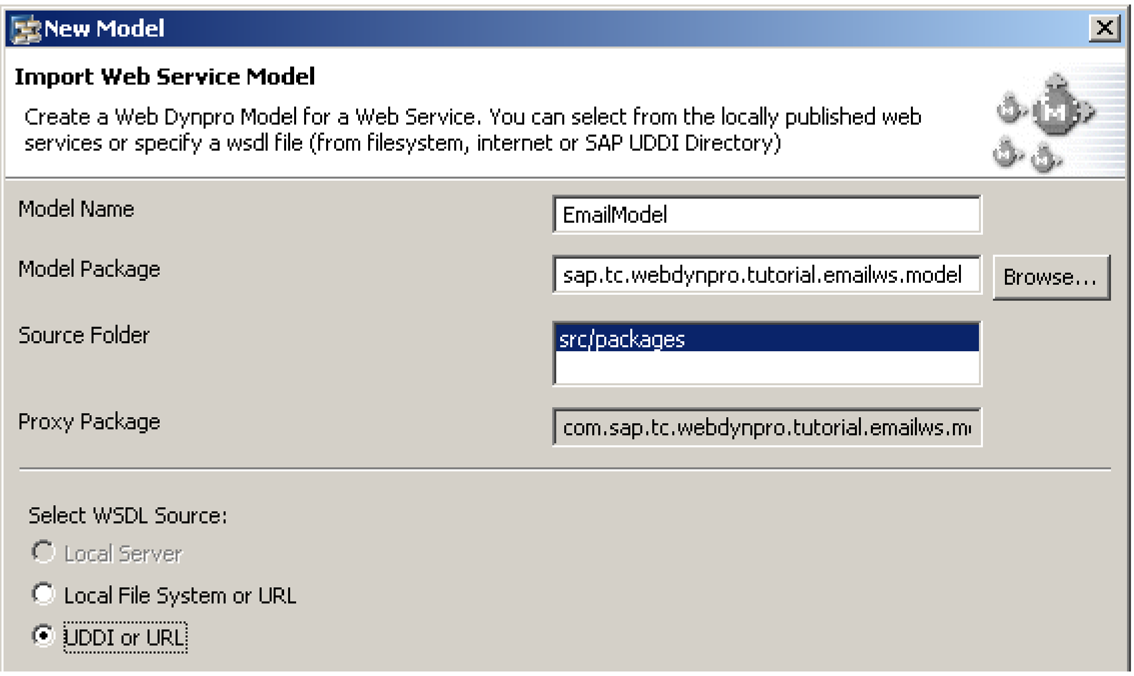
□ The structure of your project **`WebDynpro_EmailWS`** is currently displayed in the *Web Dynpro Explorer*.

Procedure

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.

Generating a Model from the WSDL Description

1. In the project structure, expand the node *Web Dynpro* → *Models*.
2. From the context menu, choose *Create Model*. The appropriate wizard appears.
3. Choose the *Import Web Service Model* option, followed by *Next*.
4. Enter the name **`EmailModel`** as the model name and **`com.sap.tc.webdynpro.tutorial.emailws.model`** as the package name.
5. Under *Select WSDL Source*, choose the radio button *UDDI or URL*, followed by *Next*.



Import Web Service Model

Create a Web Dynpro Model for a Web Service. You can select from the locally published web services or specify a wsdl file (from filesystem, internet or SAP UDDI Directory)

Model Name: EmailModel

Model Package: sap.tc.webdynpro.tutorial.emailws.model

Source Folder: src/packages

Proxy Package: com.sap.tc.webdynpro.tutorial.emailws.m

Select WSDL Source:

○ Local Server

○ Local File System or URL

● UDDI or URL

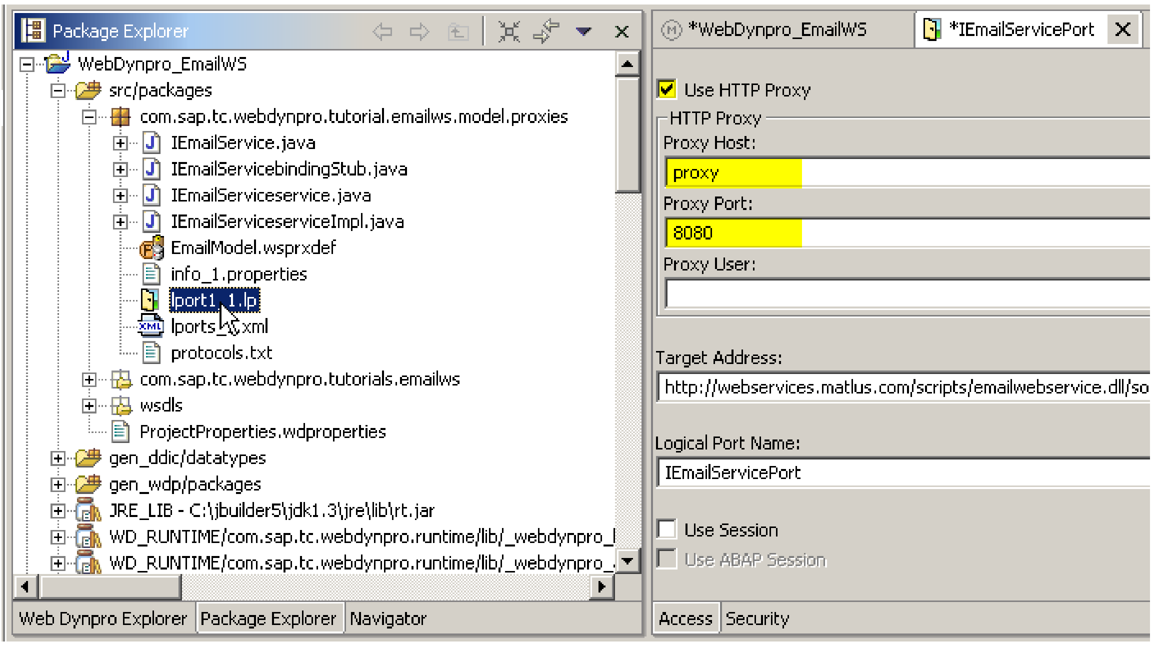
6. Enter the following WSDL description for the Web service in the *Wsdl* field: `http://webservices.matlus.com/scripts/emailwebservice.dll/wsdl/IemailService`
7. You do not need to make any entries in the next step, *Proxy Definition / URI Package Mappings*. Close the input dialog by choosing *Finish*.

The corresponding Java proxies are then generated as client stubs, and the model classes are generated for the subsequent binding of context elements.

Making HTTP Proxy Settings

If your HTTP connection to the e-mail Web service uses an HTTP proxy, you also need to make the following settings:

1. Go from the *Web Dynpro Explorer* to the *Package Explorer.*
2. Open the node **`src/packages – com.sap.tc.webdynpro.tutorials.emailws.model.proxies`**
3. Choose the file **`lport1_1.lp`**.
4. After selecting the checkbox *Use HTTP Proxy*, make the appropriate entries in the fields *Proxy Host* and *Proxy Port*:



Package Explorer

```
WebDynpro_EmailWS
  src/packages
    com.sap.tc.webdynpro.tutorial.emailws.model.proxies
      IEmailService.java
      IEmailServicebindingStub.java
      IEmailServiceservice.java
      IEmailServiceserviceImpl.java
      EmailModel.wsprxdef
      info_1.properties
      lport1_1.lp
      lports_1.xml
      protocols.txt
    com.sap.tc.webdynpro.tutorials.emailws
    wsdls
    ProjectProperties.wdproperties
  gen_ddic/datatypes
  gen_wdp/packages
  JRE_LIB - C:\jbuilder5\jdk1.3\jre\lib\rt.jar
  WD_RUNTIME/com.sap.tc.webdynpro.runtime/lib/_webdynpro_
  WD_RUNTIME/com.sap.tc.webdynpro.runtime/lib/_webdynpro_
```

Web Dynpro Explorer | Package Explorer | Navigator

*WebDynpro_EmailWS | *IEmailServicePort

☑ Use HTTP Proxy

HTTP Proxy

Proxy Host: proxy

Proxy Port: 8080

Proxy User:

Target Address: http://webservices.matlus.com/scripts/emailwebservice.dll/so

Logical Port Name: IEmailServicePort

□ Use Session

□ Use ABAP Session

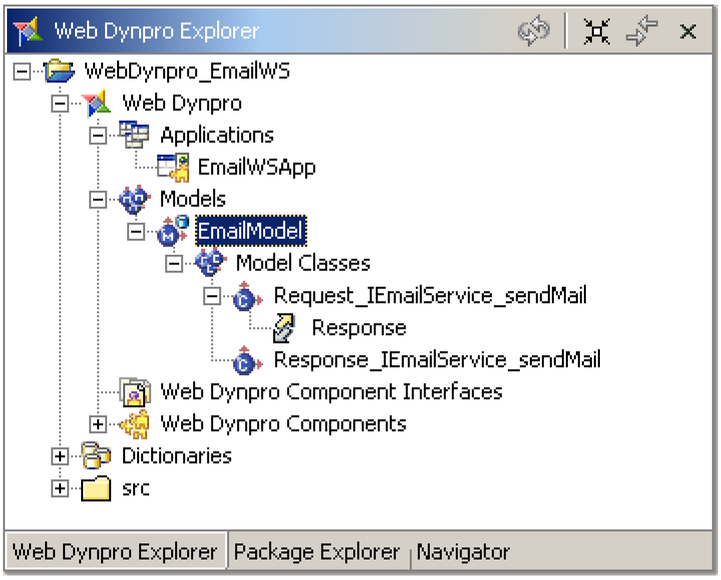
Access | Security

The field *Proxy Host* represents the host name or the IP address of the proxy server, and *Proxy Port* is the port to which the proxy server listens.

5. Save your settings by choosing (*Save Editor Contents*) in the toolbar underneath the menu bar.

Result

After the Web service model has been imported, the corresponding entries are inserted under the *Model* node in the *Web Dynpro Explorer*.



Web Dynpro Explorer

```
WebDynpro_EmailWS
  Web Dynpro
    Applications
      EmailWSApp
    Models
      EmailModel
        Model Classes
          Request_IEmailService_sendMail
            Response
          Response_IEmailService_sendMail
    Web Dynpro Component Interfaces
    Web Dynpro Components
  Dictionaries
  src
```

Web Dynpro Explorer | Package Explorer | Navigator

The nodes contained in the *EmailModel* represent the *Model Classes* () and their relations () – that are visible to you as the application developer. You can subsequently bind the appropriate context elements to these classes and relations. This structure description is independent of the specific model implementation (RFC, XML, Web service) of the imported model. Within a Web Dynpro component, Web Dynpro contexts serve as storage locations for structured data (for example, table data, form entries). Using the *Data Binding* function between UI elements and context elements, this data can easily be displayed in the Web Browser.

Next step:

Creating the Binding: Component Controller Context and Model [Seite 13]

Creating 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 with 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.

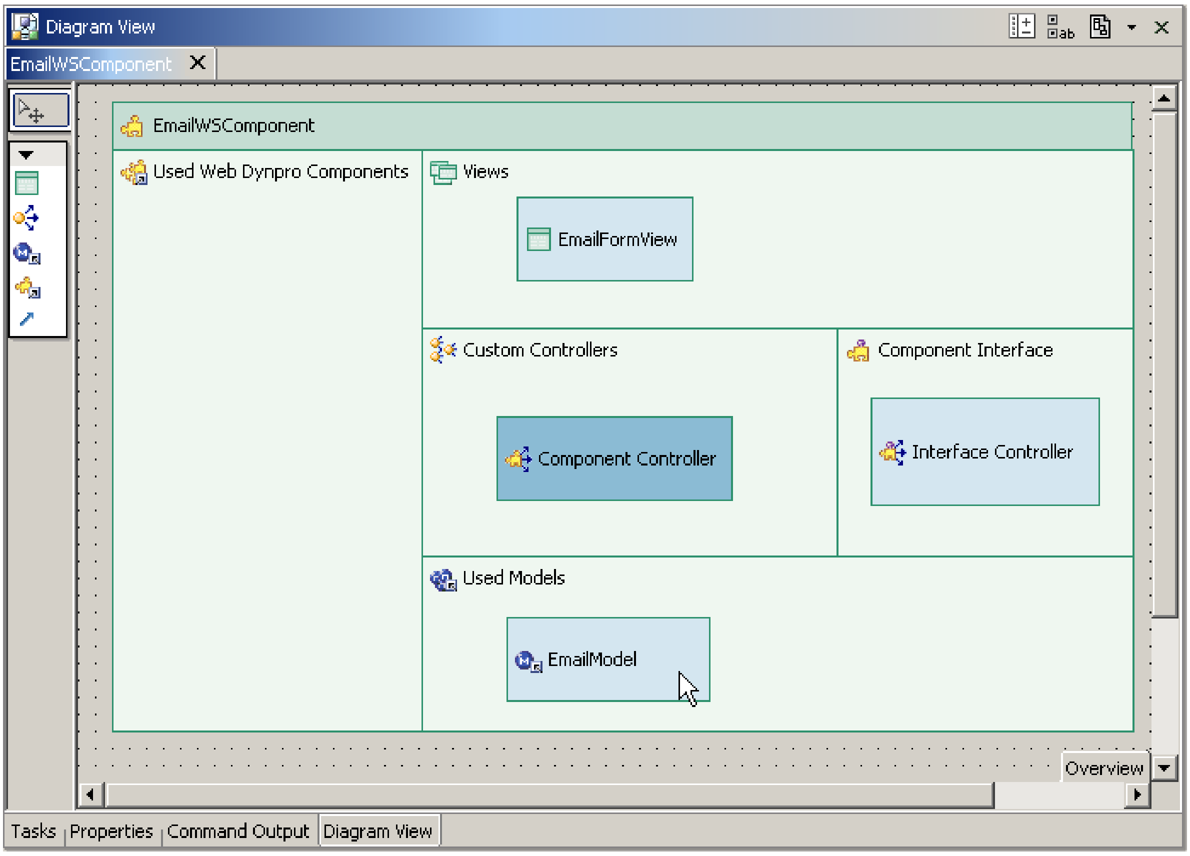
Prerequisites

- The structure of your project **`WebDynpro_EmailWS`** is currently displayed in the *Web Dynpro Explorer*.

Procedure

Adding a Model to the Web Dynpro Component with the Data Modeler

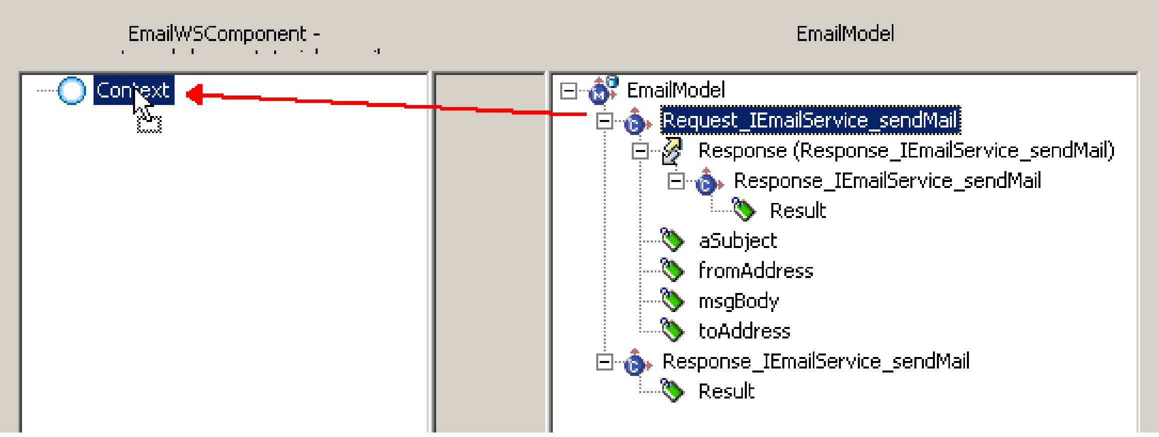
1. Select the node *WebDynpro_EmailWS* → *Web Dynpro* → *Web Dynpro Components* → *EmailWSComponent*, then choose Open Data Modeler from the context menu entry.
2. In the toolbar on the left, choose the Add a model to the component icon. The icon will turn gray.
3. Place the cursor on the Used Models area and left-click.
4. Select EmailModel and choose Ok.
5. The structure of the Web Dynpro component EmailWSComponent will look like this in the Data Modeler:



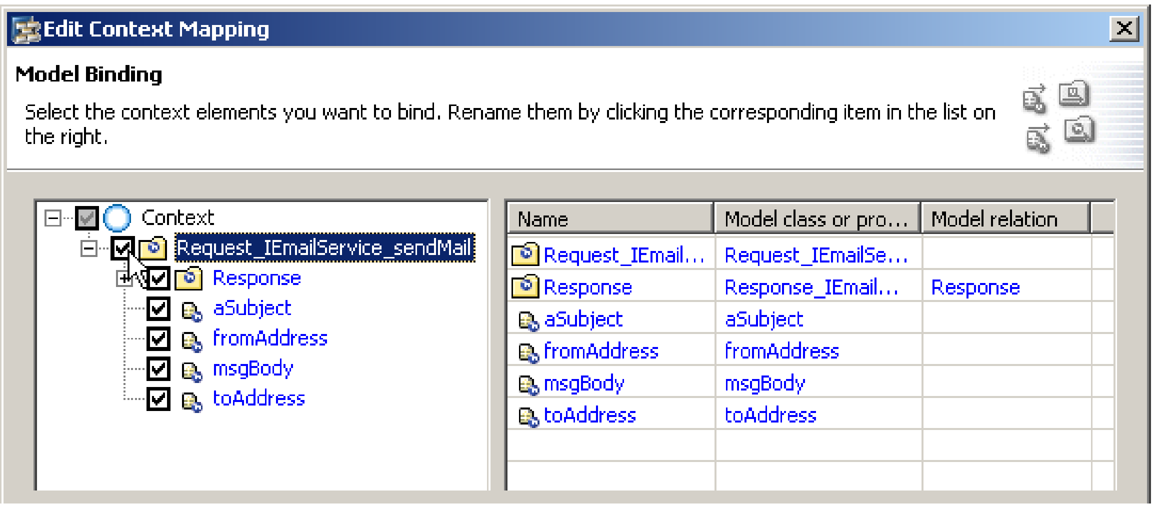
Binding the component controller context to the Web service

In the *Data Modeler,* you can easily declare the connection between the context of the component controller and the Web service model you have created.

1. Open the Data Modeler.
2. In the left toolbar, choose *Create a data link.*
3. Starting above the Component Controller rectangle, press the left mouse button, and keep it pressed.
4. Draw a line to the EmailModel rectangle and release the left mouse button. The Model Binding Wizard starts automatically.
5. Drag the node of the model class **`Request_IEmailService_sendMail`** in the *EmailModel* to the root node of the component controller context, and drop it.


EmailWSComponent -
EmailModel
Context
EmailModel
Request_IEmailService_sendMail
Response (Response_IEmailService_sendMail)
Response_IEmailService_sendMail
Result
aSubject
fromAddress
msgBody
toAddress
Response_IEmailService_sendMail
Result

6. In the dialog box that appears, select the model node **`Request_IEmailService_sendMail`**:



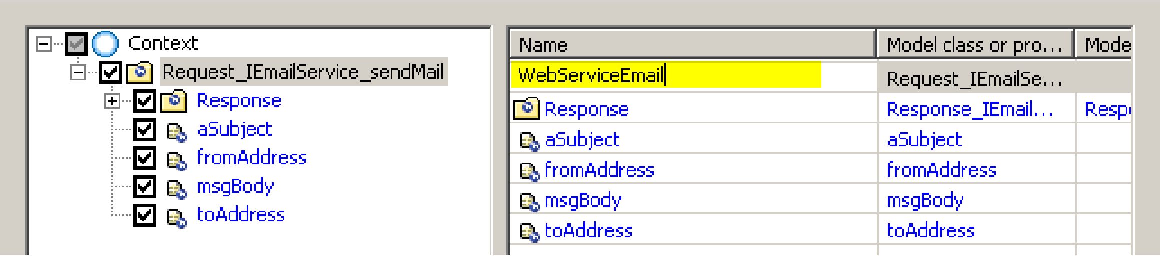
Edit Context Mapping

Model Binding

Select the context elements you want to bind. Rename them by clicking the corresponding item in the list on the right.

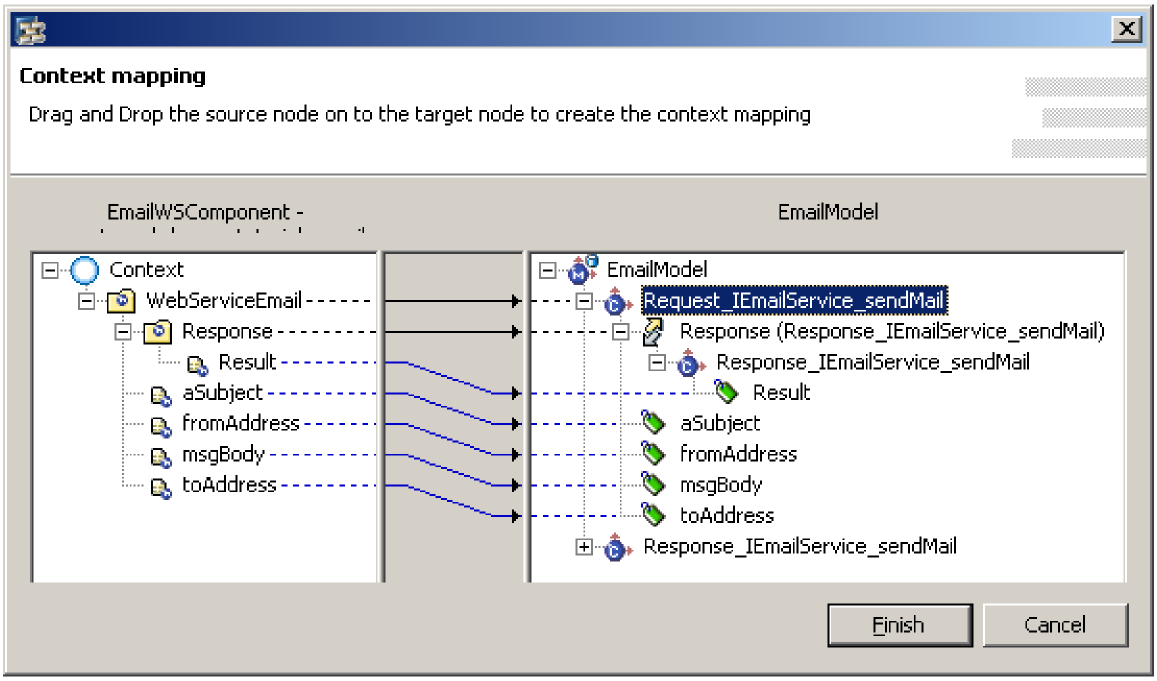
Name	Model class or pro...	Model relation
Request_IEmail...	Request_IEmailSe...	
Response	Response_IEmail...	Response
aSubject	aSubject	
fromAddress	fromAddress	
msgBody	msgBody	
toAddress	toAddress	

7. Enter the name **`WebServiceEmail`** for the new model node **`Request_IEmailService_sendMail`**, which is bound to the model class of the same name, by editing the appropriate entry in the *Name* column, and then choosing *Ok*.



Name	Model class or pro...	Mode
WebServiceEmail	Request_IEmailSe...	
Response	Response_IEmail...	Resp
aSubject	aSubject	
fromAddress	fromAddress	
msgBody	msgBody	
toAddress	toAddress	

8. In the dialog box that appears, the declared model binding between the model node *WebServiceEmail* and the corresponding model class shown graphically:

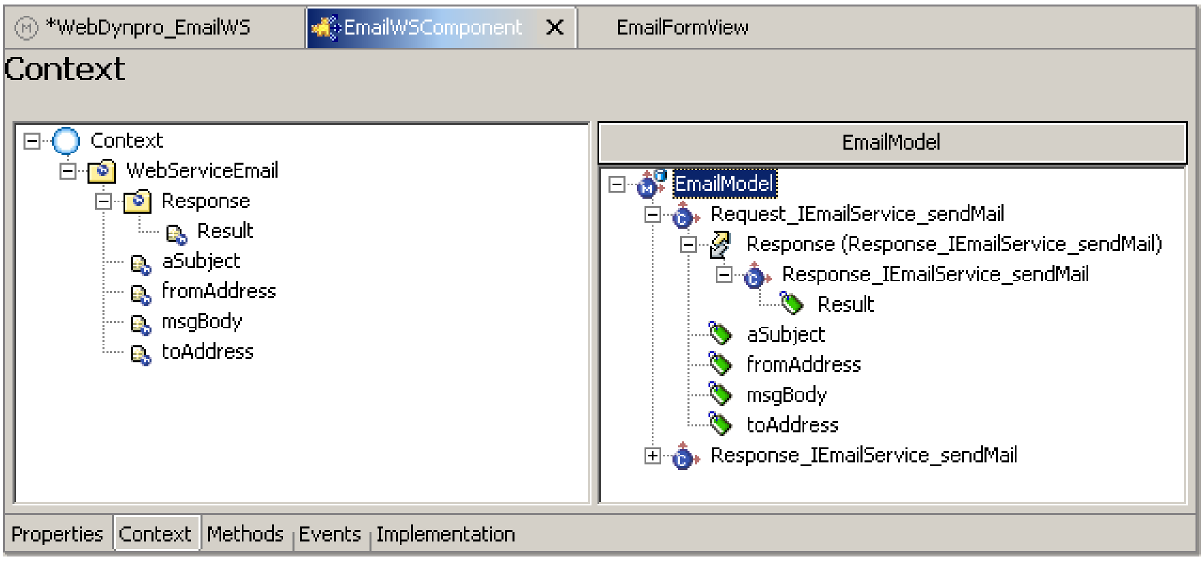


Context mapping

Drag and Drop the source node on to the target node to create the context mapping

9. Close the Model Binding Wizard by choosing *Finish*.

After completing the model binding, the context structure of the component controller looks like this:



Result

Starting with the model definition, you have now defined a structure of context model elements (model nodes and model attributes) in the component controller **`EmailWSComponent`**, and then linked it to the corresponding model class.

Next step:

Mapping View Context Elements to Component Context Elements [Seite 17]

Mapping View Context Elements to 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*. You will learn how to map context elements of a view onto the appropriate context elements of the component controller. The context elements of the view context reference only the data stored in the component context. This data is a copy of the actual model data.

Prerequisites

- The structure of your project **`WebDynpro_EmailWS`** is currently displayed in the *Web Dynpro Explorer*.

Procedure

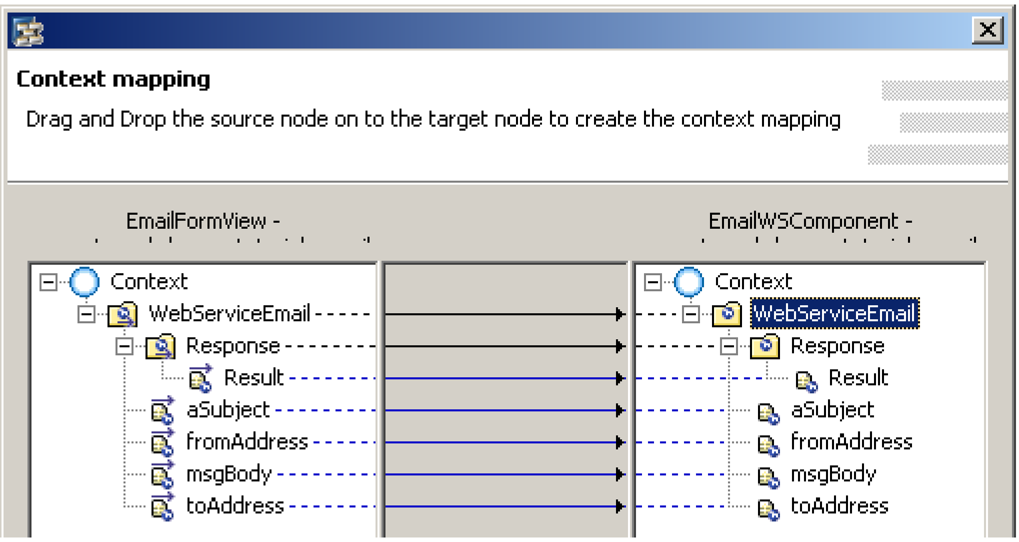
To declare the context mapping between the view controller and the component controller you again use the *Data Modeler*.

Defining a Context Mapping in the Data Modeler

The Web Dynpro tools enable you to map a context node defined in the context of the view *EmailFormView* to a node in the context of the component controller. The context elements still missing in the node of the view context are automatically added and mapped to their respective match in the component context. In addition, the *Controller Usage* of the component controller needed for the context mapping is declared in the view controller.

> The context elements mapped onto one another do not have to have the same name.

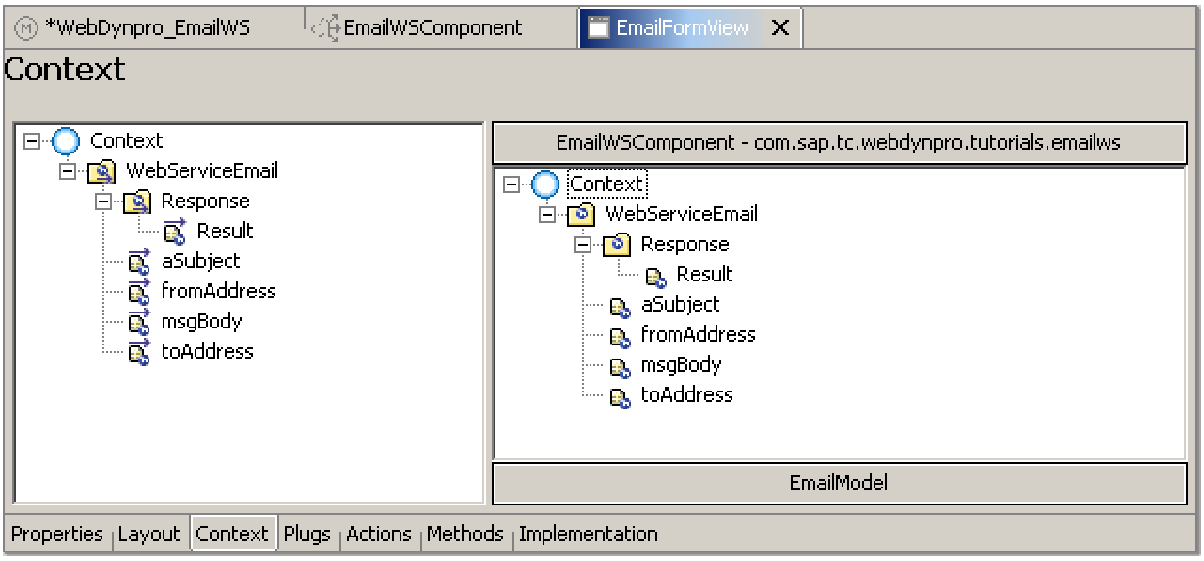
1. Open the *Data Modeler* for the Web Dynpro component *EmailWSComponent*.
2. In the left toolbar, choose ↗ *Create a data link.*
3. Starting above the *Component Controller* rectangle, press the left mouse button, and keep it pressed.
4. Draw a line to the *Component Controller* rectangle and release the left mouse button.
5. Drag the model node *WebServiceEmail* in the context of the component controller to the root node of the view controller context, and drop it.
6. In the dialog box that appears, select the model node **`WebServiceEmail`** and choose *Next.*
7. In the final dialog box, the context mapping declared between the two model nodes **`WebServiceEmail`** is displayed graphically:



Context mapping

Drag and Drop the source node on to the target node to create the context mapping

Afterwards, the model node – together with the further context elements – is mapped to the original context in the component controller.



Result

You have created the necessary view context and mapped it to the component context you created previously. You are now in a position to bind UI controls such as input fields to the corresponding context elements.

Next step:

Creating the SendEmail Action [Seite 19]

Creating the SendEmail Action

To trigger sending the email message from the **EmailFormView** view using the Web service, you need an associated *Action*. Events belonging to UI elements on the client side – such as the **onAction** event of a button – can then be bound to this action. If a corresponding event handler is assigned to this action, you can react to the triggering event (clicking a button) of this action in the user interface. You do this by implementing this event handler in the view controller. In this way, actions represent a link between events on the client side and event handlers on the server side.

Prerequisites

- The structure of your project **WebDynpro_EmailWS** is currently displayed in the *Web Dynpro Explorer*.

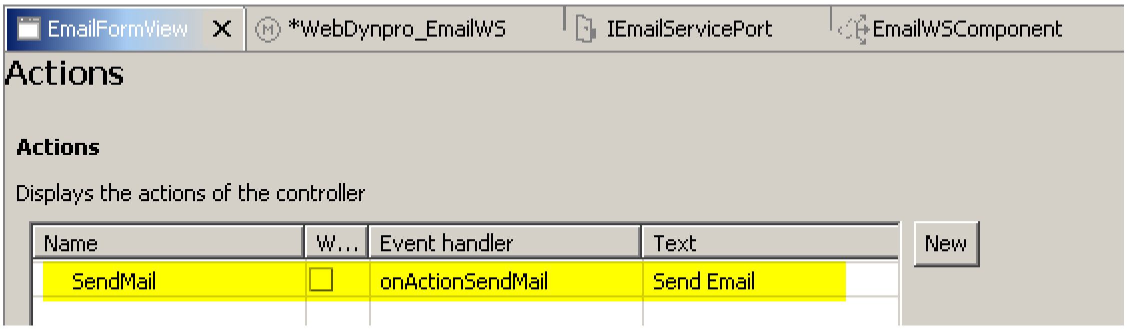
Procedure

Creating the *Go* Action

1. Open the *View Designer* for the **EmailFormView** view.
2. Choose the *Actions* tab.
3. Choose the *New* pushbutton to start the dialog box for defining a new *action*.
4. Enter the name **SendEmail** for the new action.
5. Enter **Send Email** in the *Text* field, then choose *Finish*.

At runtime, the entry is automatically displayed in the field *Text* on the button (UI element) after its *onAction* event is bound to the *SendEmail* action.

After you have defined the new action named **SendEmail**, a corresponding event handler named **onActionSendEmail()** is automatically created in the view controller class **EmailFormView.java**. The actions defined for a view are listed in the View Designer on the *Actions* tab.

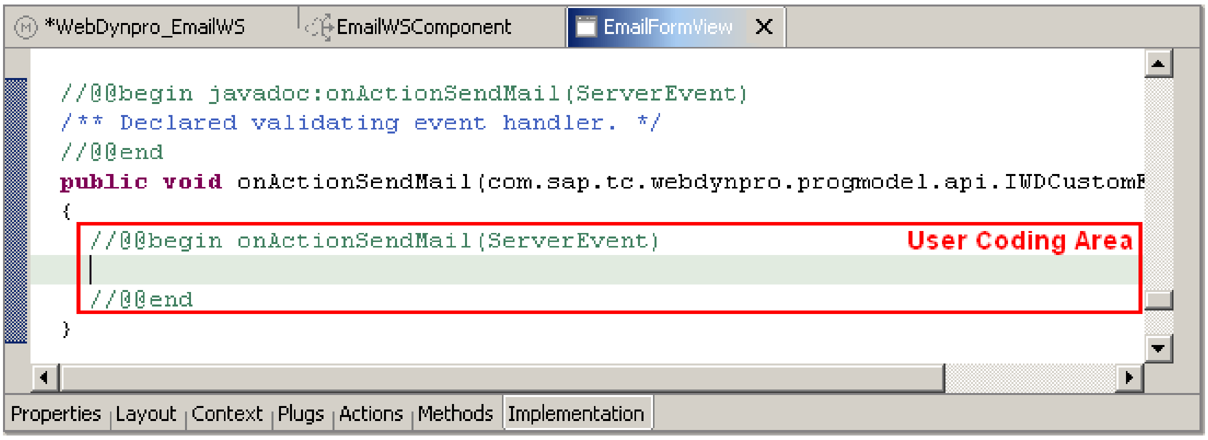


Actions

Displays the actions of the controller

Name	W...	Event handler	Text
SendMail	□	onActionSendMail	Send Email

6. Then switch to the *Implementation* tab. In this way, the Java classes belonging to the view controller are regenerated. The *User Coding Areas* that are accessible to you as the application developer are actually implemented in the class **EmailForm.java**, which has now been extended to include the event handler **onActionSendEmail**.



```
//@@begin javadoc:onActionSendMail(ServerEvent)
/** Declared validating event handler. */
//@@end
public void onActionSendMail(com.sap.tc.webdynpro.progmodel.api.IWDCustomE
{
  //@@begin onActionSendMail(ServerEvent)
  |
  //@@end
}
```

Result

In this step, you created the **`SendEmail`** action for the **`EMailFormView`** view. You can now bind this to the corresponding button as an attribute of the **`onAction`** event, which you will do in the next step. At runtime, the **`onActionSendEMail`** event handler is then called after you have clicked the button for sending the e-mail message in the view controller.

Next step:

Designing a View Layout [Seite 21]

Designing a View Layout

In this step, you will design the user interface that is used for input and for sending the e-mail message. You enhance the view layout of the **`EmailFormView`** view to include the relevant user interface elements, such as labels and input fields. Then you perform the *data binding* between the UI element properties and the model attributes. Finally, you bind the **`SendEmail`** action to the button for sending the e-mail.

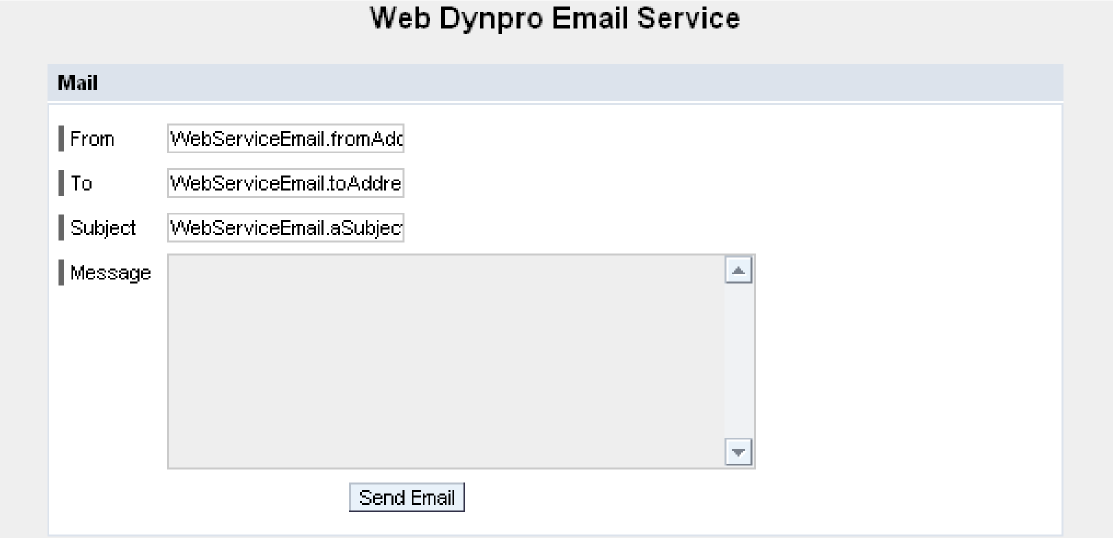
Prerequisites

- You have created the **`SendEmail`** action for the **`EmailFormView`** view.
- The structure of your project **`WebDynpro_EmailWS`** is currently displayed in the *Web Dynpro Explorer*.

Procedure

Creating a layout for the `EmailFormView` view

The following describes how to design the layout of the **`EmailForm`** view in the View Designer of the Web Dynpro tools. Afterwards, the view will look like this:



Web Dynpro Email Service

Mail

From WebServiceEmail.fromAdc

To WebServiceEmail.toAddre

Subject WebServiceEmail.aSubjec

Message

1. Open the **`EmailFormView`** view in the View Designer by clicking the *Layout* tab.
2. The View Designer displays a predefined default text. Simultaneously, the *Outline* view displays a list of the UI elements included. If you select an element in the *Outline* view or on the *Layout* tab, its associated element properties are shown in the *Properties* view.
3. First adapt the individual properties of the two already existing UI elements, as required.

Property	**Value**
For the root element **`RootUIElementContainer`**	
Properties of TransparentContainer – layout	*GridLayout*
For the *TextView* element named **`DefaultTextView`**	
Properties of TextView – design	*header1*
Properties of TextView – text	Web Dynpro e-mail service
LayoutData – hAlign	*center*
LayoutData – paddingBottom	*large*

4. Add a *Group* UI element named **`EmailFormGroup`**. To do this, go to the context menu and choose the entry *Insert Child* above the root element **`RootUIElementContainer`** of the *Outline*. Make the following settings:

Property	**Value**
For the *Group* UI element **`EmailFormGroup`**	
Properties of Group – design	*sapcolor*
Properties of Group – layout	*MatrixLayout*
Properties of Group – scrollingMode	*none*
Properties of Group – width	70%
LayoutData – hAlign	*center*
For the *Caption* UI element **`EmailFormGroup_Header`**	
Properties of Caption – text	Mail

5. The email input form consists of several pairs of *LabelField* and *InputField* UI elements as well as a button for sending the email message. Add further UI elements, one after another, to the *Group* UI element named **`EmailFormGroup`**:

Property	Value
Label **FromLabel** of type *Label*	
Properties of Label – text	From:
Input field **FromInput** of type *InputField*	
Label **ToLabel** of type *Label*	
Properties of Label – text	To:
Properties of Label – layoutdata	***MatrixHeadData***
Input field **ToInput** of type *InputField*	
Label **SubjectLabel** of type *Label*	
Properties of Label – text	Subject:
Properties of Label – layoutdata	***MatrixHeadData***
Input field **SubjectInput** of type *InputField*	
Label **MessageLabel** of type *Label*	
Properties of Label – text	Message:
Properties of Label – layoutdata	***MatrixHeadData***
Properties of Label – valign	***top***
Text editor **MessageText** of type *TextEdit*	
Properties of TextEdit – cols	60
Properties of TextEdit – rows	8
Button **SendButton** of type *Button*	
LayoutData – colSpan	2
Properties of Label – layoutdata	***MatrixHeadData***
LayoutData – hAlign	*center*

Defining a Data Binding to the View Context

The connection from the user interface elements to the business data referenced in the view controller context can now be set up easily using *Data Binding*.

To do this, make the following data binding connections between the UI element properties and the corresponding model attributes.

You can start the wizard for data binding of a UI element property to a context attribute by choosing the ... button at the right margin of the *Value* column in the *Properties* view.

Property	Value
Input field **`FromInput`** of type *InputField*	
Properties of InputField – value	✓ WebServiceEmail.fromAddress
Input field **`ToInput`** of type *InputField*	
Properties of InputField – value	✓ WebServiceEmail.toAddress
Input field **`SubjectInput`** of type *InputField*	
Properties of InputField – value	✓ WebServiceEmail.aSubject
Text editor **`MessageText`** of type *TextEdit*	
Properties of TextEdit – value	✓ WebServiceEmail.msgBody

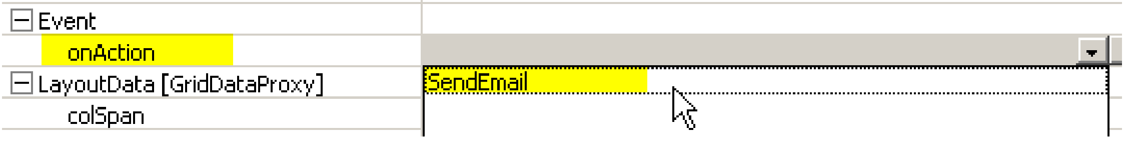
Binding an action to the `onAction` event of the UI element `SendButton`

So that the email message (entered on the client side) can be sent to the Web service (on the server side) using the view controller, a server round trip must be triggered by the **`SendEmail`** action. To do this, you bind the **`onAction`** event of the *Button* UI element **`SendButton`** to the action you have created, **`SendEmail`**.

To do this, create the following binding:

Property	Value
For the **`SendButton`** pushbutton of type *Button*	
Event – onAction	*SendEmail*

You bind the action to a user interface element event using the selection list in the right column in the *Properties* view.



⊟ Event	
onAction	SendEmail
⊟ LayoutData [GridDataProxy]	
colSpan	

Next step:

Adding the Implementation for the Web Service Connection [Seite 25]

Adding the Implementation for the Web Service Connection

Prerequisites

- You have imported a model that is based on the e-mail Web service of a third party provider.
- You have created the **`SendEmail`** action for the **`EmailFormView`** view.
- You have constructed the view context for the **`EmailForm`** view and mapped it to the component controller context.
- The structure of your project **`WebDynpro_EmailWS`** is currently displayed in the *Web Dynpro Explorer*.

Procedure

After you have executed all the declarative development steps, such as model binding, context mapping, and data binding, and before you determine the view layout, you now need to add individual lines of Java code in the source code of the view controller.

The data binding between UI elements and the view context, which itself is bound to the model, is based - at runtime - on the fact that there is a reference in a UI element property to the context attribute of an actual node element. In this current example, a node element of the same type as the appropriate, generated model class must be stored in the model node **`WebServiceEmail`**. That is, this kind of model node element must be *bound* to the model node **`WebServiceEmail`**. This binding takes place in the **`wdDoInit()`** method.

The actual Web service call must be implemented in the action event handler **`onActionSendEmail()`**, based on the data entered by the user that is stored in the context. In addition, the results returned by the Web service are to be stored in the context.

Implementing the Generic Event Handler `wdDoInit()` of the View Controller

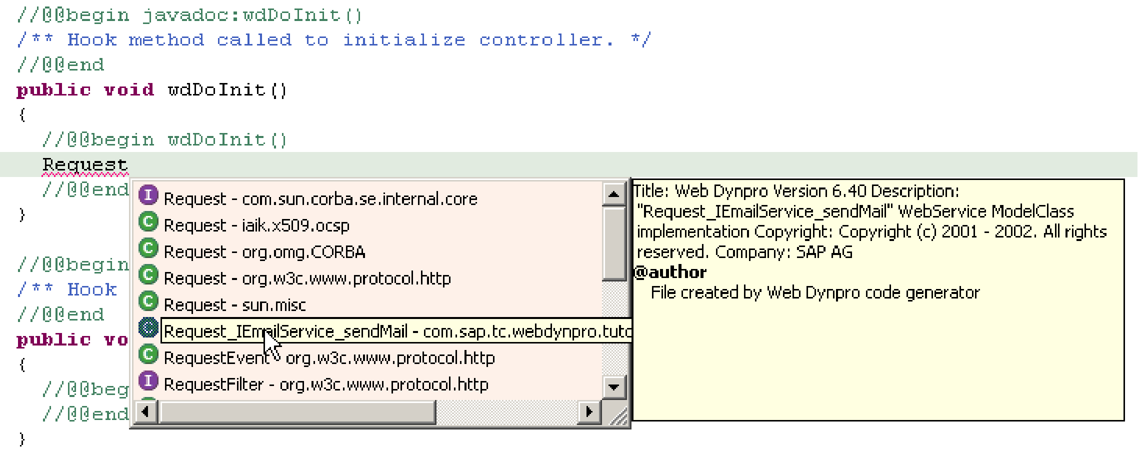
At runtime, the context model node **`WebServiceEmail`** that is bound to the model must initially be filled with an instance of the appropriate model adapter class. This model object, in turn, passes its data to the suitable Java proxy, which then communicates with the actual Web service.

1. In the View Designer, click on the *Implementation* tab for the **`EmailFormView`** view.
2. After the generation routines have been run once again, the updated source code of the view controller implementation is displayed.
3. Now add the following Java code into the *User Coding Area* provided.

```
//@@begin javadoc:wdDoInit()
/** Hook method called to initialize controller. */
//@@end
public void wdDoInit()
{
  //@@begin wdDoInit()
  // create a new instance of the Web Service ModelClass
  Request_IEmailService_sendMail req = new Request_IEmailService_sendMail();

  // bind new instance of the Web Service ModelClass to the
  // independent Model Node ’WebServiceEmail’
  wdContext.nodeWebServiceEmail().bind(req);
  //@@end
}
```

To enter the source code, you can use the *Code Assist* functions provided by the SAP NetWeaver Developer Studio by selecting the keyboard combination CTRL+SPACE.



```
//@@begin javadoc:wdDoInit()
/** Hook method called to initialize controller. */
//@@end
public void wdDoInit()
{
  //@@begin wdDoInit()
  Request
  //@@end
}

//@@begin
/** Hook
//@@end
public vo
{
  //@@beg
  //@@end
}
```

Request - com.sun.corba.se.internal.core
Request - iaik.x509.ocsp
Request - org.omg.CORBA
Request - org.w3c.www.protocol.http
Request - sun.misc
Request_IEmailService_sendMail - com.sap.tc.webdynpro.tuto
RequestEvent - org.w3c.www.protocol.http
RequestFilter - org.w3c.www.protocol.http

Title: Web Dynpro Version 6.40 Description: "Request_IEmailService_sendMail" WebService ModelClass implementation Copyright: Copyright (c) 2001 - 2002. All rights reserved. Company: SAP AG
@author
File created by Web Dynpro code generator

4. You can add the missing `import` statement by choosing *Source → Organize Imports* from the context menu:

```
...//@@begin imports
import com.sap.tc.webdynpro.tutorial.emailws
          .model.proxies.Request_IEmailService_sendMail;
import com.sap.tc.webdynpro.tutorials.emailws.wdp.IPrivateEmailFormView;
//@@end
```

Implementing the action event handler `onActionSendEmail`

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`** (this is contained in the context as an inner node underneath the node **`WebServiceEmail`**). 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.

1. In the **`onActionSendEmail()`** method, add the following source code:

```
//@@begin javadoc:onActionSendMail(ServerEvent)
/** Declared validating event handler. */
//@@end
public void
onActionSendEmail(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
{
  //@@begin onActionSendEmail(ServerEvent)
  IWDMessageManager msgMgr=
    wdThis.wdGetAPI().getComponent().getMessageManager();

  try {
    // call Email Web Service and update dependent model node 'Response'
    wdContext.currentWebServiceEmailElement().modelObject().execute();
    wdContext.nodeResponse().invalidate();
    int result = wdContext.currentResponseElement().getResult();
    String msg = "Email Web Service returned " + Integer.toString(result);

    if (result == 0) {
      msgMgr.reportSuccess("Your email was successfully sent ("
        + msg + ")!");
    } else {
      msgMgr.reportWarning("Your email was not successfully sent ("
        + msg + ")!");
    }
  } catch(Exception ex) {
    msgMgr.reportException(ex.getLocalizedMessage(),true);
  }
  //@@end
}
```

To use the generic UI service provided by the interface **`IWDMessageManager`** for displaying message texts in the user interface, you must insert an appropriate import line in the view controller implementation. To do this, choose the entry *Source* → *Organize Imports* from the context menu of the source code editor (*Implementation* tab).

After the `import` statements have been adjusted, the **`IWDMessageManager`** interface is imported into the view controller.

```
...
//@@begin imports
import com.sap.tc.webdynpro.progmodel.api.IWDMessageManager;
import com.sap.tc.webdynpro.tutorials.emailws.model.proxies
              .Request_IEmailService_sendMail;
import com.sap.tc.webdynpro.tutorials.emailws.wdp.IPrivateEmailForm;
//@@end
}
...
```

Result

The Developer Studio updates and compiles the Java classes belonging to your project. (Note: Compilation only occurs if you are using the default Workbench settings.) After you have done this, no more error messages should appear in your *Tasks* view.

Next step:

Building, Deploying, and Running Your Application [Seite 28]

Building, Deploying, and Running Your Application

Now that you have reached this stage, you can start the fully developed example application in the Web Browser as described below.

However, some preparations are essential before you can deploy and run the application successfully on the SAP J2EE Engine. Go through each of the following prerequisites carefully.

Prerequisites

- The external e-mail Web service used in the example application is available.
- You have made sure that the SAP J2EE Engine has been launched and that you are connected to an appropriate database instance of the SAP DB.

  To do this, refer to Starting and Stopping the SAP J2EE Engine [Extern]
- You have checked that the configuration settings for the J2EE server are entered correctly in the Developer Studio.

  To check the server settings, choose the menu path *Window → Preferences → SAP J2EE Engine.*

Procedure

Building the Project

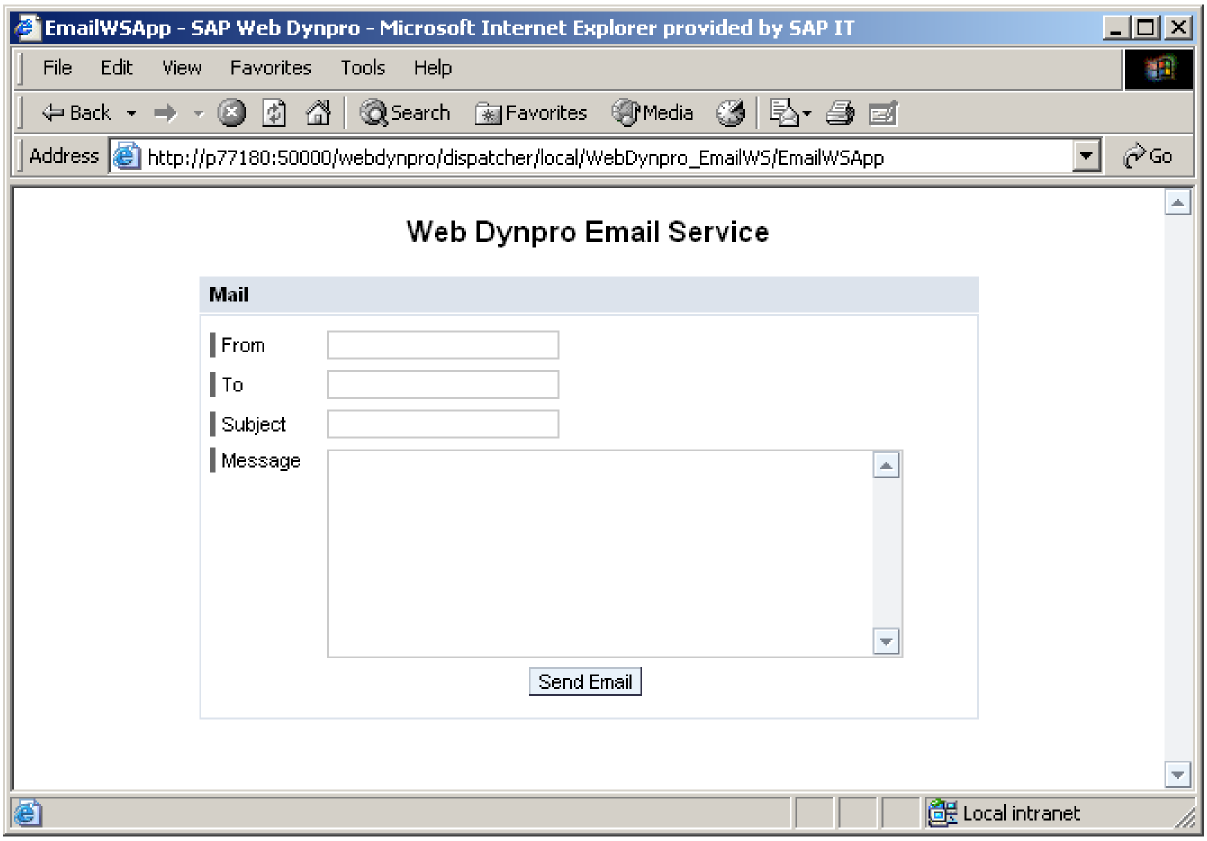
1. Save the current status of the metadata for the project using the button in the upper application toolbar of your Developer Studio, if you have not already done so.
2. Open the context menu for the project node (**`WebDynpro_EmailWS`**) in the Web Dynpro Explorer and choose *Rebuild Project.* Make sure that the *Tasks* view does not display any errors for your project. You can ignore any warning messages for *labelFor* properties that have not been set.

Deploying and Launching the Application

1. In the Web Dynpro Explorer, open the context menu for the application object **`EmailWSApp`**.
2. Choose *Deploy new archive and run*.

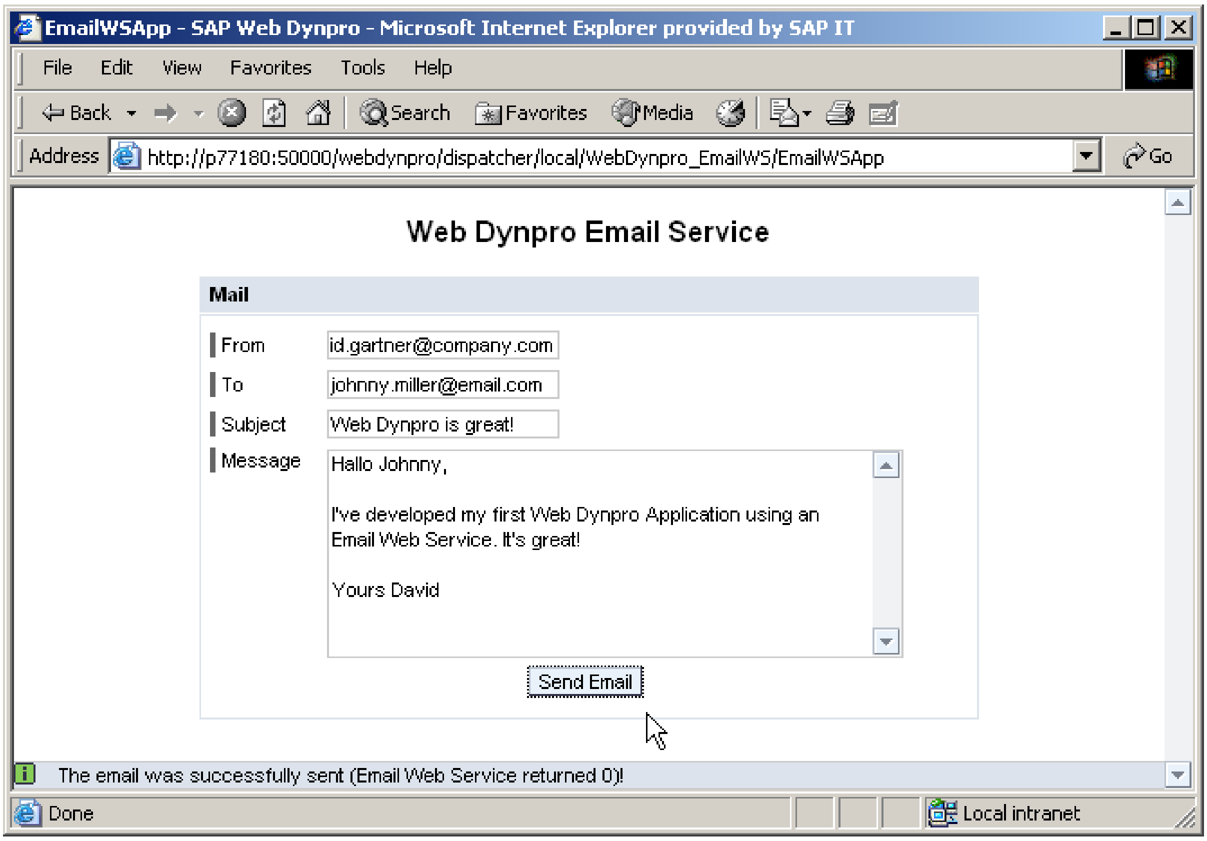
Result

The Developer Studio deploys the application in one single step, based on an automatically generated Enterprise Archive File, and then automatically launches your application in the Web browser.


EmailWSApp - SAP Web Dynpro - Microsoft Internet Explorer provided by SAP IT
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://p77180:50000/webdynpro/dispatcher/local/WebDynpro_EmailWS/EmailWSApp
Go
Web Dynpro Email Service
Mail
From
To
Subject
Message
Send Email
Local intranet

Test your new Web Dynpro application by entering your own e-mail data in the input fields *From, To, Subject*, and *Message*, and then click on *Send*.

After you have triggered a server roundtrip (here you communicate with the *Email* Web service), a success message is displayed on the user interface in the Web Browser if no errors have occurred during sending. The result value returned by the e-mail Web service is also displayed in the message text.


EmailWSApp - SAP Web Dynpro - Microsoft Internet Explorer provided by SAP IT
File
Edit
View
Favorites
Tools
Help
Back
Search
Favorites
Media
Address
http://p77180:50000/webdynpro/dispatcher/local/WebDynpro_EmailWS/EmailWSApp
Go
Done
Local intranet

Web Dynpro Email Service

Mail

From: id.gartner@company.com

To: johnny.miller@email.com

Subject: Web Dynpro is great!

Message:

Hallo Johnny,

I've developed my first Web Dynpro Application using an Email Web Service. It's great!

Yours David

Send Email

The email was successfully sent (Email Web Service returned 0)!