

2011

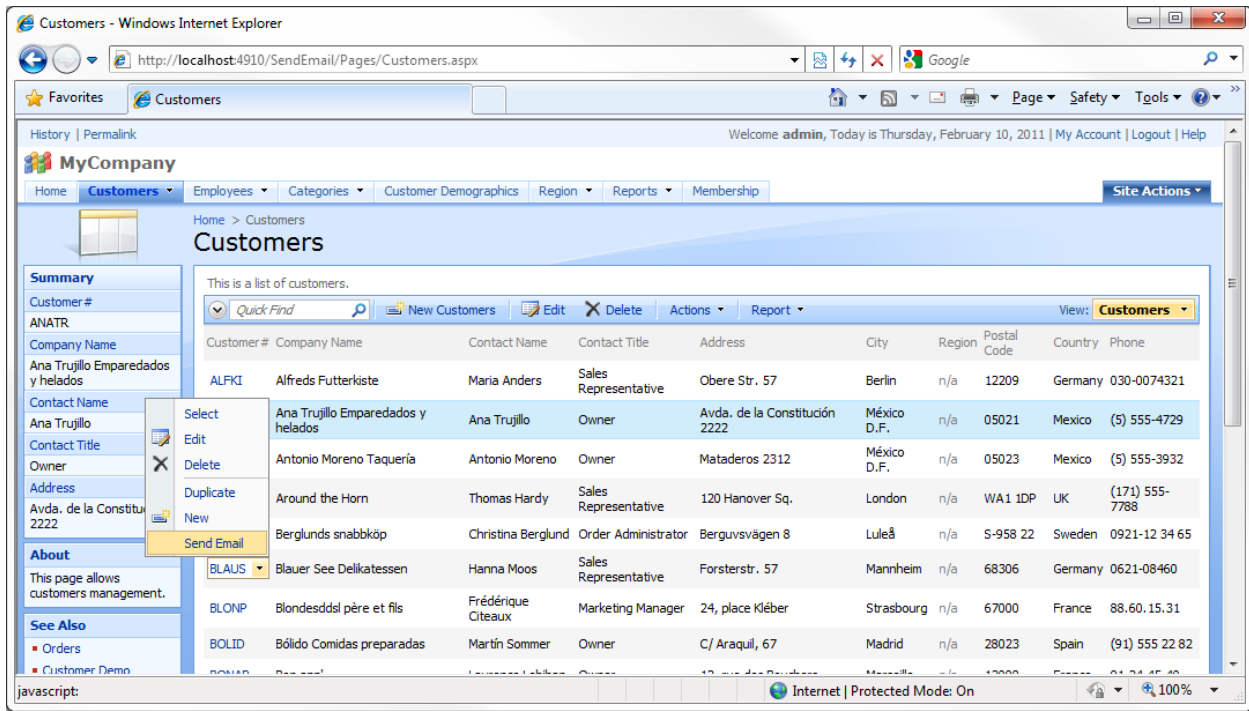


# **COOKBOOK**

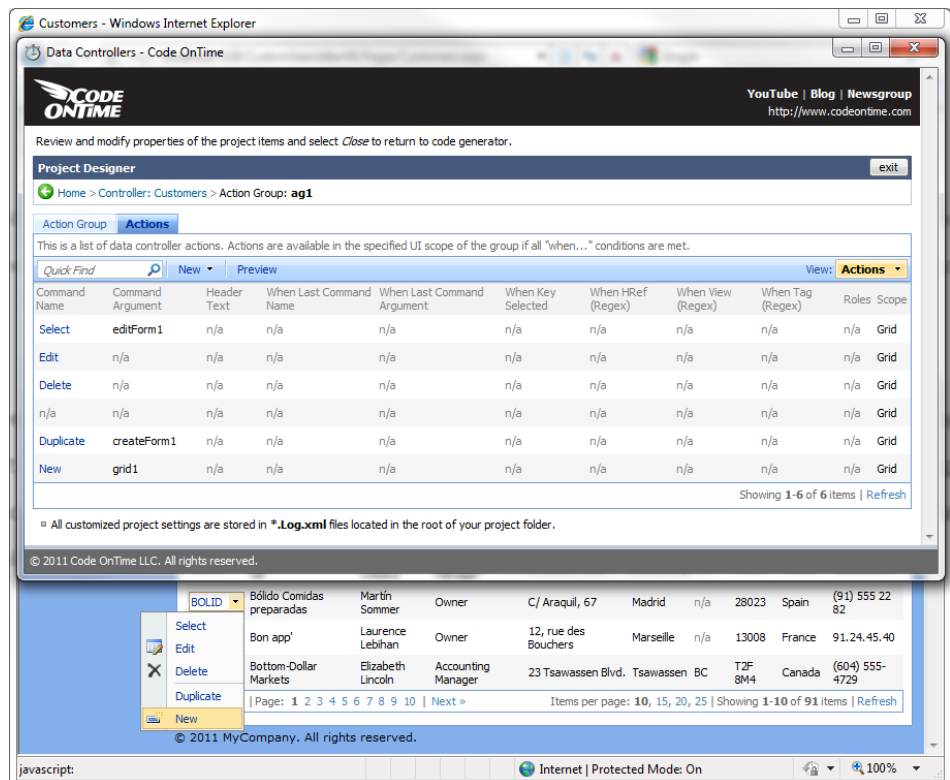
Sending an Email in Response to Actions

## Send an Email

Let's expand the capabilities of the context menu in Customers grid view. We will add a new option, seen below, which will execute a custom business rule and send an email.



Run *Code On Time Generator*, select the name of the project you wish to edit, and press *Design*. In the list of controllers, select *Customers*. Switch to the *Action Groups* tab, and select *ag1* with scope of *Grid*. This is the group of actions that reside in the grid view context menu. Switch to the *Actions* tab, and you will see the actions that are available on the context menu.



On the action bar, press *New/ New Action*. The *Command Name* will be “Custom”, and the *Command Argument* will be “SendEmail”, with *Header Text* of “Send Email”.

**Project Designer** exit

Home > Controller: Customers > Action Group: **ag1**

Action Group **Actions**

Please fill this form and click OK button to create a new action record. Click Cancel to return to the previous screen.

View: **New Action** ▾

\* - indicates a required field OK Cancel

<p><b>General</b></p> <p>Key action properties.</p> <p>If command name is not specified then the action is presented as a break line in the action group popup menu.</p>	<p>Command Name Custom ▾</p> <p>Command Argument SendEmail</p> <p>Header Text Send Email</p> <p>Causes Validation * Yes ▾</p>
<p><b>Conditions</b></p>	<p>When Last Command Name</p>

Now, go back to the settings for the *Customers* controller. Edit, and in the *Handler* field, insert “CustomersBusinessRules”.

**Project Designer** exit

Home > Controller: **Customers**

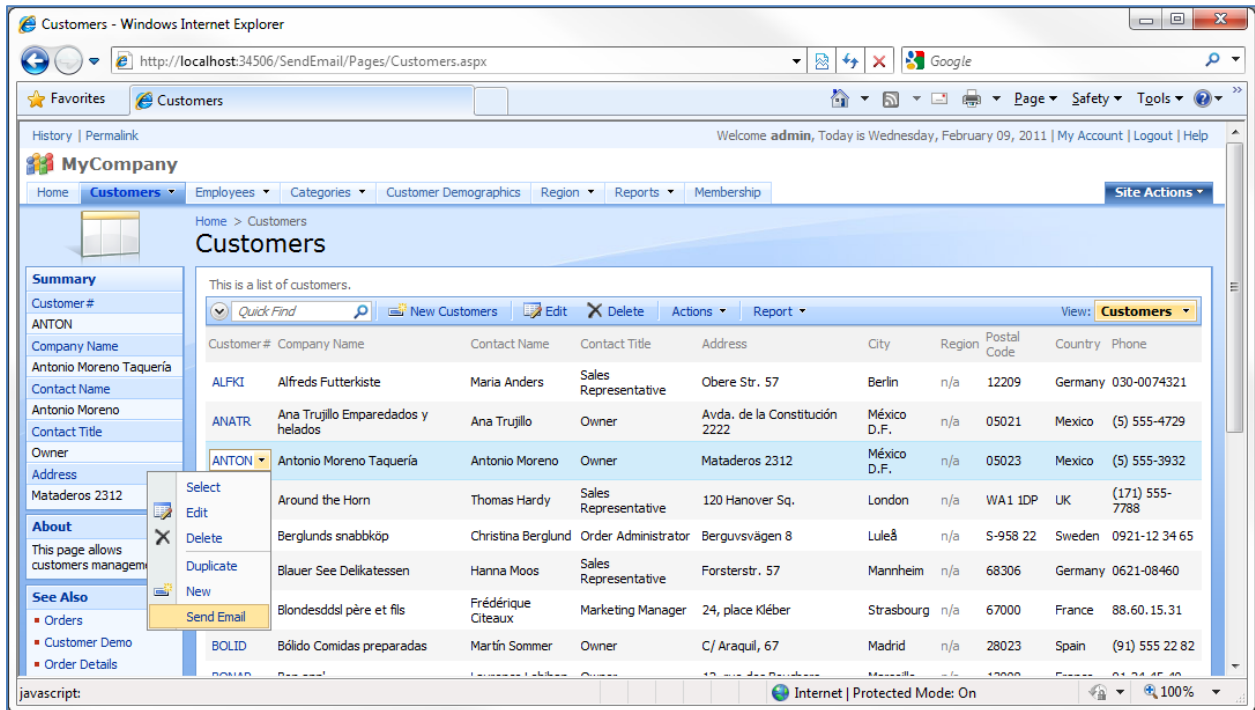
Controller Commands Fields Views Categories Data Fields Action Groups Actions

Please review data controller information below. Click Edit to change this record, click Delete to delete the record, or click Cancel/Close to return back.

Record ▾ View: **Controller** ▾

<p><b>General</b></p> <p>Name of data controller.</p>	<p>Controller Name * Customers</p> <p>Include in code generation * Yes ▾</p>
<p><b>Miscellaneous</b></p> <p>Specify conflict detection strategy and optional connection string name. Specify a connection string name only if the controller is working with the database other than the one selected for this project.</p>	<p>Conflict Detection * <input checked="" type="radio"/> Overwrite Changes <input type="radio"/> Compare All Values</p> <p>Connection String Name _____</p>
<p><b>Business Rules</b></p> <p>Specify the name of the <a href="#">business rules</a> class that can be extended to respond to the controller actions.</p>	<p>Handler CustomersBusinessRules</p>
<p><b>Annotations</b></p> <p>Specify if free form notes and attachments can be assigned any data row by end-users at run-time.</p>	<p>Allow Annotations N/A ▾</p>

Save your changes, exit the *Designer*, and generate the application. When the web page appears, navigate to *Customers*. When you activate the dropdown next to a row, you can see the new “Send Email” option. When pressed, the row will be selected, but otherwise nothing else will happen.



Let's add emailing functionality by opening the project in *Microsoft Visual Studio* or *Visual Web Developer*. Open the file that has been created, under *App\_Code/Rules/CustomersBusinessRules.vb(cs)*. The following code will send an email to the selected customer.

## VB.Net

```
Imports MyCompany.Data
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Linq
Imports System.Net.Mail
Imports System.Net
```

```
Namespace MyCompany.Rules
```

```
    Partial Public Class CustomersBusinessRules
        Inherits MyCompany.Data.BusinessRules
```

```
        <ControllerAction("Customers", "Custom", "SendEmail")> _
        Sub SendAnEmailToACustomer(ByVal companyName As String, ByVal contactName As String, _
            ByVal customerID As String)
            Dim message As MailMessage = New MailMessage()
            message.From = New MailAddress("SENDER@gmail.com", "web App Admin")

            ' prepare smtp client
            Dim smtp As SmtplibClient = New SmtplibClient("smtp.gmail.com", 587)
            smtp.EnableSsl = True
            smtp.UseDefaultCredentials = False
            smtp.Credentials = New NetworkCredential("SENDER@gmail.com", "PASSWORD")
```

```

        ' compose and send an email message
        ' use CustomerID to find the recipient's email
        Dim sendTo As String = "recipient@contoso.com" ' fake email address of a customer

        message.To.Add(sendTo)
        message.Subject = String.Format("Hello {0} at {1}.", contactName, companyName)
        message.Body = "Hello there!"
        smtp.Send(message)

        Result.ShowAlert("Email has been sent.")
    End Sub
End Class
End Namespace

```

## C#

```

using System;
using System.Data;
using System.Collections.Generic;
using System.Linq;
using MyCompany.Data;
using System.Net.Mail;
using System.Net;

namespace MyCompany.Rules
{
    public partial class CustomersBusinessRules : MyCompany.Data.BusinessRules
    {
        [ControllerAction("Customers", "Custom", "SendEmail")]
        public void SendAnEmailToACustomer(string companyName, string contactName, string customerId)
        {
            MailMessage message = new MailMessage();
            message.From = new MailAddress("SENDER@gmail.com", "Web App Admin");

            // prepare smtp client
            SmtplibClient smtp = new SmtplibClient("smtp.gmail.com", 587);
            smtp.EnableSsl = true;
            smtp.UseDefaultCredentials = false;
            smtp.Credentials = new NetworkCredential("SENDER@gmail.com", "PASSWORD");

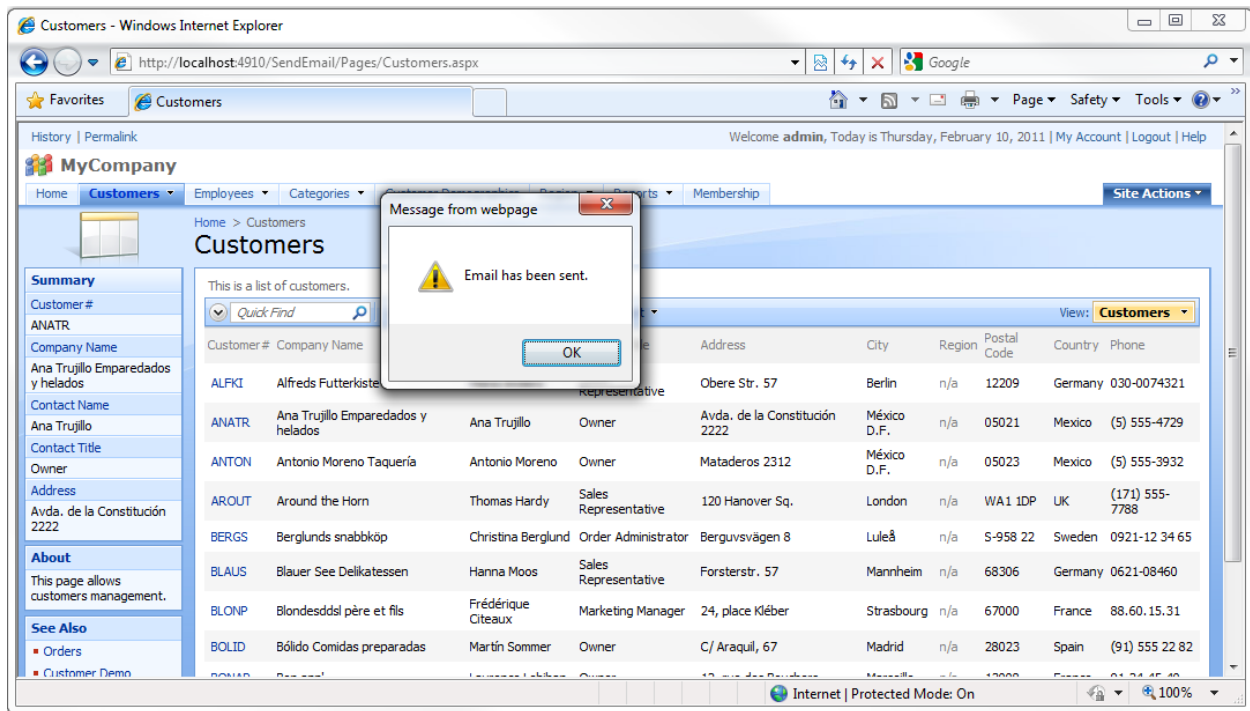
            // compose and send an email message
            // use CustomerID to find the recipient's email
            string sendTo = "recipient@contoso.com"; // fake email address of a customer
            message.To.Add(sendTo);
            message.Subject = String.Format("Hello {0} at {1}.", contactName, companyName);
            message.Body = "Hello there!";
            smtp.Send(message);

            Result.ShowAlert("Email has been sent.");
        }
    }
}

```

The business rules class shows how to handle the action by sending an email from an existing Gmail account to the selected recipient. Notice that the order of parameters in the business rule method is arbitrary. You can list every single field available in client-side view or limit the list of arguments to only those that are really needed to compose an email.

Now, when you select the *Send Email* action in the context menu of the grid view, an email will be sent to the employee email address from the email account specified under *Web App Admin* alias.



Follow the same pattern to implement any other actions that must affect a single record selected by a web application user.

You may explore the possibility of adding custom actions into other action groups.

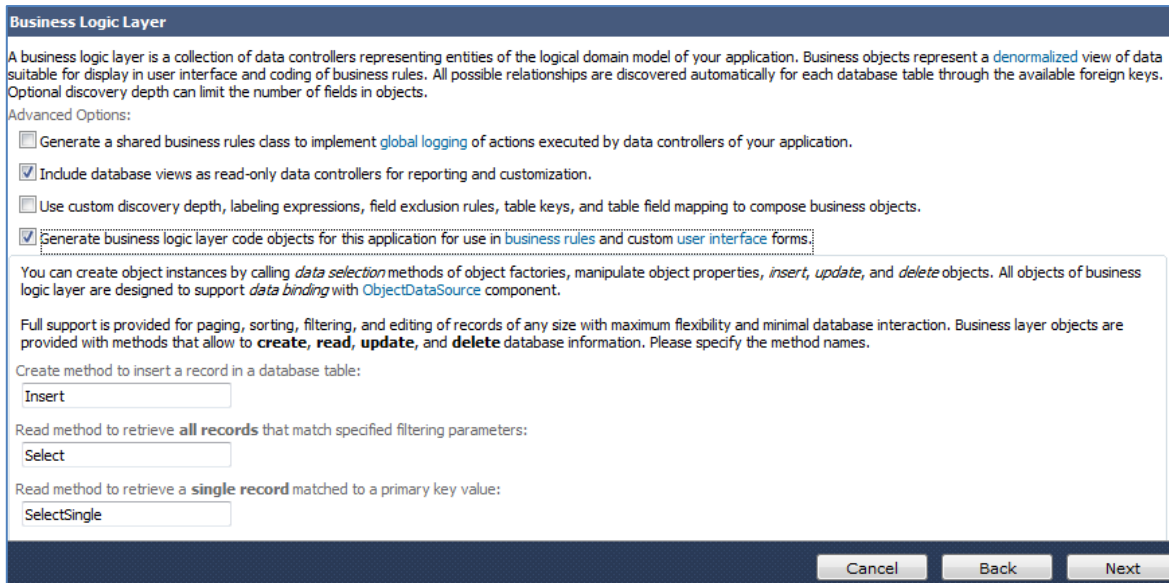
For example, action group with the scope "Form" will present an action as a button.

Action with the scope "Action Bar" will present an action as a menu option on the action bar of a view.

## Send an Email to Multiple Recipients

If you enable selection of multiple records then you can send a customized email to the corresponding recipients.

To do this, open *Code On Time Generator* and click on the project name. Press *Next* until you reach the *Business Logic Layer* page. Activate the checkbox next to “Generate business logic layer”.



The screenshot shows the 'Business Logic Layer' configuration page. It includes a title bar, a descriptive paragraph about business logic layers, and a section for 'Advanced Options' with four checkboxes. The fourth checkbox, 'Generate business logic layer code objects for this application for use in business rules and custom user interface forms', is checked. Below this are three text input fields for defining database methods: 'Insert', 'Select', and 'SelectSingle'. At the bottom right, there are 'Cancel', 'Back', and 'Next' buttons.

**Business Logic Layer**

A business logic layer is a collection of data controllers representing entities of the logical domain model of your application. Business objects represent a [denormalized](#) view of data suitable for display in user interface and coding of business rules. All possible relationships are discovered automatically for each database table through the available foreign keys. Optional discovery depth can limit the number of fields in objects.

Advanced Options:

- Generate a shared business rules class to implement [global logging](#) of actions executed by data controllers of your application.
- Include database views as read-only data controllers for reporting and customization.
- Use custom discovery depth, labeling expressions, field exclusion rules, table keys, and table field mapping to compose business objects.
- Generate business logic layer code objects for this application for use in [business rules](#) and custom [user interface forms](#).

You can create object instances by calling *data selection* methods of object factories, manipulate object properties, *insert*, *update*, and *delete* objects. All objects of business logic layer are designed to support *data binding* with [ObjectDataSource](#) component.

Full support is provided for paging, sorting, filtering, and editing of records of any size with maximum flexibility and minimal database interaction. Business layer objects are provided with methods that allow to **create**, **read**, **update**, and **delete** database information. Please specify the method names.

Create method to insert a record in a database table:

Read method to retrieve **all records** that match specified filtering parameters:

Read method to retrieve a **single record** matched to a primary key value:

Cancel Back Next

Continue pressing *Next* until you reach the *Data Controllers* page, and press “Start Designer”.

From the list of *All Controllers*, select the *Customers* controller.

Switch to the *Action Groups* tab, and select *ag5*. On the action bar, press *New | New Action*.

Leave all the settings blank except for the *Roles* field, which *will* be “Administrators”, and save. This will create a “break” line in the list of actions.

Notice that if the user is not in the role of “Administrators” then the “break” line will not be displayed in the menu.

Project Designer
exit

➤ Home > Controller: Customers > Action Group: ag5

Action Group
Actions

Please fill this form and click OK button to create a new action record. Click Cancel to return to the previous screen.

View: New Action

\* - indicates a required field

<p><b>General</b></p> <p>Key action properties.</p> <p>If command name is not specified then the action is presented as a break line in the action group popup menu.</p>	<p>Command Name  <input type="text" value="N/A"/></p> <p>Command Argument  <input type="text"/></p> <p>Header Text  <input type="text"/></p> <p>Causes Validation *  <input type="text" value="Yes"/></p>
<p><b>Conditions</b></p> <p>Conditions control action availability in the user interface.</p> <p>Conditions <i>When Last Command Name</i> and <i>When Last Command Argument</i> are compared with the last executed command.</p> <p>Condition <i>When Key Selected</i> is true if there is a selected record (row).</p> <p>Conditions <i>When HRef</i>, <i>When View</i>, and <i>When Tag</i> are <b>regular expressions</b> that must positively match the browser window HRef (current page URL), the view ID (grid1, editForm1, etc.), or the data view extender tag accordingly. If you would like to have a negative match then put "false:" without double quotes in front of the expression.</p> <p>The data view extender tag can be specified on the data view of the page in Designer. Return to the designer home and select <i>All Pages</i>, click on the page and choose <i>Data Views</i> tab.</p>	<p>When Last Command Name  <input type="text" value="N/A"/></p> <p>When Last Command Argument  <input type="text"/></p> <p>When Key Selected  <input type="text" value="N/A"/></p> <p>When HRef (Regex)  <input type="text"/></p> <p>When View (Regex)  <input type="text"/></p> <p>When Tag (Regex)  <input type="text"/></p> <p>When Client Script (JavaScript)  <input type="text"/></p>
<p><b>Presentation</b></p> <p>Specify action presentation properties.</p>	<p>Description  <input type="text"/></p> <p>Confirmation  <input type="text"/></p> <p>Css Class  <input type="text"/></p>
<p><b>Security</b></p> <p>List roles allowed to execute this action. Role check is performed on the server and precedes the action conditions.</p>	<p>Roles  <input type="text" value="Administrators"/></p>



Create another action. This action will have a *Command Name* of “Custom”, *Command Argument* of “NotifySelectedCustomers”, and *Header Text* of “Notify”.

**Project Designer** [exit]

Home > Controller: Customers > Action Group: ag5

Action Group **Actions**

Please fill this form and click OK button to create a new action record. Click Cancel to return to the previous screen.

View: **New Action**

\* - indicates a required field

**General**  
Key action properties.  
If command name is not specified then the action is presented as a break line in the action group popup menu.

Command Name: Custom  
Command Argument: NotifySelectedCustomers  
Header Text: Notify  
Causes Validation \*: Yes

**Conditions**  
Conditions control action availability in the user interface.  
Conditions *When Last Command Name* and *When Last Command Argument* are compared with the last executed command.

When Last Command Name: N/A  
When Last Command Argument:

OK Cancel

*Description* will be “Send notification to selected Customers”, *Confirmation* will be “Notify?”, and *Roles* will be “Administrators” to ensure that only administrators can initiate this action.

**Presentation**  
Specify action presentation properties.

Description: Send notification to selected Customers.  
Confirmation: Notify?  
Css Class:

**Security**  
List roles allowed to execute this action. Role check is performed on the server and precedes the action conditions.

Roles: Administrators

\* - indicates a required field

OK Cancel

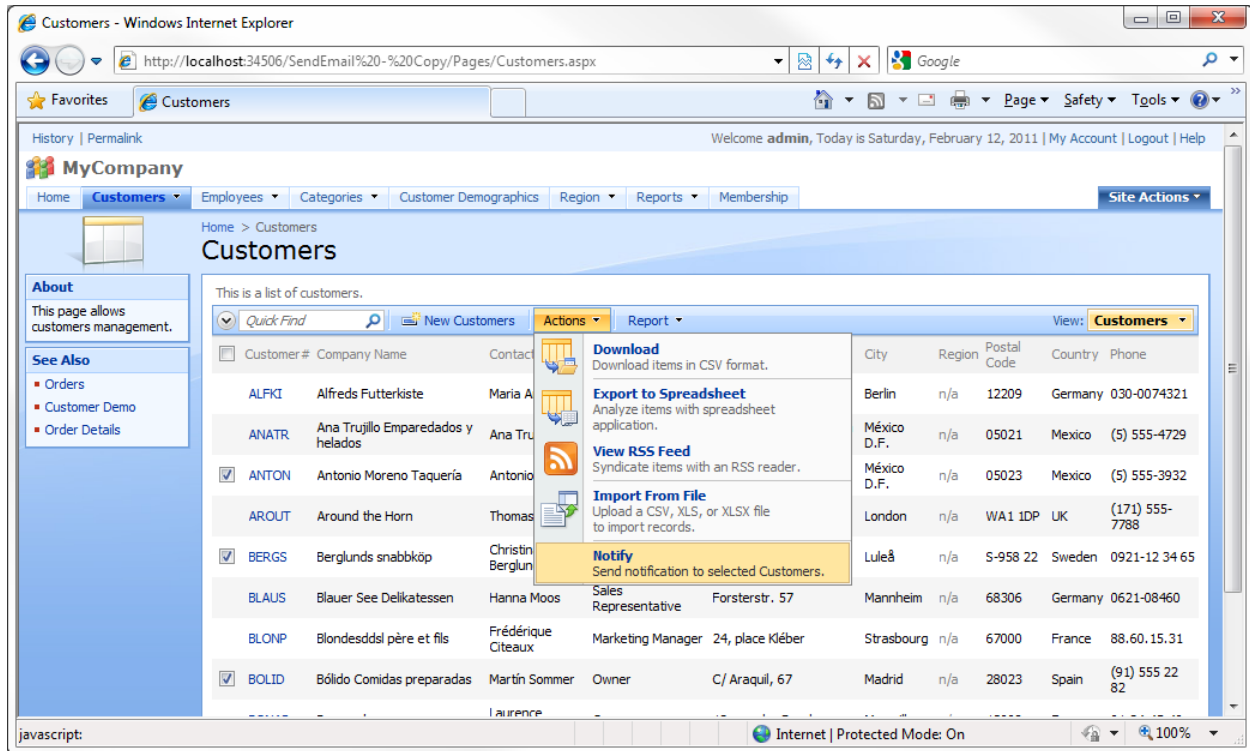
Save the action, and navigate up to the list of *All Pages*. Select the *Customers* page, and switch to the *Data Views* tab. Click on *view1*, and change *Selection Mode* to “Multiple”.

**Presentation**  
Presentation properties of the data view.

Show In Summary  
Page Size:   
Selection Mode \*: Multiple

Save the view, exit the *Designer*, and generate the application.

When the web page appears, navigate to *Customers* page. You will see that you can select multiple customers with a checkbox in the first column. There is also a “Notify” button under the *Actions* menu. Currently, pressing this button will not do anything except prompt a “Notify?” confirmation screen.



You will need to make modifications to *CustomersBusinessRules.vb(cs)* to implement the required business logic that will execute in response to the action. The code will find the email addresses of the selected customers, and send a notification. The updated business rules class can be found below.

Notice that method *SendAnEmailToSelectedCustomers* does not require any parameters. We are not interested in the current selected record and instead are paying attention to *Arguments.SelectedValues* array of selected primary keys.

The name of the method is irrelevant.

The arguments of the *ControllerAction* attribute ensure that the method is executed in response to user actions.

We are using *Customers* business object to locate customer information while iterating through the primary keys of the selected customers.

## VB.NET

```
Imports MyCompany.Data
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Linq
Imports System.Net.Mail
Imports System.Net
Imports MyCompany.Data.Objects

Namespace MyCompany.Rules

    Partial Public Class CustomersBusinessRules
        Inherits MyCompany.Data.BusinessRules

        <ControllerAction("Customers", "Custom", "NotifySelectedCustomers")> _
        Sub SendAnEmailToSelectedCustomers()
            Dim message As MailMessage = New MailMessage()
            message.From = New MailAddress("SENDER@gmail.com", "web App Admin")

            ' prepare smtp client
            Dim smtp As SmtplibClient = New SmtplibClient("smtp.gmail.com", 587)
            smtp.EnableSsl = True
            smtp.UseDefaultCredentials = False
            smtp.Credentials = New NetworkCredential("SENDER@gmail.com", "PASSWORD")

            ' compose and send an email message to selected customers
            For Each customerId As String In Arguments.SelectedValues

                ' use customerId to find the recipient's email

                Dim sendTo As String = "recipient@contoso.com" ' fake email address of a customer
                Dim customer As Customers = Customers.SelectSingle(customerId)

                message.To.Add(sendTo)
                message.Subject = String.Format("Hello {0} at {1}.", customer.ContactName, _
                    customer.CompanyName)
                message.Body = "Hello there!"
                smtp.Send(message)

            Next

            Result.ShowMessage(String.Format("{0} notifications have been sent.", _
                Arguments.SelectedValues.Length))

        End Sub

        <ControllerAction("Customers", "Custom", "SendEmail")> _
        Sub SendAnEmailToACustomer(ByVal companyName As String, ByVal contactName As String, _
            ByVal customerId As String)
            Dim message As MailMessage = New MailMessage()
            message.From = New MailAddress("SENDER@gmail.com", "Web App Admin")
            ' prepare smtp client
            Dim smtp As SmtplibClient = New SmtplibClient("smtp.gmail.com", 587)
            smtp.EnableSsl = True
            smtp.UseDefaultCredentials = False
            smtp.Credentials = New NetworkCredential("SENDER@gmail.com", "PASSWORD")
            ' compose and send an email message
            Dim sendTo As String = "recipient@contoso.com" ' fake email address of a customer
            message.To.Add(sendTo)
            message.Subject = String.Format("Hello {0} at {1}.", contactName, companyName)
            message.Body = "Hello there!"
            smtp.Send(message)

            Result.ShowAlert("Email has been sent.")

        End Sub
    End Class
End Namespace
```

## C#

```
using System;
using System.Data;
using System.Collections.Generic;
using System.Linq;
using MyCompany.Data;
using System.Net.Mail;
using System.Net;
using MyCompany.Data.Objects;

namespace MyCompany.Rules
{
    public partial class CustomersBusinessRules : MyCompany.Data.BusinessRules
    {
        [ControllerAction("Customers", "Custom", "NotifySelectedCustomers")]
        public void SendAnEmailToSelectedCustomers()
        {
            MailMessage message = new MailMessage();
            message.From = new MailAddress("SENDER@gmail.com", "Web App Admin");

            // prepare smtp client
            SmtpClient smtp = new SmtpClient("smtp.gmail.com", 587);
            smtp.EnableSsl = true;
            smtp.UseDefaultCredentials = false;
            smtp.Credentials = new NetworkCredential("SENDER@gmail.com", "PASSWORD");

            // compose and send an email message to selected customers
            foreach (string customerId in Arguments.SelectedValues)
            {
                Customers customer = Customers.SelectSingle(customerId);
                // use CustomerID to find the recipient's email
                string sendTo = "recipient@contoso.com"; // fake email address of a customer

                message.To.Add(sendTo);
                message.Subject = String.Format("Hello {0} at {1}.", customer.ContactName,
                    customer.CompanyName);
                message.Body = "Hello there!";
                smtp.Send(message);
            }
            Result.ShowMessage(String.Format("{0} notifications have been sent.",
                Arguments.SelectedValues.Length));
        }

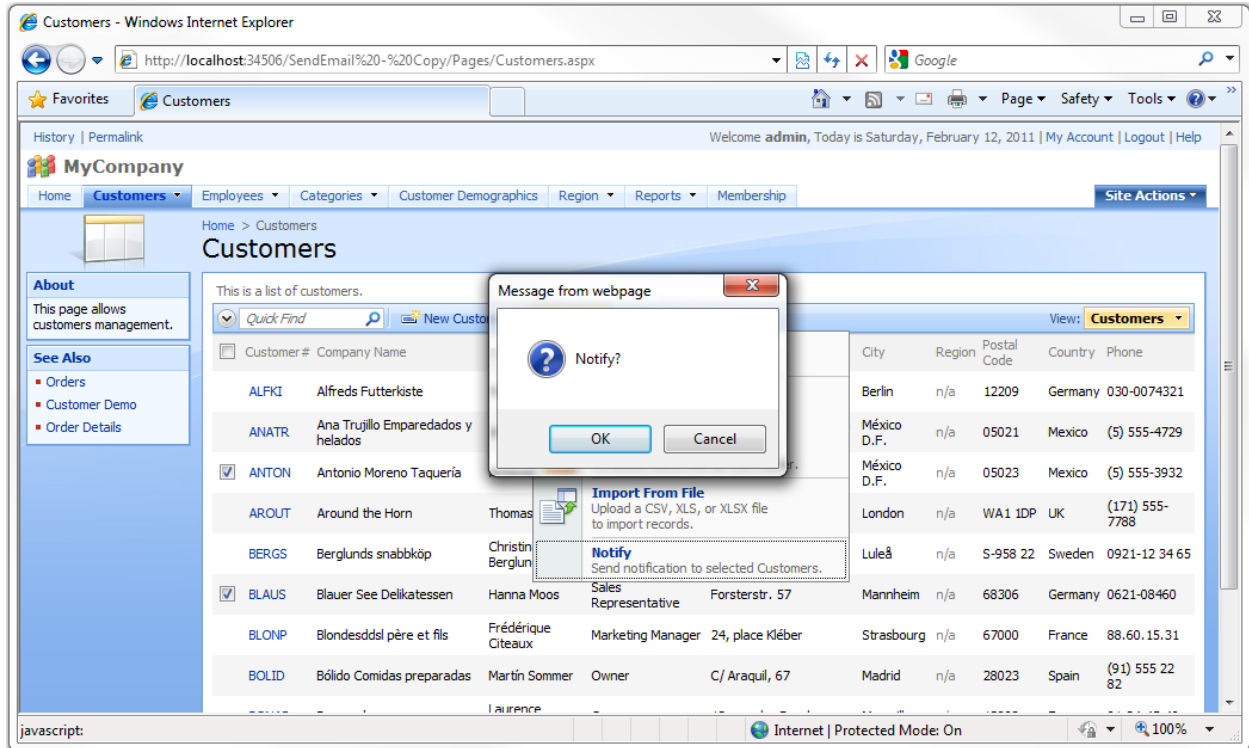
        [ControllerAction("Customers", "Custom", "SendEmail")]
        public void SendAnEmailToACustomer(string companyName, string contactName, string customerId)
        {
            MailMessage message = new MailMessage();
            message.From = new MailAddress("SENDER@gmail.com", "Web App Admin");

            // prepare smtp client
            SmtpClient smtp = new SmtpClient("smtp.gmail.com", 587);
            smtp.EnableSsl = true;
            smtp.UseDefaultCredentials = false;
            smtp.Credentials = new NetworkCredential("SENDER@gmail.com", "PASSWORD");

            // compose and send an email message
            // use CustomerID to find the recipient's email
            string sendTo = "recipient@contoso.com"; // fake email address of a customer
            message.To.Add(sendTo);
            message.Subject = String.Format("Hello {0} at {1}.", contactName, companyName);
            message.Body = "Hello there!";
            smtp.Send(message);

            Result.ShowAlert("Email has been sent.");
        }
    }
}
```

Save the business rules file, and refresh the web page. When you select multiple customers and press the “Notify” button, a confirmation will appear.



Press “OK”, and the notification emails will be sent to the selected customers.

The same exact scenario applies to any other business logic that must affect multiple records selected in a grid view.