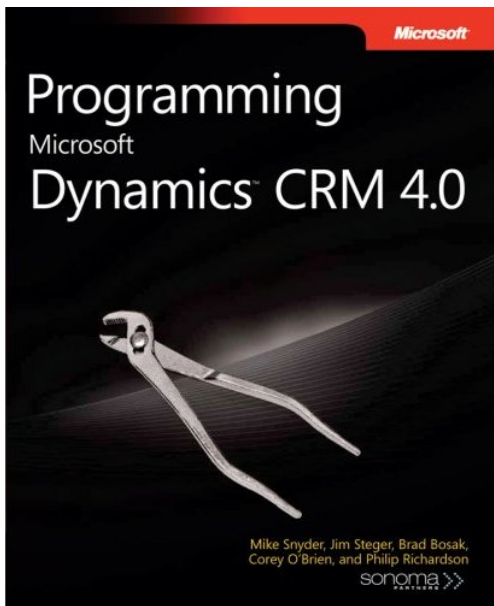


Programming Microsoft Dynamics CRM 4.0 Sample Chapter: Developing Offline Solutions



This sample chapter is from the book "Programming Microsoft Dynamics CRM 4.0" co-authored by Jim Steger, Mike Snyder, Brad Bosak, Corey O'Brien, and Philip Richardson. The book is published by Microsoft Press (ISBN: 978-0735625945) and can be purchased from Amazon.com.

Chapter 10, "Developing Offline Solutions," outlines the nuances of writing custom code that works properly using Microsoft Dynamics CRM for Outlook with Offline Access.

The book's authors, Jim Steger, Mike Snyder, Brad Bosak, and Corey O'Brien are executives at the Chicago-based consulting firm Sonoma Partners. Sonoma Partners is a Microsoft Gold Certified Partner that sells, customizes, and implements Microsoft Dynamics CRM for enterprise and midsize companies throughout the United States. Sonoma Partners has worked exclusively with Microsoft Dynamics

CRM since the version 1.0 prerelease beta software. Founded in 2001, Sonoma Partners possesses extensive experience in several industries, including financial services, professional services, health care, and real estate.

Also, feel free to check out our additional Microsoft Dynamics CRM 4.0 titles from Microsoft Press:

[Working with Microsoft Dynamics CRM 4.0](#)

[Microsoft Dynamics CRM 4.0 Step by Step](#)

For more information about the book or Sonoma Partners Microsoft CRM consulting services, please visit our website at <http://www.sonomapartners.com>.

Sonoma Partners, LLC
525 West Monroe St, Suite 240
Chicago, IL 60661
312.627.0700 p
866.766.6621 t
312.627.1305 f
www.sonomapartners.com
info@sonomapartners.com



Chapter 10

Developing Offline Solutions

One of the unique benefits of Microsoft Dynamics CRM is that your users can use Microsoft Dynamics CRM for Outlook with Offline Access to work with customer data while they are disconnected from the server. This feature can come in handy if your users need to work someplace without an Internet connection, such as on an airplane, at a trade show, or onsite at a customer's office. Microsoft Dynamics CRM refers to the concept of working disconnected from the server as working offline. When the user reconnects to the server again (known as going online), Microsoft Dynamics CRM for Outlook with Offline Access will automatically perform a bidirectional update of data changes with the server. Therefore, records created or modified while offline update to the CRM server.

As a developer, you should know how your programming customizations will behave if your company deploys Microsoft Dynamics CRM for Outlook with Offline Access. Fortunately, Microsoft Dynamics CRM includes an offline API so that most of your programming customizations can run offline in addition to online. Of course, you should learn about a few unique nuances to developing offline solutions. One of the most significant constraints of offline development is that workflow rules and asynchronous plug-ins do not execute offline.

In this chapter we will talk about communicating with the Microsoft Dynamics CRM SDK offline and then guide you through creating custom IFrame pages and plug-ins that function both online and offline. The chapter covers the following topics:

- Overview of developing with Microsoft Dynamics CRM for Outlook with Offline Access
- Offline development environment
- Offline navigation
- Communicating with the Microsoft Dynamics CRM SDK API offline
- Scripting for offline
- Developing IFrames for offline
- Developing an offline plug-in
- Offline development considerations

Overview

When you install Microsoft Dynamics CRM for Outlook with Offline Access, the software automatically installs all of the components your users need to work offline, including:

- Microsoft ASP.NET Cassini, the same Web server used by Microsoft Visual Studio 2008
- Microsoft Dynamics CRM Web files
- Microsoft SQL Server 2005 Express Edition, used to store the local copy of the Microsoft Dynamics CRM database

The Web files are installed to the same location that was selected during the install of Microsoft Dynamics CRM for Outlook. You can also find this location by looking at *InstallPath* value in the following registry key: *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSCRMCLIENT*. That folder contains a folder named Client, and under the Client folder is another directory named res that contains the Web files.



Tip The default location of the offline Web files is C:\Program Files\Microsoft Dynamics CRM\Client\res\web, and the offline database files are installed to C:\Program Files\Microsoft Dynamics CRM\sql\4.0.

When the user is offline, Microsoft Dynamics CRM references data in the local database instead of the data on the CRM server. By default, Microsoft Dynamics CRM for Outlook does not copy the entire server database to the local database. Instead, Microsoft Dynamics CRM for Outlook uses local data groups to determine which subset of the data it should copy to the client computer. Users can modify their local data group within the CRM menu in Microsoft Dynamics CRM for Outlook.



Tip The MSDN Code Gallery offers a tool that you can use to automate the local data group creation for multiple users. You can find this tool at <http://code.msdn.microsoft.com/mscrmlocaldatagroup>.

During your offline development, you may want to query or manipulate your test data in the offline database. You can do this easily by using Microsoft SQL Server Management Studio Express to connect to the offline database. You can download SQL Server Management Studio Express for free from the Microsoft Web site: <http://www.microsoft.com/downloads/>.

Offline Development Environment

Because the offline API functions very similarly to the online API, you won't have much trouble understanding how it works. Therefore, one of the trickiest parts of developing an offline solution for Microsoft Dynamics CRM might be configuring your offline development environment. To develop for the offline client, you need an install of the offline client configured and pointed at your Microsoft Dynamics CRM install and organization. When you setup your offline development environment, you do have a few other options for deploying and testing:

- Workstation
- Laptop
- Virtual PC

Workstation

Deploying and testing your offline code on your workstation is the simplest and quickest method of the three. If you are currently not using the Microsoft Dynamics CRM for Outlook client on your workstation, you can install and configure it to point at your Microsoft Dynamics CRM development server.

This approach does have a few drawbacks. You can only configure Microsoft Dynamics CRM for Outlook with Offline Access for one user per computer, so it's not ideal if your testing requires multiple users or users in different roles. If another user logs on to your workstation and tries to configure the Outlook Client, she receives an error message stating that Microsoft Dynamics CRM for Outlook with Offline Access can only be configured for one user (Figure 10-1).

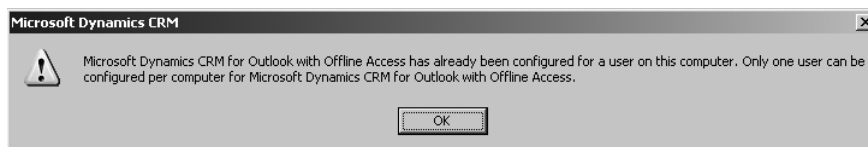


FIGURE 10-1 Microsoft Dynamics CRM for Outlook with Offline Access configuration error

In addition, if you install Microsoft Dynamics CRM for Outlook on your personal workstation, you run the risk of mixing up your regular Outlook information with contacts, appointments, and tasks from the development system.

Laptop

Another option is to install the offline client on a laptop. Although this option still only allows for the offline client to be configured for one user account, you have the option to share the computer with other developers or testers by simply moving the laptop from office to office.

Virtual PC

If you need to test your offline code with multiple user accounts or can't use the workstation or laptop option, you can create a Virtual PC with Microsoft Outlook installed on it. Then you can install and configure the Microsoft Dynamics CRM Outlook client on the Virtual PC and create a new Virtual PC for each user account you test with. You can then share the Virtual PC with multiple developers and testers.

Offline Navigation

While customizing Microsoft Dynamics CRM, you may add your own navigation and menu items through the ISV Config and the SiteMap. By default these custom elements are displayed in Microsoft Dynamics CRM for Outlook with Offline Access.

The ISV.Config allows you to add links to the left navigation area of an entity, custom menus, and items to an entity's toolbar, and custom buttons to an entity's grid. The SiteMap allows you to customize the Microsoft Dynamics CRM main navigation. Just as with the ISV Config, the SiteMap allows you to specify which clients your customizations are displayed in and also whether they are displayed in offline mode.

Client Attribute

You can use the *Client* attribute on an ISV Config or SiteMap node to configure which Microsoft Dynamics CRM clients display the customization. This is useful if you only want your custom navigation element to show in the Web interface or only in the Microsoft Dynamics CRM for Outlook client. The default value for this attribute is set to display in all clients. The ISV Config *Client* attributes can be set to either Web or Outlook. The following ISV Config elements implement the *Client* attribute:

- *Button*
- *Entity*
- *MenuItem*
- *NavBarItem*

The *SubArea* element of the SiteMap also has the *Client* attribute. It also defaults to display in all clients and can be updated to any of the following values:

- *All*
- *Outlook*
- *OutlookLaptopClient*
- *OutlookWorkstationClient*
- *Web*

The *OutlookLaptopClient* value refers to Microsoft Dynamics CRM for Outlook with Offline Access. The *OutlookWorkstationClient* value refers to the non-offline enabled version of Microsoft Dynamics CRM for Outlook. You can set multiple values by entering them as a comma-separated list. The following code sample shows a SiteMap *SubArea* that only appears in the Web client:

```
<SubArea Id="nav_calendar"
  Icon="/_imgs/area/18_calendar.gif"
  ResourceId="Homepage_Calendar"
  Url="/workplace/home_calendar.aspx"
  Client="Web">
  <Privilege Entity="activitypointer" Privilege="Read" />
</SubArea>
```

AvailableOffline Attribute

If you want to disable your custom navigation or buttons when the user is offline, you need to specifically set the *AvailableOffline* attribute of your element to false. This optional attribute defaults to true unless otherwise specified. The *AvailableOffline* attribute is available on the following ISV Config elements:

- *Button*
- *Entity*
- *MenuItem*
- *NavBarItem*

The *AvailableOffline* attribute is also available on the *SubArea* element of the SiteMap. The following code sample shows a button added to the ISV Config that is only available in the Outlook client and shows up in offline mode:

```
<Button Icon="/_imgs/ico_18_debug.gif"
  JavaScript="alert('test');"
  Client="Outlook" AvailableOffline="true">
  <Titles>
  <Title LCID="1033" Text="Outlook Only" />
  </Titles>
  <ToolTips>
  <ToolTip LCID="1033" Text="Outlook Only - This is available offline also." />
  </ToolTips>
</Button>
```

Communicating with the Microsoft Dynamics CRM SDK API Offline

Both the *CrmService* and *MetadataService* provided by the Microsoft Dynamics CRM SDK contain functionality that you can use in your offline programming. You can further extend the Microsoft Dynamics CRM for Outlook client using the Microsoft Dynamics CRM Outlook SDK.



Note In offline mode the Microsoft Dynamics CRM Web services will always use integrated authentication to authenticate the user. You cannot use impersonation while offline.

CrmService Offline

You can use the *CrmService* offline in the same way you use it online. Any data manipulation done in offline mode is stored in the user's local database and Microsoft Dynamics CRM will automatically synchronize with the main database the next time the user goes online.



Warning The synchronization process does not merge field-level changes with existing data. If you update a record offline, the sync process will update the entire record on the server. It's possible to encounter a scenario where an offline user modifies a record, and an online user modified that same record during the time the user was offline. In this scenario, the user who modified the record last wins and his changes survive on the server. This is true even if the two users modified different fields on the record.

Offline Message and Entity Availability

You can check whether a *CrmService* message is available offline by checking its availability, which you can do by querying the *sdkmmessage* entity and checking its *availability* attribute. Table 10-1 contains a list of the possible *availability* values and what they mean. If the message can be used for more than one entity, you should also query the *sdkmmessagefilter* entity's *availability* attribute to make sure the message works for the entity you are trying to use it with. For example, the *Create* message works both online and offline, but the *AddMemberList* message only works online.

TABLE 10-1 Message Availability Values

Value	Description
0	Message is only available on the server (online).
1	Message is only available on the client (offline).
2	Message is available both on the server and the client (online and offline).

The Microsoft Dynamics CRM 4.0 SDK contains a list of entities that are available for offline use. You can also check this programmatically by retrieving the entity's *EntityMetadata* and checking its *EntityMetadata.IsAvailableOffline* property. The *IsAvailableOffline* property is a *CrmBoolean* data type. The following code sample shows an example of checking an *EntityMetadata*'s offline availability:

```
// entityMetadata is an instance of the EntityMetadata class
if (entityMetadata.IsAvailableOffline.Value)
{
    // custom offline logic here
}
```

Creating a *CrmService* Instance Offline

To create an instance of the *CrmService* offline, you need to extract a few values from the registry. Because the offline Web site is running on the local Cassini Web server we can use "localhost" as the server name. As you can see from the following code sample, we then need to grab the port number from the *CassiniPort* registry entry and the organization name from the *ClientAuthOrganizationName* registry entry. We can then construct our server as we normally would.

```
RegistryKey regkey = Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\MSCRMClient");
string orgName = regkey.GetValue("ClientAuthOrganizationName").ToString();
string portNumber = regkey.GetValue("CassiniPort").ToString();

string baseUrl = "http://localhost:" + portNumber + "/mscrmservices/2007/";
crmUrl = baseUrl + "crmservice.asmx";

CrmAuthenticationToken token = new CrmAuthenticationToken();
token.OrganizationName = orgName;
token.AuthenticationType = 0;

CrmService service = new CrmService();
service.Credentials = System.Net.CredentialCache.DefaultCredentials;
service.CrmAuthenticationTokenValue = token;
service.Url = crmUrl;
```



Tip You can find the Microsoft Dynamics CRM for Outlook with Offline Access registry values at HKEY_CURRENT_USER\Software\Microsoft\MSCRMClient.

MetadataService Offline

When using the *MetadataService* offline, only the retrieve messages are available. You cannot manipulate entities—such as programmatically adding an attribute—while offline. This restriction is in place to prevent a conflict when the user goes online and the databases are synchronized. Chapter 8, "Developing with the Metadata Service," contains sample code for creating the *MetadataService* while running in offline mode.

Microsoft Dynamics Outlook SDK

You can use the *Microsoft.Crm.Outlook.Sdk* assembly to customize Microsoft Dynamics CRM for Outlook. The assembly contains a class named *CrmOutlookService* that has a few useful methods for programming for offline. To use the *CrmOutlookService* you create an instance the same way you do with the *CrmService* class. Table 10-2 lists the methods available for you to use from the *CrmOutlookService* instance, and Table 10-4 lists all of its properties. The *State* property of the *CrmOutlookService* has a value with the type of *ClientState*. Table 10-5 lists all of the possible *ClientState* values.

TABLE 10-2 *CrmOutlookService* Methods

Name	Description
<i>GoOffline</i>	Executes the same functionality as the Go Offline button. The online database data and customizations are synchronized to the user's local database.
<i>GoOnline</i>	Executes the same functionality as the Go Online button. The user's local database will be synchronized with the online database.
<i>SetOffline</i>	Sets Microsoft Dynamics CRM for Outlook with Offline Access to offline mode without synchronizing the online database with the local offline database.
<i>Sync</i>	Executes the synchronization between the online Microsoft Dynamics CRM database and Microsoft Dynamics CRM for Outlook.

The *Sync* method on the *CrmOutlookService* takes one argument of the type *OutlookSyncType*. This determines whether the online data will be synchronized with the client computer's Outlook store (for example, contacts, tasks, and appointments), the offline database, or the client computer's Outlook address book. Table 10-3 contains possible values for the *OutlookSyncType* enumeration.

TABLE 10-3 *OutlookSyncType* Members

Name	Description
<i>Outlook</i>	The synchronization occurs between Microsoft Dynamics CRM and the client computer's Outlook store.
<i>AddressBook</i>	The synchronization occurs between Microsoft Dynamics CRM and the client computer's Outlook address book.
<i>Offline</i>	The synchronization occurs between Microsoft Dynamics CRM and the local offline database.

TABLE 10-4 *CrmOutlookService* Properties

Name	Type	Description
<i>IsCrmClientLoaded</i>	<i>Boolean</i>	Returns true if the Microsoft Dynamics CRM for Outlook client has been loaded by Microsoft Outlook.
<i>IsCrmClientOffline</i>	<i>Boolean</i>	Returns true if Microsoft Dynamics CRM for Outlook with Offline Access is currently in offline mode.

TABLE 10-4 *CrmOutlookService* Properties

Name	Type	Description
<i>IsCrmDesktopClient</i>	<i>Boolean</i>	Returns true if Microsoft Dynamics CRM for Outlook has been installed. Returns false if Microsoft Dynamics CRM for Outlook with Offline Access has been installed.
<i>ServerUri</i>	<i>Uri</i>	Returns the Uri to connect to the Microsoft Dynamics CRM SDK. The return value differs depending on whether the client is online or offline.
<i>State</i>	<i>ClientState</i>	Returns the current state of the client.

TABLE 10-5 *ClientState* Members

Field	Description
<i>Online</i>	The client is online.
<i>Offline</i>	The client is offline.
<i>SyncToOutlook</i>	The client is currently in the synchronization process between the online server and Outlook.
<i>SyncToOutlookError</i>	An error has occurred during the synchronization with Outlook.
<i>GoingOffline</i>	The client is currently in the process of going into offline mode.
<i>GoingOnline</i>	The client is currently in the process of going into online mode.
<i>ClientLoadFailure</i>	Microsoft Outlook failed to load the Microsoft Dynamics CRM for Outlook client.
<i>ClientVersionLower</i>	The client version is lower than the version of Microsoft Dynamics CRM installed on the server.
<i>ClientVersionHigher</i>	The client version is higher than the version of Microsoft Dynamics CRM installed on the server.
<i>PostOfflineUpgrade</i>	The client has been upgraded while in offline mode.
<i>OnlineCrmNotAvailable</i>	The online Microsoft Dynamics CRM server is currently unavailable.
<i>GoingOfflineCanceled</i>	The user canceled the process of going offline.
<i>BackgroundGoingOffline</i>	The process of synchronizing the offline database is currently running in the background.

Scripting for Offline

In Chapter 7, “Form Scripting,” you learned about adding custom JavaScript code to the Microsoft Dynamics CRM entity forms and attributes. Microsoft Dynamics CRM will automatically take all custom scripts offline to the client computer—you cannot specify some scripts to run offline or online only. Therefore, you need to make sure that your scripts will not encounter errors when they run offline disconnected from the server. Table 7-2 from Chapter 7 lists global functions available to you for detecting which client the user is on and whether the user is online or offline. You will see an example of this when we create our offline IFrame in the following section.

Developing IFrames for Offline

If you have developed custom IFrame or Web pages for your Microsoft Dynamics CRM system, you most likely want them to function for your users when they go offline with the Microsoft Dynamics CRM for Outlook with Offline Access. By planning ahead of time for offline usage, you can easily create custom Web pages and IFrames that will work both offline and online. However, some scenarios might exist where your custom Web pages cannot function offline because the Internet connection is absent:

- When it requires access to an external Web Service
- When it displays content from an external Web site
- When it retrieves or updates data from an external database

Of course, these scenarios are not caused by a Microsoft Dynamics CRM constraint. Instead, they are simply a limitation because no connectivity to these external systems exists. In cases where you know your IFrame cannot function offline, it is best to display a message explaining this to the user, or have your IFrame detect whether the user is offline and redirect the user to an offline friendly page.

Programming the IFrame

In this section we will develop an IFrame page that functions both online and offline. We will add the IFrame to the native Account entity's form and display contact information from the Account's selected primary contact. If the user views the page while in offline mode, we alert the user to let her know the information she is viewing may not be up to date. If you want to peek ahead to see what the finished output looks like, please refer to Figure 10-3.

Adding the AccountPrimaryContactInfo.aspx page

1. Open the ProgrammingWithDynamicsCrm4 solution with Microsoft Visual Studio 2008.
2. If you have not created the ProgrammingMicrosoftDynamicsCrm4.Web project in a previous chapter, right-click the solution name, add a new Web Site project named **ProgrammingMicrosoftDynamicsCrm4.Web** and add references to these Microsoft Dynamics CRM SDK assemblies: Microsoft.Crm.Sdk.dll and Microsoft.Crm.SdkProxy.dll.
3. Right-click the ProgrammingWithDynamicsCrm4.Web project and select Add New Item.
4. Select the Web Form template and type **AccountPrimaryContactInfo** in the Name box.
5. Click Add.

The preceding steps add two new files to the Web project: AccountPrimaryContactInfo.aspx and AccountPrimaryContactInfo.aspx.cs. Listing 10-1 shows the source code for our user interface page.

LISTING 10-1 AccountPrimaryContactInfo.aspx source code

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeFile="AccountPrimaryContactInfo.aspx.cs"
    Inherits="IFrames_AccountPrimaryContactInfo" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>

    <script type="text/javascript">

        function window.onload()
        {
            if ( !parent.IsOnline() )
            {
                document.getElementById("message").innerHTML =
                "Warning: You are currently offline. The contact information
                displayed on this page may be out of date.";
            }
        }

    </script>

</head>
<body style="font-size: 11px;
    font-family: Tahoma;
    margin: 0px;
    border: 0px;
    background-color: #eaf3ff;">
    <form id="form1" runat="server">
    <table cellpadding="1"
        cellspacing="0"
        style="width: 100%; height: 100%; table-layout: fixed;"
        border="0">
        <colgroup>
            <col style="width: 111;" />
            <col />
            <col style="width: 115; padding-left: 23px;" />
            <col />
        </colgroup>
        <tr>
            <td colspan="4">
                <div id="message" style="color: Red" />
            </td>
        </tr>
    </table>
    </form>
</body>
</html>
```

```

        <td>
            Full Name
        </td>
        <td>
            <asp:Label ID="fullname" runat="server" />
        </td>
        <td>
            &nbsp;
        </td>
        <td>
            &nbsp;
        </td>
    </tr>
    <tr>
        <td>
            Business Phone
        </td>
        <td>
            <asp:Label ID="phone" runat="server" />
        </td>
        <td>
            E-mail
        </td>
        <td>
            <asp:Label ID="email" runat="server" />
        </td>
    </tr>
    <tr>
        <td>
            City
        </td>
        <td>
            <asp:Label ID="city" runat="server" />
        </td>
        <td>
            State
        </td>
        <td>
            <asp:Label ID="state" runat="server" />
        </td>
    </tr>
</table>
</form>
</body>
</html>

```

The display is a simple HTML table laid out similarly to the four-column layout used on the Microsoft Dynamics CRM entity forms. You will also notice that styles were added to the page to help blend it into the Account entity form. The end user will most likely not even realize that he is looking at custom IFrame page. You can find the styles used in the Microsoft Dynamics UI Style Guide included in the Microsoft Dynamics CRM SDK.

The code in the JavaScript block at the top of the aspx page is used to alert the user when the IFrame is being displayed in offline mode. The code calls the *IsOnline* method of the entity form:

```
if ( !parent.IsOnline() )
{
    document.getElementById("message").innerHTML = "Warning: You are currently offline.
The contact information displayed on this page may be out of date.";
}
```

The code to actually retrieve our Contact data is located in the code-behind file. Listing 10-2 shows the source code for this file.

LISTING 10-2 AccountPrimaryContactInfo.aspx.cs source code

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using Microsoft.Crm.Sdk;
using Microsoft.Win32;
using Microsoft.Crm.Sdk.Query;
using Microsoft.Crm.SdkTypeProxy;

public partial class IFrames_AccountPrimaryContactInfo : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string id = Request.QueryString["id"];

        if (string.IsNullOrEmpty(id))
            throw new Exception("The required query string parameter \"id\" was not
found.");

        CrmService crmService = GetCrmService();

        // First retrieve the primary contact for the account
        ColumnSet accountCols = new ColumnSet();
        accountCols.AddColumn("primarycontactid");

        account crmAccount = (account)crmService.Retrieve("account",
                                                    new Guid(id),
                                                    accountCols);

        if (crmAccount.primarycontactid != null)
        {
            Guid primaryContactId = crmAccount.primarycontactid.Value;
        }
    }
}
```

```

        ColumnSet contactCols = new ColumnSet();
        contactCols.AddColumns("fullName",
                               "telephone1",
                               "emailaddress1",
                               "address1_city",
                               "address1_stateorprovince");

        contact crmContact = (contact)crmService.Retrieve("contact",
                                                         primaryContactId,
                                                         contactCols);

        this.fullName.Text = crmContact.fullName;
        this.phone.Text = crmContact.telephone1;
        this.city.Text = crmContact.address1_city;
        this.state.Text = crmContact.address1_stateorprovince;
        this.email.Text = crmContact.emailaddress1;
    }
}

private CrmService GetCrmService()
{
    CrmAuthenticationToken token = new CrmAuthenticationToken();
    string crmUrl;
    string orgName = String.Empty;
    bool online = true;

    if (Request.Url.Host.ToString() == "127.0.0.1")
    {
        online = false;
    }

    if (online)
    {
        RegistryKey regkey =
            Registry.LocalMachine.OpenSubKey("SOFTWARE\\Microsoft\\MSCRM");
        string serverUrl = regkey.GetValue("ServerUrl").ToString();
        crmUrl = serverUrl + "/2007/crmservice.asmx";

        if (Request.QueryString["orgname"] != null)
        {
            orgName = Request.QueryString["orgname"];
        }
    }
    else
    {
        RegistryKey regkey =
            Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\MSCRMClient");
        orgName = regkey.GetValue("ClientAuthOrganizationName").ToString();
        string portNumber = regkey.GetValue("CassiniPort").ToString();

        string baseUrl = "http://localhost:"
            + portNumber
            + "/mscrmservices/2007/";
    }
}

```

```

        crmUrl = baseUrl + "crmservice.asmx";
    }

    token.OrganizationName = orgName;
    token.AuthenticationType = 0;

    CrmService service = new CrmService();
    service.Credentials = System.Net.CredentialCache.DefaultCredentials;
    service.CrmAuthenticationTokenValue = token;
    service.Url = crmUrl;

    return service;
}
}

```

Because this page needs to function both online and offline, we need to accommodate the different locations of the Microsoft Dynamics CRM SDK services. We used the *GetCrmService* method in Listing 10-2 to accomplish this. As we discussed earlier in the section “Communicating with the Microsoft Dynamics CRM SDK API Offline,” if the page is being run offline, the values needed by the *CrmService*, such as port number and organization name, are pulled from the client computer’s registry.

After the *CrmService* has been created, the rest of the functionality is pretty straightforward. First we attempt to retrieve the primary contact *Guid* from the Account. If a value is returned, we then retrieve the necessary fields from that Contact record and populate our *Label* controls.

Deploying the IFrame Page

Now that we have completed our IFrame development, we need to update the Account form and deploy our Web files to the Microsoft Dynamics CRM server and client computer.

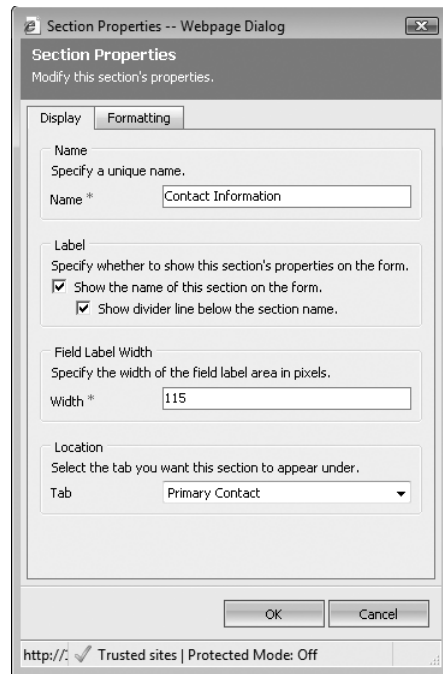
Microsoft Dynamics CRM Customizations

The first thing we need to do is add our IFrame to the Account entity’s form. To accomplish this, we add a new tab named Primary Contact that contains an IFrame pointed at our aspx page.

Adding the IFrame to the Account Form

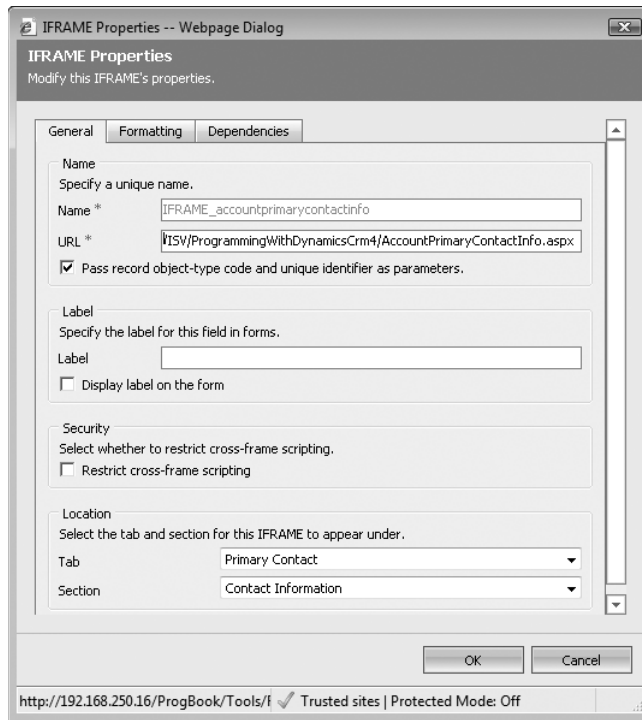
1. Open Microsoft Dynamics CRM in a Web browser and navigate to the Account entity’s form customizations screen.
2. Click Add A Tab in the Common Tasks menu.
3. Type **Primary Contact** into the Name field and click OK.

4. An IFrame must be housed in a section, so click Add A Section from the Common Tasks menu.
5. Type **Contact Information** into the Name field.
6. Select both the Show The Name Of This Section On The Form and the Show Divider Line Below The Section Name check boxes and verify that the tab is set to our new Primary Contact tab. Click OK.

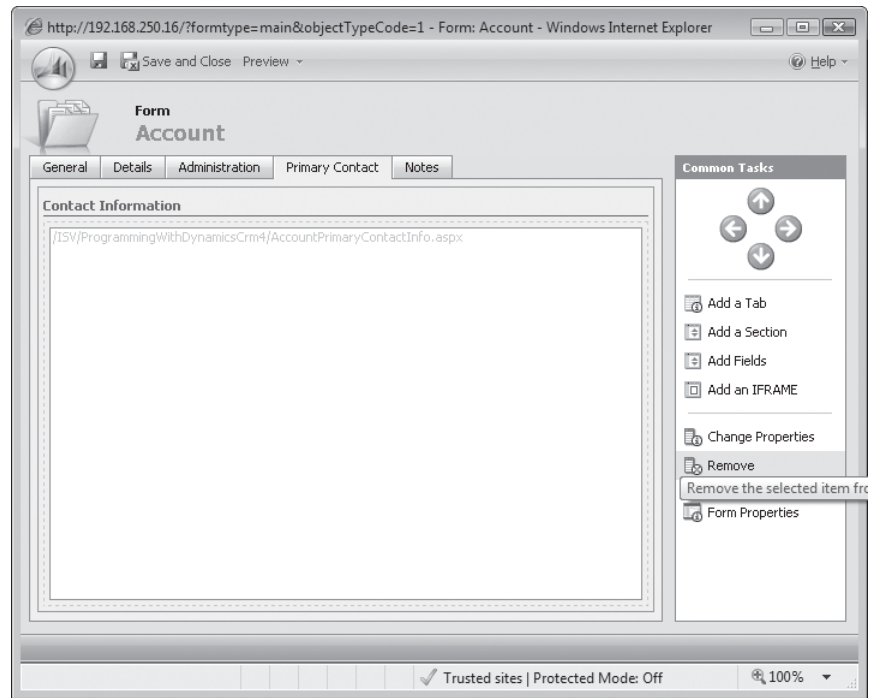


7. Select our new section and click Add An IFrame in the Common Tasks menu.
8. Enter **accountprimarycontactinfo** in the Name field.
9. Later we will deploy our Web files under the ISV folder of Microsoft Dynamics CRM, so type **/ISV/ProgrammingWithDynamicsCrm4/AccountPrimaryContactInfo.aspx** in the URL field.
10. Select the Pass Record Object-Type Code And Unique Identifier As Parameters check box.

11. Clear the Restrict Cross-Frame Scripting check box.
12. Verify that Tab is set to Primary Contact and Section is set to Contact Information.



13. In the Formatting tab's Row Layout section, select the Automatically Expand To Use Available Space check box.
14. In the Formatting tab's Border section, clear the Display Border check box. This makes our IFrame integrate seamlessly into the form. Click OK.



15. Save the form and publish your customizations.

After these changes are published, they are synced with the offline customizations in Microsoft Dynamics CRM for Outlook with Offline Access the next time the user goes offline.

Microsoft Dynamics CRM Web Server

For our IFrame to function while the user is online, we need to deploy our files to the CRM Web server. You can find a more detailed look at deployment in Chapter 9, "Deployment."

Deploying the Web files

1. On the Microsoft Dynamics CRM server, create a new folder named **Programming-WithDynamicsCRM4** under the ISV folder in the directory with your Microsoft Dynamics CRM Web files.
2. Copy the `AccountPrimaryContactInfo.aspx` and `AccountPrimaryContactInfo.aspx.cs` files into the newly created folder.

Microsoft Dynamics CRM for Outlook with Offline Access

Because Microsoft Dynamics CRM for Outlook with Offline Access uses its own set of Web files in offline mode, you also need to deploy the Web files to each client computer under the ISV folder in the directory with the offline Web files. You can do this using the same steps in the previous section.



Tip The default location for the Web files for Microsoft Dynamics CRM for Outlook with Offline Access is C:\Program Files\Microsoft Dynamics CRM\Client\res\web\.

Testing the IFrame Page Offline

Now we will test our new IFrame when working offline. Open your instance of Microsoft Outlook that has Microsoft Dynamics CRM for Outlook with Offline Access installed and configured. Click the Go Offline button. When the synchronization process completes and the Go Offline button has been replaced with a Go Online button, navigate to the Accounts grid (Figure 10-2).

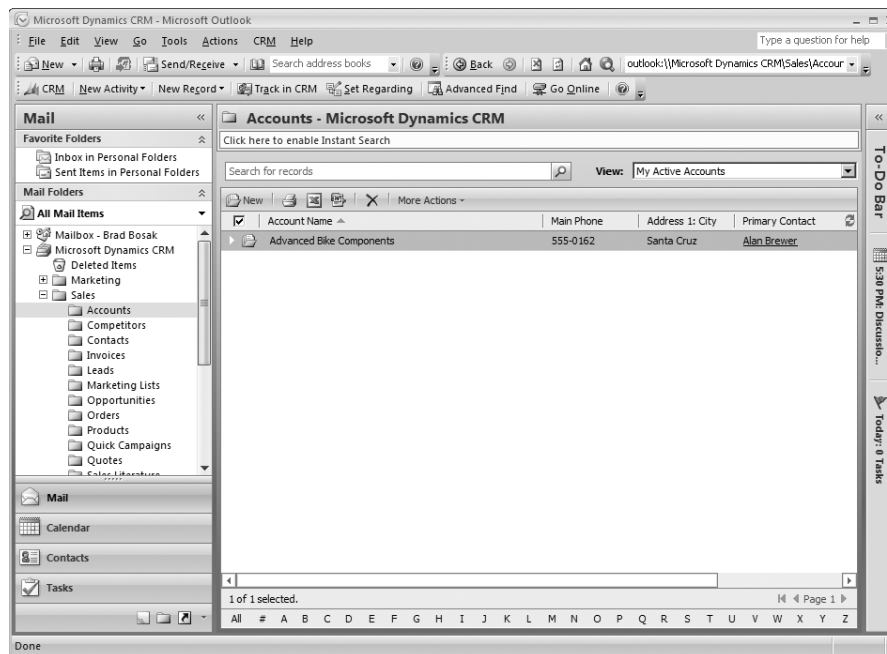


FIGURE 10-2 Offline Accounts view

Open an Account record that has the Primary Contact field populated, or open an Account, populate the Primary Contact field, and click Save. Click the Primary Contact tab we added. You should see the Primary Contact information along with our warning message (Figure 10-3).

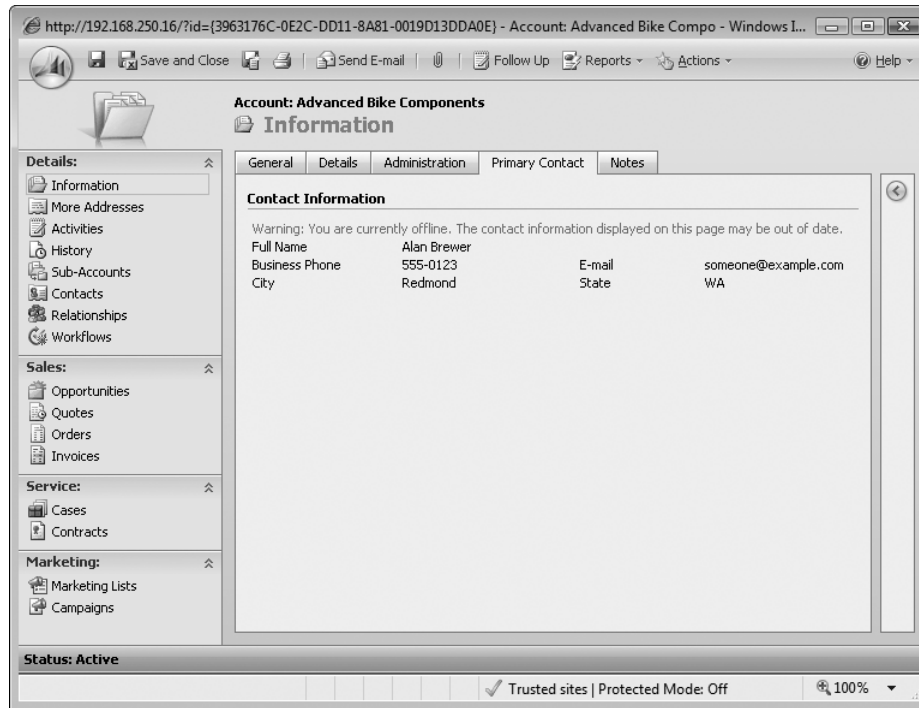


FIGURE 10-3 IFrame displayed in offline mode

Close the Account window and click the Go Online button. When the synchronization process is complete and you no longer see the Go Online button, reopen the same Account record and click the Primary Contact tab. You can now see the Primary Contact Information, but the warning message is gone because you are viewing the actual online form (Figure 10-4).

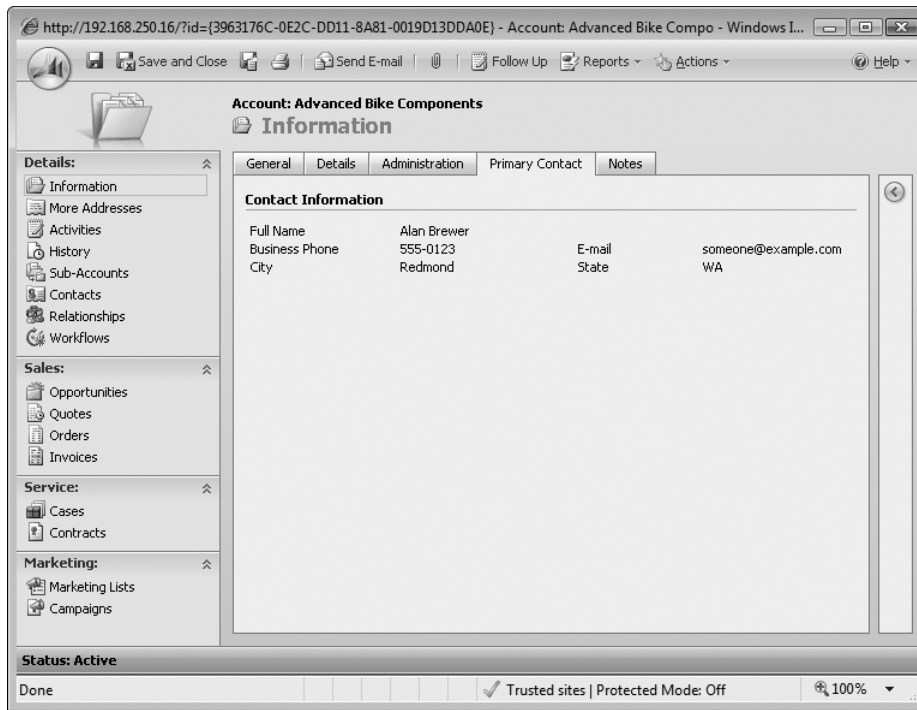


FIGURE 10-4 IFrame displayed in online mode

Developing an Offline Plug-in

Chapter 5, “Plug-ins,” gave some examples of creating and deploying plug-ins for Microsoft Dynamics CRM. In this section we expand on this by creating a simple plug-in that functions while running Microsoft Dynamics CRM for Outlook in offline mode.

Programming the Plug-in

We’ll develop the offline-capable plug-in in a similar manner as the other plug-ins we have developed, but we need to add logic to determine when the client computer is in offline mode. Our plug-in will update the subject of a Lead record when it is created. We will update the subject to the Lead’s last name followed by its first name and the current date. When you create plug-ins, you want to be careful that Microsoft Dynamics CRM doesn’t execute them twice (once offline and once online). In a situation where the user is offline and creates a lead and then returns online at a later date, our plug-in would be run both when the lead was created offline and when the user returned online. As you will see later in this section, we add a condition to the plug-in to prevent this from happening. Listing 10-3 shows the entire source code for the LeadTopicUpdater plug-in. Before we start coding, let’s talk briefly

about the Microsoft Dynamics CRM customizations that we need for this example to function properly. Because the plug-in automatically updates the subject attribute on the Lead entity, we disabled this attribute on the Lead form and set its requirement level to No Constraint. This allows the user to create a new lead without entering a subject.

Adding the LeadTopicUpdater.cs file

1. Open the ProgrammingWithDynamicsCrm4 solution with Microsoft Visual Studio 2008.
2. Right-click the ProgrammingWithDynamicsCrm4.Plugins project we created in Chapter 5 and select Add New Item.
3. Select the Class template and type **LeadTopicUpdater** in the Name box.
4. Click Add.

LISTING 10-3 LeadTopicUpdater.cs source code

```
using System;
using System.Collections.Generic;
using System.Text;
using ProgrammingWithDynamicsCrm4.Plugins.Attributes;
using Microsoft.Crm.Sdk;
using System.Xml;
using Microsoft.Crm.SdkTypeProxy;

namespace ProgrammingWithDynamicsCrm4.Plugins
{
    [PluginStep("Create",
        PluginStepStage.PostEvent,
        Description = "Updates the lead's topic",
        StepId = "LeadPostCreate",
        PrimaryEntityName = "lead",
        Mode = PluginStepMode.Synchronous,
        SupportedDeployment = PluginStepSupportedDeployment.Both)]

    public class LeadTopicUpdater : IPlugin
    {
        public void Execute(IPluginExecutionContext context)
        {
            DynamicEntity target =
                (DynamicEntity)context.InputParameters[ParameterName.Target];

            if (!target.Properties.Contains("subject") ||
                string.IsNullOrEmpty(target.Properties["subject"].ToString()))
            {
                ICrmService crmService = context.CreateCrmService(true);

                lead crmLead = new lead();
                crmLead.leadid = new Key();
                crmLead.leadid.Value =
                    new Guid(context.OutputParameters[ParameterName.Id].ToString());
            }
        }
    }
}
```

```

// we will create a new topic with the format:
// Last Name, First Name - Created Date
string firstName = String.Empty;
string lastName = String.Empty;
string date = String.Empty;

if (target.Properties.Contains("firstname"))
    firstName = target.Properties["firstname"].ToString();
if (target.Properties.Contains("lastname"))
    lastName = target.Properties["lastname"].ToString();

// get the user's date format
string fetch = String.Format(@"<fetch mapping='logical'>
    <entity name='usersettings'>
        <attribute name='dateformatstring' />
        <filter type='and'>
            <condition
                attribute='systemuserid'
                operator='eq'
                value='{0}' />
        </filter>
    </entity>
</fetch>", context.UserId.ToString());

string fetchResults = crmService.Fetch(fetch);

XmlDocument resultDoc = new XmlDocument();
resultDoc.LoadXml(fetchResults);

string dateFormat =
resultDoc.DocumentElement.FirstChild.SelectSingleNode("dateformatstring").InnerText;

date = DateTime.Now.ToString(dateFormat);

crmLead.subject = String.Format("{0}, {1} - {2}",
                                lastName,
                                firstName,
                                date);

    crmService.Update(crmLead);
}
}
}
}

```

We want to register the plug-in for both the Web and Outlook clients, so note that we set the *SupportedDeployment* of the *PluginStep* attribute to *Both*. You can find the *PluginStepSupportedDeployment* enum in the *ProgrammingWithDynamicsCrm4.Plugins.Attributes* project we created in Chapter 5. Table 10-6 lists the fields of the *PluginStepSupportedDeployment* enum along with a description. Another thing to note is that the *Mode* is set to *Synchronous*. *Asynchronous* plug-ins do not run in offline mode.



Note The *PluginStep* attribute is discussed in detail in Chapter 5.

TABLE 10-6 SupportedDeployment Members

Field	Value	Description
<i>Both</i>	2	Registers the plug-in for both the Web server and the Outlook Client
<i>OutlookClientOnly</i>	1	Registers the plug-in for only the Outlook Client
<i>ServerOnly</i>	0	Registers the plug-in for only the Web server

The logic in our plug-in is pretty simple. We first check to see whether the lead currently has the subject field populated. If it doesn't have a subject, we then create an instance of the lead object, update its subject to our specified format, and call the *Update* method of the *CrmService*. Because we are checking to see whether a subject already exists, we are safe if the Plug-in is run more than once. This prevents the date from being incorrectly updated if the user created the lead offline. If the check is not in place, the next time the user goes online, the subject is updated with the current date instead of keeping the actual date the lead was created.

Another way to prevent a plug-in from running twice is to check the *CallerOrigin* property of the *IPluginExecutionContext* argument that is passed into the plug-in's *Execute* method. If the origin is of the type *OfflineOrigin*, the plug-in was fired when the offline client went into online mode. This concept is demonstrated in the following code snippet:

```
using System;
using Microsoft.Crm.Sdk;
using Microsoft.Crm.SdkTypeProxy;

public class OnlinePlugin : IPlugin
{
    public void Execute(IPluginExecutionContext context)
    {
        // Check to see if the plug-in was executed when going back online.
        CallerOrigin callerOrigin = context.CallerOrigin;
        if (callerOrigin is OfflineOrigin)
        {
            // The plug-in was executed when the user clicked the go online button
            return;
        }
        else
        {
            // add the plug-in logic here
        }
    }
}
```



Tip If you want a piece of code to only run online or offline, you can check the value of the *IsExecutingInOfflineMode* property on the *IPluginExecutionContext* argument that is passed into your plug-in's *Execute* method.

Deploying the Plug-in

The "Deployment" section of Chapter 5 has all of the necessary steps for deploying the plug-in. You should deploy the plug-in to the database instead of the file system so that Microsoft Dynamics CRM automatically registers the plug-in on the client's computer the next time the client goes offline.

After we register the plug-in, we have one more step before it can run in offline mode. Microsoft Dynamics CRM for Outlook with Offline Access has another layer of security for offline plug-ins. When Microsoft Dynamics CRM for Outlook with Offline Access is installed it creates a registry key called *AllowList*. We need to add the public token key guid of our plug-in assembly as a new key under the *AllowList* key.



Note See the "Offline Applications" section of Chapter 9 for detailed instructions on adding the Plug-in assembly's public token key guid to the *AllowList* registry key.

Testing the Plug-in

To test our lead plug-in, open your instance of Microsoft Outlook that has Microsoft Dynamics CRM for Outlook with Offline Access installed and configured. Click the Go Offline button. When the synchronization process is complete and the Go Offline button has been replaced with a Go Online button, navigate to the Leads grid. Click New, and fill out the new Lead form. Click the Save button or the Save And Close button. Our plug-in should have executed and you can now see the subject updated with the Lead's name followed by the current date (Figure 10-5).

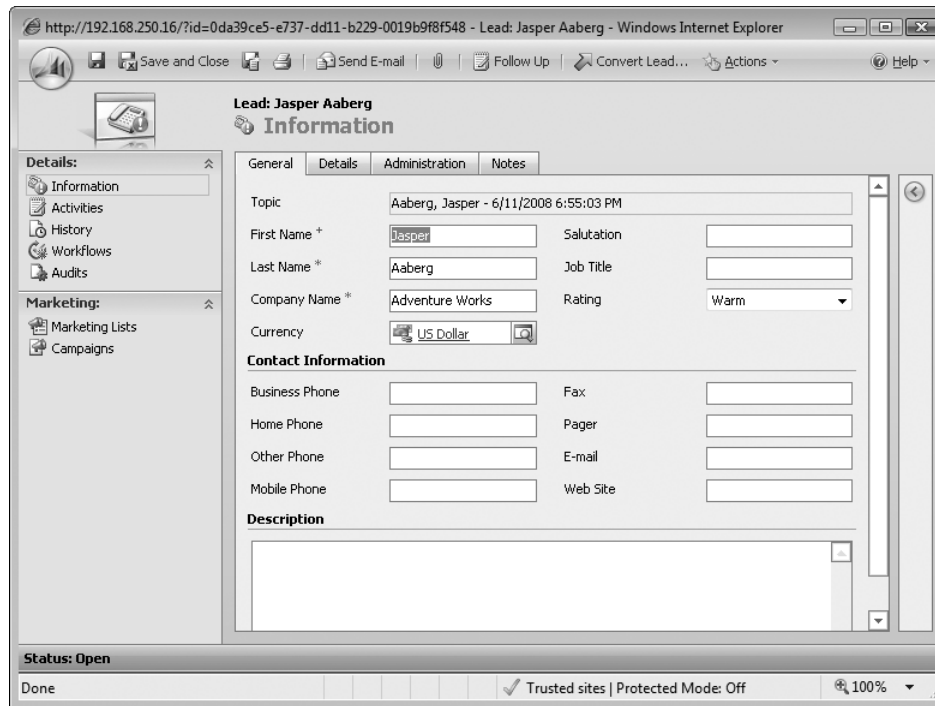


FIGURE 10-5 Updated lead topic

Offline Development Considerations

While developing for Microsoft CRM Dynamics for Outlook with Offline Access, remember to consider these factors unique to offline development:

- Not all *CrmService* messages are available for all entities while in offline mode.
- Only *Retrieve* messages are available on the *MetadataService* while in offline mode.
- Workflows do not work in offline mode.
- All offline plug-ins must be registered to run synchronously.
- Plug-ins can potentially run twice, once while offline and once on the server when going back online. You can add a check to your plug-in to avoid this scenario.
- You must also add the public key token of your plug-in assembly to the *AllowList* registry key so that it can run offline.
- You must deploy custom *.aspx* pages to each client computer. Microsoft Dynamics CRM does not automatically deploy the custom Web pages for you.

Summary

This chapter has provided you with an overview of developing for Microsoft Dynamics CRM for Outlook with Offline Access. The topics and examples in this chapter have given you good foundation knowledge and ideas for coding custom pages and plug-ins to run offline or in both online and offline mode. You should also now have a better idea of what to watch out for when architecting your offline solutions.

