



**Infor Education**  
Master Infor products.  
Maximize your potential.

# Infor CRM: v8.4 Developing Web Training Workbook

**Infor CRM**

July 10, 2018

Course code: 01\_0610840\_IEN0174\_CRM

## **Legal Notice**

Copyright © 2018 Infor. All rights reserved.

### **Important Notices**

The material contained in this publication (including any supplementary information) constitutes and contains confidential and proprietary information of Infor.

By gaining access to the attached, you acknowledge and agree that the material (including any modification, translation or adaptation of the material) and all copyright, trade secrets and all other right, title and interest therein, are the sole property of Infor and that you shall not gain right, title or interest in the material (including any modification, translation or adaptation of the material) by virtue of your review thereof other than the non-exclusive right to use the material solely in connection with and the furtherance of your license and use of software made available to your company from Infor pursuant to a separate agreement, the terms of which separate agreement shall govern your use of this material and all supplemental related materials ("Purpose").

In addition, by accessing the enclosed material, you acknowledge and agree that you are required to maintain such material in strict confidence and that your use of such material is limited to the Purpose described above. Although Infor has taken due care to ensure that the material included in this publication is accurate and complete, Infor cannot warrant that the information contained in this publication is complete, does not contain typographical or other errors, or will meet your specific requirements. As such, Infor does not assume and hereby disclaims all liability, consequential or otherwise, for any loss or damage to any person or entity which is caused by or relates to errors or omissions in this publication (including any supplementary information), whether such errors or omissions result from negligence, accident or any other cause.

Without limitation, U.S. export control laws and other applicable export and import laws govern your use of this material and you will neither export or re-export, directly or indirectly, this material nor any related materials or supplemental information in violation of such laws, or use such materials for any purpose prohibited by such laws.

### **Trademark Acknowledgements**

The word and design marks set forth herein are trademarks and/or registered trademarks of Infor and/or related affiliates and subsidiaries. All rights reserved. All other company, product, trade, or service names referenced may be registered trademarks or trademarks of their respective owners.

# Table of contents

About this workbook.....	5
<b>Course overview.....</b>	<b>7</b>
Course description and agenda.....	8
<b>Lesson 1: Project requirements.....</b>	<b>13</b>
Project requirements.....	14
Entity Relationship Diagram (ERD) .....	14
Notes on required forms.....	14
Business rules and events .....	15
Statement of Work.....	15
Check your understanding .....	18
<b>Lesson 2: Configuring the Development environment.....</b>	<b>21</b>
Configuring the Development Environment .....	22
Exercise 2.1: Configuring the development environment .....	22
Exercise 2.2: Perform dynamic customizations.....	26
Check your understanding .....	30
<b>Lesson 3: Creating a custom entity.....</b>	<b>32</b>
Creating a custom entity .....	33
Exercise 3.1: Creating a custom entity .....	33
Exercise 3.2: Creating a Detail form .....	36
Exercise 3.3: Creating an Insert page .....	39
Exercise 3.4: Creating a View page.....	41
Exercise 3.5: Inserting and viewing a project.....	43
Check your understanding .....	46
<b>Lesson 4: Workspace enhancements.....</b>	<b>48</b>
Workspace enhancements.....	49
Exercise 4.1: Enhancing the View page .....	49
Exercise 4.2: Enhancing the Navigation Bar.....	51
Exercise 4.3: Enhancing the Insert page .....	54
Exercise 4.4: Verifying the enhancements.....	55
Check your understanding .....	56
<b>Lesson 5: Adding automation to a form.....</b>	<b>57</b>
Adding Automation to a Form.....	58
Exercise 5.1: Associating an account to a project.....	58
Exercise 5.2: Creating lookups.....	62
Exercise 5.3: Adding events to automate input.....	65
Exercise 5.4: Verifying the automation changes .....	68
Check your understanding .....	70
<b>Lesson 6: Associating multiple objects to an entity.....</b>	<b>72</b>
Associating multiple objects to an entity .....	73
Exercise 6.1: Creating a many-to-many relationship.....	73
Exercise 6.2: Creating an Add and Edit form .....	78
Exercise 6.3: Creating a form to view and add ClientProjectContacts .....	83
Exercise 6.4: Creating a form to view Contact Projects.....	88
Exercise 6.5: Verifying the many-to-many association .....	91
Check your understanding .....	94
<b>Lesson 7: Modifying out-of-the-box entities .....</b>	<b>95</b>
Modifying out of the box entities.....	96
Exercise 7.1: Modifying an out-of-the-box entity .....	96
Exercise 7.2: Creating a ProjectTickets form .....	98

Exercise 7.3: Verifying the Ticket associations .....	101
Check your understanding .....	103
<b>Lesson 8: Using pick lists.....</b>	<b>104</b>
Using Pick Lists .....	105
Exercise 8.1: Creating Pick Lists .....	105
Exercise 8.2: Verifying Pick Lists.....	108
Check your understanding .....	110
<b>Lesson 9: Creating business rules .....</b>	<b>111</b>
Creating Business Rules.....	112
Exercise 9.1: Creating an external DLL with business rules.....	112
Exercise 9.2: Creating business rules for closing a project .....	117
Exercise 9.3: Creating business rules for ticket status .....	120
Exercise 9.4: Modifying the client project before updating.....	123
Check your understanding .....	126
<b>Lesson 10: Bundling a customization .....</b>	<b>127</b>
Bundling a customization .....	128
Exercise 10.1: Creating a manifest and bundle.....	128
Exercise 10.2: Installing a customization bundle.....	129
Check your understanding .....	131
<b>Lesson 11: Customizing the Customer portal.....</b>	<b>132</b>
Customer portal .....	133
Exercise 11.1: Customizing the Customer Portal.....	133
Exercise 11.2: Allowing customers to add a ticket to a project .....	135
Check your understanding .....	137
<b>Course summary .....</b>	<b>138</b>
<b>Appendices.....</b>	<b>139</b>
Appendix A: User accounts.....	140
Appendix B: Frequently asked questions.....	141
Appendix C: Incremental and full builds.....	145
Exercise C.1: Forcing a full build .....	145
Appendix D: Additional debugging techniques.....	147
Exercise D.1: Troubleshooting a build error.....	147
Exercise D.2: Debugging with Visual Studio .....	148
Appendix E: Sample project exercise .....	155
Exercise goal: .....	155
Exercise requirements: .....	155
Scenario .....	155

# About this workbook

Welcome to this Infor Education course! We hope you will find this learning experience enjoyable and instructive. This Training Workbook is designed to support the following forms of learning:

- Classroom instructor-led training
- Virtual instructor-led training
- Self-directed learning

This Training Workbook is not intended for use as a product user guide.

## Activity data

You will be asked to complete some practice exercises during this course. Step-by-step instructions are provided in this guide to assist you with completing the exercises. Where necessary, data columns are included for your reference.

Your instructor will provide more information on systems used in class, including server addresses, login IDs, and passwords.

## Self-directed learning

If you are taking this course as self-directed learning, there may be instructor-recorded presentations and/or simulations available to assist you.

If instructor-recorded presentations are available, a hyperlink to the recording will be included on the first page of each corresponding Lesson.

If simulations are available, the demos and exercises throughout this Training Workbook will include hyperlinks that allow you to view and/or practice the execution of the demo or exercise in a simulated training environment.

## Learning Libraries

Learning Libraries in Infor Campus include learning materials that are available to you online, anytime, anywhere. These materials can supplement instructor-led training, providing you with additional learning resources to support your day-to-day business tasks and activities.

Please note that if you accessed this Training Workbook directly via a Learning Library, you will not have access to the Infor Education Training Environment that is provided with all instructor-led and most self-directed learning course versions, as referenced above. Therefore, you will not be able to practice the exercises in the specific Training Environment for which the exercises in this Training Workbook were written.

## Symbols used in this workbook



Hands-on exercise  
("Exercise")



Your notes



Question



Instructor demonstration  
("Demo")



Important note



Answer



Can be used for either  
("Scenario" or "Discussion")



Critical note



Task simulation



For your reference



# Course overview

## Estimated time

.25 hours

## Learning objectives

Upon completion of this course, you'll be able to:

- Convert customer requests into project requirements.
- Identify the entities and relationships that must be included in the project and as they apply to the requirements.
- Identify interface items and business rules that must be included in the project and as they apply to the requirements.
- Categorize project tasks for customer sign-off.
- Configure a web development environment for Infor CRM.
- Describe a scenario in which you could take advantage of the no deploy process in the Application Architect.
- Describe what happens when making a change to a form using the Form Designer in the Web Administrator.
- Discuss common properties of a new entity and identify which ones are needed to create a relationship with another entity.
- Summarize the purpose of smart parts and pages.
- List components included in a main view.
- Identify what OnClickActions are available for a control button.
- Explain what happens during a full build and deploy versus the no deploy process.
- Discuss what happens when adding validation to a form control and contrast that process with adding validation to the entity itself.
- Explain the purpose of a portal.
- Discuss what happens when setting the data bindings of a control.

## Topics

- Course description and agenda

# Course description and agenda

This course teaches learners to configure and develop within an Infor CRM Application Architect development environment.

Successful learners have knowledge of the Infor CRM Web client, Visual Studio, and proficiency with VB.Net or C#.

After completing this course, learners will be able to:

- Use the Infor CRM Application Architect
- Configure an Infor CRM web development environment
- Create a new entity and smart part
- Build a main view and add validation to a form control.

## Prerequisite courses

- Infor CRM: v8.x Using Web
- Infor CRM: v8.x Administering (for customers)
- Infor CRM: v8.x Implementing (for partners)

## Course duration

32 hours

## Prerequisite knowledge

To optimize your learning experience, Infor recommends you have the following knowledge prior to attending this course:

- Knowledge of Microsoft Windows operating systems, Microsoft Windows server technology and security, and Microsoft Internet Information Services (IIS) Manager
- Knowledge of Microsoft SQL Server and relational databases
- A working knowledge of VB.NET and/or C#
- A working knowledge of Microsoft Visual Studio
- Experience using the Infor CRM Windows and Web Clients

## Audience

- Customer User
- Business Consultant
- Technical Consultant
- Support

## System requirements

Infor CRM Training Environment

## Reference materials

Infor CRM reference materials are available from the following locations:

## 8 Course overview



- Infor CRM Help menu
- Infor Xtreme®

## Course agenda

The agenda below details the contents of this course, including lesson-level learning objectives and supporting objectives.

Lesson	Lesson Title	Learning Objectives	Day
1	Project requirements	<ul style="list-style-type: none"> <li>• Convert customer requests into project requirements.</li> <li>• Identify the entities and relationships that must be included in the project and as they apply to the requirements.</li> <li>• Identify interface items and business rules that must be included in the project and as they apply to the requirements.</li> <li>• Categorize project tasks for customer sign-off.</li> </ul>	1
2	Configuring the development environment	<ul style="list-style-type: none"> <li>• Export the Virtual File System (VFS) to a Local Project Workspace.</li> <li>• Deploy the required Core Portals.</li> <li>• Explain the no-deploy feature</li> <li>• Create and Restore Project Backups</li> </ul>	1
3	Creating a custom entity	<ul style="list-style-type: none"> <li>• Discuss common properties of a new entity</li> <li>• Identify which properties are needed to create a relationship with another entity.</li> <li>• Create a form to show details of an entity</li> <li>• Create a page to view a form</li> <li>• Summarize the purpose of smart parts and pages.</li> </ul>	1
4	Enhancing the view page	<ul style="list-style-type: none"> <li>• Customize a main view page to mimic out of the box features.</li> <li>• Add context menus to navigation items</li> <li>• Redirect to the detail page after inserting an object.</li> </ul>	1
5	Adding automation to a form	<ul style="list-style-type: none"> <li>• Create a form to show related entities in their parent's main view.</li> <li>• Create a lookup to easily address relationships.</li> </ul>	2

		<ul style="list-style-type: none"> <li>• Create automation to enhance security.</li> </ul>	
6	Associating multiple objects to an entity	<ul style="list-style-type: none"> <li>• Create an Entity to allow many to many relationships</li> <li>• Create a form to allow for adding and editing data</li> <li>• Add code snippets to events to automate some data entry</li> <li>• Create forms to see the relationships</li> </ul>	2
7	Modifying out of the box entities	<ul style="list-style-type: none"> <li>• Modify an out of the box Infor CRM Entity.</li> <li>• Modify an out of the box detail form</li> </ul>	2
8	Using pick lists	<ul style="list-style-type: none"> <li>• Create a custom pick list</li> <li>• Apply a custom pick list to a form</li> </ul>	2
9	Creating business rules	<ul style="list-style-type: none"> <li>• Create a project in Visual Studio to hold external assemblies.</li> <li>• Add an external assembly into Application Architect.</li> <li>• Create business rules to use external assemblies.</li> </ul>	3
10	Bundling a customization	<ul style="list-style-type: none"> <li>• Bundle a customization</li> <li>• Install a customization bundle</li> </ul>	3
11	Feature requests	<ul style="list-style-type: none"> <li>• Deploy the Customer Portal.</li> <li>• Customize the Customer Portal interface.</li> </ul>	3
Course summary		Debrief course.	3

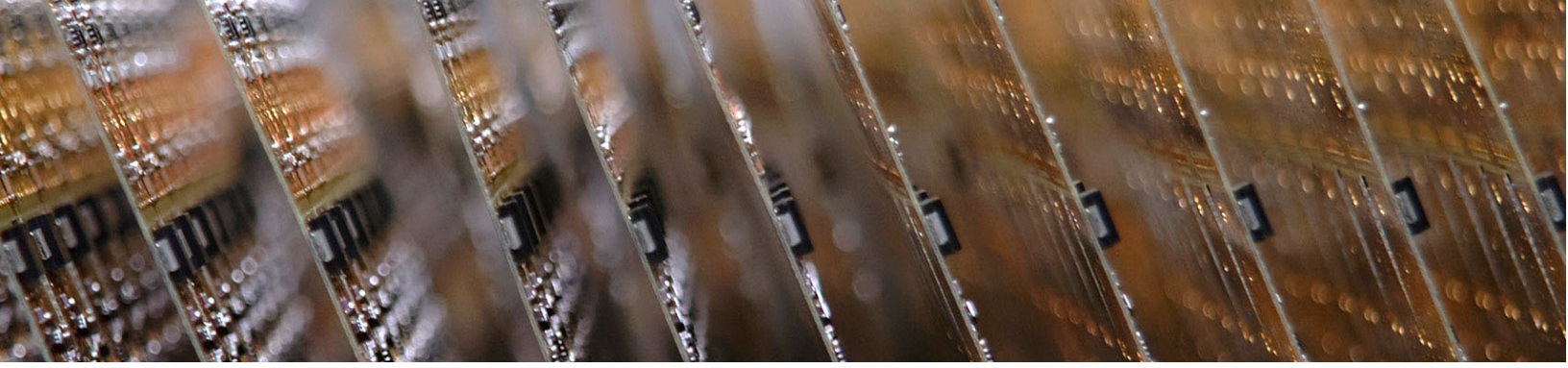
## Appendices

This section contains information that is not part of the instructional content of this course, but provides additional related reference information.

Appendix	Appendix Title	Content Description
Appendix A	Student user accounts	This appendix provides a reference for student login credentials.
Appendix B	Frequently Asked Questions	Reviews common questions in addition to topics not specifically addressed in this course.

Appendix C	Incremental and full builds	Describes the differences between incremental and full builds, and how to utilize both.
Appendix D	Additional debugging techniques	Reviews debugging techniques for build errors within Application Architect, C# Snippet Action Items, and External Assemblies
Appendix E	Sample project exercises	This appendix provides a sample project exercise which utilizes a set of requirements that are based on a typical client's needs.

## 12 Course overview



# Lesson 1: Project requirements

## Estimated time

1 hour

## Learning objectives

In this lesson, you will:

- Convert customer requests into project requirements.
- Identify the entities and relationships that must be included in the project and as they apply to the requirements.
- Identify interface items and business rules that must be included in the project and as they apply to the requirements.
- Categorize project tasks for customer sign-off.

## Topics

- Discuss requirements for the project completed during this course.
- Visualize the project with an entity relationship diagram.
- Visualize the forms and pages with wireframe images.
- Discuss potential business rules that will be needed.
- View a statement of work for the project.

# Project requirements

You're a new Infor CRM Developer. Your company's Business Analyst just returned from a meeting with a customer (Phoenix Computers) to discuss potential Infor CRM customizations. Here's what you found out from the interview:

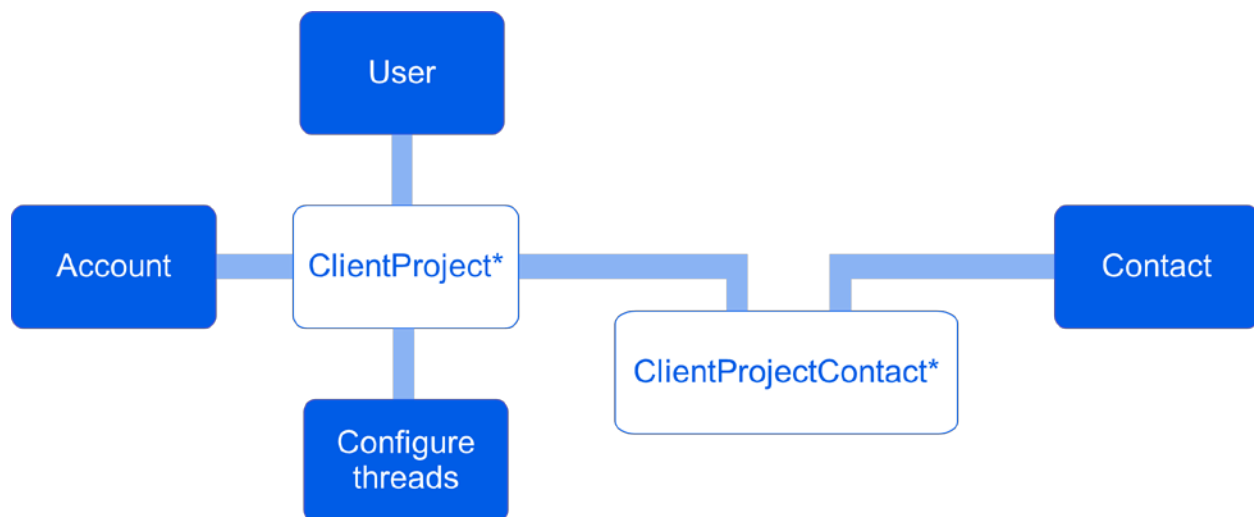
- Phoenix Computer's provides project management services to almost all accounts and they need a way to track those projects inside Infor CRM.
- There's always at least one contact who deals with each project and most of the time there is more than one.
- Phoenix Computer's wants an easy way to frequently update project-related tasks.
  - These tasks are different than activities
  - Tasks can be initiated by their customers
- Before closing a project, all tasks in that project must also be closed.

Next, the Business Analyst promises to sketch out an entity relationship diagram (ERD), provide notes on required forms and business rules, and provide a copy of the Statement of Work given to the client.

Review the notes and sketches to make sure you understand their requests accurately.

## Entity Relationship Diagram (ERD)

The Business Analyst provided the ERD sketched below. New entities are marked with an asterisk.



## Notes on required forms

The customer, Phoenix Computers, has employees working primarily as Project Managers. To assist these employees, a new navigation bar should be added to the Infor CRM Web Client that has links to ClientProjects, Accounts, Contacts, and Tickets.

A new main view will be required for the new ClientProject entity. This main view should include tab controls to view associated Contacts and Tickets and can add new Contacts and Tickets associated with the project.

The new entity, ClientProjects, should have a menu item under the **New** menu that links to an **Insert** page for ClientProjects.

A tab should be created for out-of-the-box entities (Contact and Account) that displays any associated project. This tab should also include a button to add a new ClientProject.

The out-of-the-box Tickets detail page must be updated to include a lookup to the new ClientProject entity.

Wherever possible, Phoenix Computers would like any customization to mimic Infor CRM's default objects. To do this, add redirects, context menus, common tasks, and filters where possible.

## Business rules and events

The customer, Phoenix Computers, wants to ensure a client's project can only be closed or cancelled if there are no open or in progress tickets associated with it. Your Business Analyst determines two business rules are required to complete this task.

1. The first business rule checks if a client project has a status of Closed or Cancelled. This rule is named **IsProjectStatusClosed**.
2. The second business rule checks if any tickets associated with a client project have a status of Open or In Progress. This rule is named **CanCloseProject**.

These rules are used at two times:

1. When the status control is changed on a Client Project object or
2. When **Save** is attempted

If the ClientProject object does not meet the criteria to close, the rules will not allow that to occur.

## Statement of Work

The Business Analyst also completed a Statement of Work which helps itemize each task in the customization project. See how to use this list to budget the number of hours for each task and submit back to the customer for a quote:

New Schema	
<b>ClientProject entity and table</b>	<b>New entity to store project data</b> Properties: ClientProjectId, AccountId, CreateUser, CreateDate, ModifyUser, ModifyDate, StartDate, EndDate, Description, EstimatedCost, ManagerId, Title
<b>ClientProjectsContacts entity and table</b>	<b>New entity for relationship to Contact and ClientProject entities</b> Properties: ClientProjectsContactsId, CreateUser, CreateDate, ModifyUser, ModifyDate, ContactId, ClientProjectId, Role
New Relationships	
<b>Account → ClientProject</b>	<b>Account related to ClientProject</b> Account.AccountId 1:M ClientProject.AccountId
<b>ClientProject ↔ Contacts</b>	<b>ClientProject related to Contact</b> ClientProject.ClientProjectId 1:M ClientProjectContact.ClientProjectId Contact.ContactId 1:M ClientProjectsContact.ContactId

<b>ClientProject → Ticket</b>	<b>ClientProject related to Ticket</b> ClientProject.ClientProjectId 1:M Ticket.ProjectId
<b>ClientProject ← User</b>	<b>ClientProject related to User</b> ClientProject.ManagerId M:1 User.UserId
<b>New Forms</b>	
<b>ProjectDetails</b>	<b>Quick form to show project details</b> Controls: Title, EstimatedCost, Description, StartDate, EndDate, Account (lookup), ProjectManager (lookup), Status (picklist)
<b>AccountProjects</b>	<b>Quick form with a data grid to list projects associated to an account</b> Columns: Title, Description, StartDate, EndDate
<b>ProjectContacts</b>	<b>Quick form with a data grid to list contacts associated with a project</b> Columns: Name, Account, Role
<b>AddEditProjectContact</b>	<b>Quick form to add or edit project contacts</b> Controls: Project (lookup), Contact (lookup), Role
<b>ContactProjects</b>	<b>Quick form with a data grid to list projects associated to a contact</b> Columns: Title, ProjectManager, StartDate, EndDate
<b>ProjectTickets</b>	<b>Quick form with a data grid to list tickets associated with a project</b> Columns: TicketNumber, AssignedDate, NeededbyDate, Urgency, TicketStatus
<b>Modified Forms</b>	
<b>TicketDetails</b>	Add a Project lookup
<b>New Pages (List Smart Parts and Their Display Modes)</b>	
<b>Project Detail Page</b>	LiveGroupViewer, ProjectDetails (MainContent / Detail), ProjectContacts (TabControl / Detail), AddEditProjectContact (DialogWorkspace), ProjectTickets (TabControl / Detail)
<b>Insert Project Page</b>	Add ProjectDetails (MainContent / Insert)
<b>Modified Pages (List Smart Parts and Their Display Modes)</b>	
<b>Account Detail Page</b>	AccountProjects (TabControl / Detail)
<b>Contact Detail Page</b>	ContactProjects (TabControl / Detail) EditProjectContact (DialogWorkspace)
<b>Navigation / Menu Items</b>	
<b>“New” Menu</b>	New → Project



<b>“Projects” Nav Bar Group</b>	Include Nav items for Projects, Accounts, Contacts, Tickets
<b>New Business Rules/Events</b>	
<b>IsProjectStatusClosed</b>	Add a method to the OnClick action of the Project Status pick list on the ProjectDetails form. This action checks if status was set to Closed or Cancelled.
<b>CanCloseProject</b>	Add a method to the OnSuccess action of the OnClick action for Project Status. This action checks if all project tickets are closed. If not all tickets are closed, warn the user that their selection is invalid.
<b>OnBeforeUpdate</b>	Add a step to the OnBeforeUpdate event for the ClientProject entity. This event checks if the Project Status pick list was changed. If yes, then it implements IsProjectStatusClosed and CanCloseProject methods to prevent a user from committing a save if not all project tickets are closed.

## Check your understanding



How do I convert customer requests into project requirements?



---

---

---

---

---

---

---

---



What required entities and relationships must be included in a project?



---

---

---

---

---

---

---

---



Which entities and relationships must be included in a project and as they apply to the requirements?



---

---

---

---

---

---

---

---



Which interface items and business rules must be included in the project and as they apply to the requirements?



---

---

---

---

---

---

---

---



How can I categorize project tasks for customer sign-off?



---

---

---

---

---

---

---

---



# Lesson 2: Configuring the Development environment

## Estimated time

1 hour

## Learning objectives

In this lesson, you will:

- Export the Virtual File System (VFS) to a Local Project Workspace.
- Deploy the required Core Portals.
- Explain the no-deploy feature.
- Create and Restore Project Backups.

## Topics

- Export a project workspace
- Backup a project workspace
- Restore a project workspace
- Deploy core portals
- Explore the no-deploy feature
- Use the form designer

# Configuring the Development Environment

Before you begin any customization or development project within Infor CRM's Web Client, you'll need a development environment to work in.

Project workspaces can be exported from the VFS into a local file system which allow you to use files located on a local drive, instead of using the files stored within the database. This can improve performance and is especially noticeable while doing customization builds.

Backups are strongly suggested both before you begin any development work and during the development life-cycle so you may revert to previous versions if needed.

You must deploy the Core Portals within Infor CRM as well so you can test your changes within an Infor CRM Web Client.



## Exercise 2.1: Configuring the development environment

In this exercise, you will configure the training workstation by exporting a project workspace to a local file system, learning to backup and restore a project workspace, and deploying core portals.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps



Verify you are logged in to the Training Desktop. If not, log in following instructions provided by your course instructor.

**Note:** If you are taking this course as self-directed learning, follow the instructions on the course Lab On Demand screen.

### Part 1: Export a project workspace

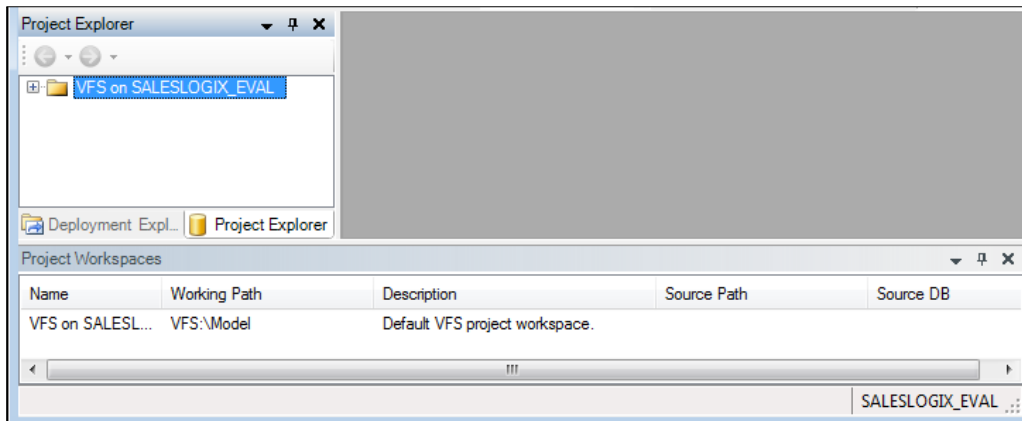
1. Select **Start > All Programs > Saleslogix > Application Architect**. The **Please log on window** opens.



Because you will be using Application Architect often during this course, Infor CRM suggests using the **Pin to Start** option for easy access. To do so, right-click **Application Architect** and select **Pin to Start**.

2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Select **View > Project Explorer** to open the **Project Explorer**.

5. Select **View > Project Workspace Manager** to open the **Project Workspace Manager**.



Working from an exported project workspace provides better performance than working from the Virtual File System (VFS) when you are running builds and deploys frequently in a development environment.

6. Right-click in the **Project Workspace** and select **Add**. The **Add Project Workspace** window opens.
7. Type *LFS* in the **Name** field.
8. Click the **ellipsis** near the **Working Path** field.
9. Create a new folder called **VFSLocal** on the **C-Drive**.
10. Click **OK**.
11. Type *Local Copy of VFS Files* in the **Description** field.
12. Select the **Export Files Upon Creation** check box.
13. Click **Create**.

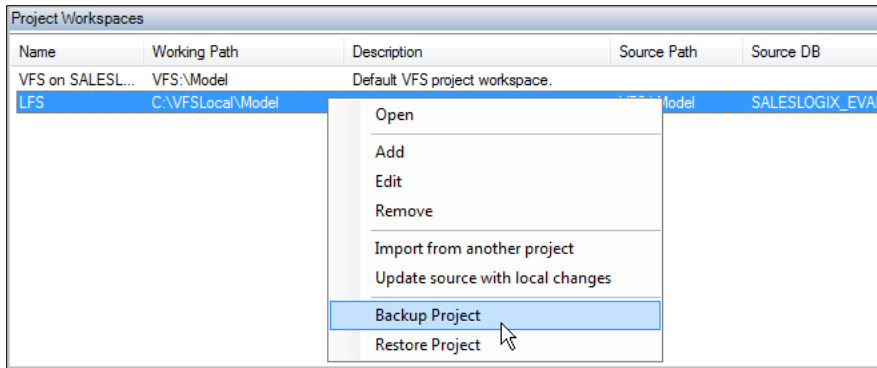
Application Architect begins exporting the project. This process may take a few minutes. It must recreate the contents of the Virtual File System in the database and save the files to the C:\VFSLocal location.

When the export completes, click the LFS workspace. **Note:** This workspace should be set as the default. Notice you're working inside the LFS workspace from within the **Project Explorer**.

## Part 2: Backup/restore a project workspace

Creating a backup takes the model from the workspace (either exported or the VFS) and puts it in a zip file on the hard drive. In the VFS (as opposed to a model exported on the hard drive), you can back up the database as another restore option. **Note:** Using a project backup is easier since you don't have to back up test data.

1. Select **View > Project Workspace Manager** to open the **Project Workspace Manager**.
2. Right-click the **LFS** project workspace and select **Backup Project**. The **Backup Project** window opens.



3. Click the **ellipsis** near the **Project Backup FileName** field. The **Set Project Backup FileName** window opens.
4. Browse to the **C:\VFSLocal** folder.
5. Type *LFS01.backup.zip* in the **File name** field.
6. Click **Save**.
7. Type *Clean Project Workspace* in the **Backup Description** field.
8. Click **OK**.

The backup process starts. When the backup is complete, let's verify the backup worked by starting a restore.



Don't restore now. You haven't changed anything since the backup. Keep these restore steps in mind if you run into issues during the course.

9. Right-click the **LFS** project workspace and select **Restore Project**. The **Select Project Backup File to Restore From** window opens.
10. Verify the backup is listed in the directory as **C:\VFSLocal**.
11. Click **Cancel** as no changes were made.



To restore, select the file and then click **OK**. Otherwise, click **Cancel**.

### Part 3: Deploy core portals

The Infor CRM Web Client uses the **Core Portals** deployment. Primarily, the **SLXClient** deployment is used so let's set that as the startup portal. This means that when you click **Run**, the default web browser opens automatically to the **SLXClient** portal after the build and deploy processes complete. The Core Portals must also be set as the deployment to use for debugging.

1. Select **View > Deployment Explorer** to open the **Deployment**.
2. Expand the **Deployments** node and then double-click **Core Portals**.
3. Select the **Use this deployment for debugging** check box.
4. Verify the following settings in the **IIS Target Settings**:

## 24 Lesson 2: Configuring the Development environment



- **Server:** SRV10 (matches the computer name)
- **Port:** 3333
- **App Pool:** InforCRM



By default, when you install the Web Host on IIS from your installation media, an Application Pool is created under the name “Saleslogix”. On this training environment, an identical Application Pool was created called “InforCRM.” Any new user or IIS Admin will then know the Application Pool is used by Infor CRM.

5. Click the **SlxClient** tab within the **IIS Portal Configuration** area.
6. Select the **Startup Portal** check box.
7. Click **Save All**.
8. Click **Run**. The **Infor CRM Web Log On** screen opens once the output is complete.

This process automatically includes three steps:

- a. **Builds the Web Platform** - Builds all the interface items and compiles all quick forms into .ascx files. These files are saved in a temporary location on the hard drive here:  
C:\Users\Administrator\AppData\Roaming\Sage\Platform\Output.
- b. **Deploys the core portals** - Builds the .aspx Web pages for the site and moves them to the designated deployment directory on the Web Server. In this case, you’re developing on the Web Server. The deployment directory is C:\inetpub\wwwroot\slxclient.
- c. **Opens the startup portal** - Opens a Web browser directed at the Infor CRM Web Log On page. The URL for the Web Client is http://SRV10:3333/slxcclient.

**TIP:** Minimize **Application Architect** when the browser launches.

9. Type *admin* in the **Username** field. No password is required.
10. Click **Sign In**.
11. Explore the client interface. Getting familiar with the core product features helps determine how to mimic your customizations off existing functionality.



## Exercise 2.2: Perform dynamic customizations

In this exercise, you will explore the “no deploy” and Form Designer features.

Let’s begin by exploring the no-deploy feature in Application Architect.

After making changes in Application Architect, build the web platform. This compiles all interface items and converts new quick forms into .ascx files. These files are then saved in a temporary location on the hard drive

(C:\Users\Administrator\AppData\Roaming\Sage\Platform\Output). When deploying the appropriate portal, App architect builds the .aspx web pages for the site and stores them in the designated deployment directory on the Web Server (generally C:\inetpub\wwwroot\slxclient).

To minimize this development time, a no-deploy feature is available when you modify and save any quick form after it has been built once. This allows you to work directly with the virtual file system or exported project workspace, meaning you can see the change in the Web Client without having to do a full build and deploy of the site. Notice how the deployment path and file name for this smart part appears in the Output window as:

\deployment\sites\SlxClient\SmartParts\[Entity]\[SmartPart].ascx. This path references either the virtual file system or the exported file system you’re using. Browse to this file in Application Architect using the Virtual File System Explorer or Windows File Explorer if using an exported file system. Let’s see how this works by making a simple change.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Explore the no-deploy feature in Application Architect

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Select **View > Project Explorer** to open the **Project Explorer**.
5. Expand **LFS > Entity Model > Packages > Saleslogix Application Entities > Contact > Forms**.

**Tip:** You can also press **CTRL+T** to search for an item within the **Project Explorer**.

6. Double-click **ContactDetails** to open the form in the workspace.
7. Click the **Asst: (txtAssistant)** control.

Quickform Details	
Name:	Prefix, FirstName, MiddleName
Work:	WorkPhone
Account:	Account
Fax:	Fax
Title:	Title
Mobile:	Mobile
Asst:	Assistant
Home:	HomePhone
Dear:	Salutation
Other:	OtherPhone

8. Select **View > Properties**. The **Properties** pane opens on the right-hand side of the screen.
9. Type *Assistant* in the **Caption** field and remove the abbreviation.
10. Press **Enter**.

Properties

txtAssistant TextBox

Appearance

Caption: Assistant:

Control Id: txtAssistant

Data Bindings: (Collection)

Enabled: True

Hot Key:

Lines: 1

MaxLength: 64

Read Only: False

Style Scheme:

TabIndex: 0

11. Click **Save All**.

The **Output Window** contains the following message:

Quick Form 'ContactDetails' built as  
 \deployment\sites\SlxClient\SmartParts\Contact\ContactDetails.ascx

12. Open the **Infor CRM Web Client**.
13. Type *admin* in the **Username** field. No password is required.
14. Click **Sign In**.
15. Click the **Sales Nav Bar** and select **Contacts**.
16. Click **Abbott, John** to open John Abbott's **Contact Detail** view.

Notice the caption change appears without having to build and deploy. All users (not just admin) can see this change if done in the production environment but it does require a browser refresh.

Name:	John Abbott	Work:	(312) 555-7854	Primary Contact for Acct.	<input checked="" type="checkbox"/>
Account:	Abbott Ltd.	Fax:	(312) 555-7545	Authorized Service Contact	<input checked="" type="checkbox"/>
Title:	President	Mobile:	(312) 555-1234	Do not Solicit	<input type="checkbox"/>
Assistant:	Ms. Jane Smith	Home:	(312) 555-3543	Do not Email	<input type="checkbox"/>
Dear:		Other:		Do not Call	<input type="checkbox"/>
Address:	4206 W. Grand Avenue Suite 900 Chicago, IL 60651 USA	Preferred Contact:	Phone	Do not Mail	<input type="checkbox"/>
		E-mail:	jabbot@abb...	Do not Fax	<input type="checkbox"/>
		Web URL:	www.abbott.c...		

## Part 2: Use the Form Designer

Infor CRM Web Client users (non-developers) using Form Designer, who have administrative access, can change captions and move items on a form by dragging-and-dropping the controls into different positions. They can also add controls to a form from an existing property. Changes made to a form affect everyone in the production environment as they're saved within the Virtual File System Manager (VFS).

Let's log on to the Infor CRM Web client and make a sample change to a form to see what happens to a project workspace when a customer uses dynamic customizations.

1. Open the **Abbott Ltd. Account Detail** view.
2. Right-click **Edit Form** and select **Open link in new tab**. The **Form Designer – AccountDetails** screen opens.



The Form Designer opens in a new window so you can easily toggle back and forth to see your changes. You can also open the form designer by using the **Form Designer** button in the **Administration Nav Bar**. Anyone with the appropriate role can access the **Edit Form** button or **Form Designer Nav Bar**.

3. Right-click the **Desc. of Business** cell and then select **Insert Row > Below**.

**Note:** You don't have to create a new row; one is created automatically when you drag a new control on the form. However, it's easier to visualize your steps this way. You can also right-click on a gray square to the left of each row or above a column for more options. The maximum number of columns for a form is six (6).

4. Drag-and-drop the **Territory** field from the list of fields on the right into the empty cell below **Desc. Of Business**.



As you drag-and-drop a field, you will see a green or red highlight. The green highlight indicates that you can drop the field in this location while red means you cannot.

Account	AccountName	Main	MainPhone	Type	Type
Division	Division	Fax	Fax	Sub Type	Sub Type
Address		Toll Free	TollFree	Status	Status
		Web URL	WebAddress	Industry	Industry
		Owner	Owner	Desc. of Business	BusinessDescri...
lueParentAccount	Acc Manager	AccountManager			
				Territory	Erp Territory

5. Click **Save**.
6. Toggle to the other browser with the **Account Detail** view open for **Abbott Ltd**.
7. Press **F5** to refresh the page.

The new field should appear on the form. Had a user made this change in the production environment, all other Infor CRM users in the system would immediately see this change the next time they logged on or refreshed the browser.

8. Open **Windows File Explorer** and navigate to **C:\VFSLocal\Model\deployment\sites\SixClient\SmartParts\Account**. The **AccountDetails.ascx** smart part and related files are viewable here.



Depending on the active project workspace, these files may reside in a different location. In this training, your files are in **C:\VFSLocal** as you're using the LFS workspace you created in this location. If you were still using the VFS on the Saleslogix workspace, these files could be seen in the **Project Explorer** by selecting **Virtual File System Explorer > Model > deployment > sites > SixClient > SmartParts > Account**.

## Check your understanding



How do I configure a web development environment for Saleslogix?



---

---

---

---

---

---

---

---



In what scenario could you take advantage of the no deploy process in Application Architect?



---

---

---

---

---

---

---

---



What happens when a change is made to a form using the Form Designer in the Web Administrator?



---

---

---

---

---

---

---

---



What happens during a full build and deploy versus a no deploy process?



---

---

---

---

---

---

---

---



# Lesson 3: Creating a custom entity

## Estimated time

3 hours

## Learning objectives

In this lesson, you will:

- Discuss common properties of a new entity.
- Identify which properties are needed to create a relationship with another entity.
- Create a form to show details of an entity.
- Create a page to view a form.
- Summarize the purpose of smart parts and pages.

## Topics

- Create the ClientProject entity and table
- Create a relationship between ClientProject and Account
- Add a ProjectDetails form
- Create an InsertProject page
- Create a Project option on the *New* menu
- Create a Main View for ClientProject
- Create a Project Management navigation bar
- Create an All Projects group



# Creating a custom entity

One of the most common customizations within Infor CRM is the creation of a new entity. This allows you to create an entity with the specific fields, events, relationships, and other properties needed for your client.

Your customer, Phoenix Computers, is requiring the creation of a new Entity to handle their Projects. However, Infor CRM already has a Projects entity within it so they have agreed to use ClientProjects as the entity name.



## Exercise 3.1: Creating a custom entity

In this exercise, you will create a new custom entity, and add a relationship to an existing Infor CRM entity.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Create a ClientProject entity & table

Before creating a new entity, you'll create a new package in Application Architect. Although it's not required to separate your customizations from those listed under Infor CRM Application Entities, it's a good practice if you want to easily keep track of new content you create for a custom entity.

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Open the **Project Explorer**.
5. Expand **LFS > Entity Model > Packages**. **Note:** Several out-of-the-box packages are listed.
6. Right-click the **Packages** node and select **New Package**. The **New Package** screen opens.
7. Type *Training* in the **Name** field and then click **Save All**.
8. Right-click the **Training** node and select **New Entity**. The **New Entity Wizard** opens.
9. Select the **Create a business entity and a new table** radio button.
10. Click **Next**. The **New Entity From New Table** screen opens.
11. Type *ClientProject* in the **Display Name** field and then press **Tab** to populate the remaining fields.
12. Select the **Enable Field-Level Security** check box.

13. Click **Next**. The **Primary Relationship** screen opens.

14. Clear the **Relate to an existing entity** check box.

You know from the Statement of Work that your ClientProject entity has relationships with several entities (Account, Contact, Ticket, User). This screen allows you to create just one. You can create the Account relationship here but the ClientProject table will be nested under the Account table if viewed in the Database Manager. Let's skip the relationship creation for now and return to it later.

15. Click **Next**. The **Enter Properties** screen opens.

This screen contains the properties for the ClientProject, such as project title, start date, etc. Some properties, such as the primary key are created for you.

The CreateDate, ModifyDate, CreateUser, and ModifyUser properties are included with every entity or table to support synchronization with Remote Clients. If you choose to enable field security, the Seccodeid property is included after the wizard is finished.

16. Click **Add Property** or press **Alt+A** to add the following properties:

	DisplayName	Property Name	Column Name	DataType	Length	
	AccountID	AccountID	ACCOUNTID	Standard Id	12	
	ManagerID	ManagerID	MANAGERID	Standard Id	12	
	Title	Title	TITLE	Text	64	
	Description	Description	DESCRIPTION	Text	255	
	StartDate	StartDate	STARTDATE	DateTime	0	
	EndDate	EndDate	ENDDATE	DateTime	0	
	EstimatedCost	EstimatedCost	ESTIMATEDCOST	Text	64	

17. Click **Next**. The **Additional Options** screen opens.

18. Select **Title** from the **Display Property** drop-down list. The selected **Display Property** is what appears on the title bar of the main view in the Web Client.



If you would like to have an icon used for this entity, you can select the image to use for both a small icon and large icon by selecting an image from the **Small Image** and **Large Image** options.

19. Click **Next**. The **Create Entities** screen opens.

20. Click **Next** again.


21. Click **Finish** when the process finishes. The ClientProject entity appears in the workspace with all properties listed.

22. Click the **Entity** tab.

23. Type *Client Project* in the **Display Name** field.

24. Type *Client Projects* in the **Plural Display Name** fields.

25. Clear the **Include** check box next to the **ClientProjectId** property.

Entity Properties								
	Property Name	Display Name	Data Type	Include	Nullable	Indexed	Audited	Dynamic
	AccountID	AccountID	<a href="#">Standard Id</a>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ClientProjectId	Id	<a href="#">Standard Id</a>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	CreateDate	Create Date	<a href="#">DateTime</a>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	CreateUser	Create User	<a href="#">Standard Id</a>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	Description	Description	<a href="#">Text</a>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	EndDate	EndDate	<a href="#">DateTime</a>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	EstimatedCost	EstimatedCost	<a href="#">Decimal</a>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ManagerID	ManagerID	<a href="#">Standard Id</a>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ModifyDate	Modify Date	<a href="#">DateTime</a>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ModifyUser	Modify User	<a href="#">Standard Id</a>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	SeccodeId	SeccodeID	<a href="#">Standard Id</a>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	StartDate	StartDate	<a href="#">DateTime</a>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	Title	Title	<a href="#">Text</a>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

26. Click **Save All**.

27. Click **Build Interfaces**.

Let's choose to omit the ClientProjectId (Id) from the entity properties since all entities already have an underlying Id property. Clearing this extra ClientProjectId property on the ClientProject entity means you not only eliminate ambiguity that may result from having two properties that refer to the same physical field in the database, but relationships are made more clear.

For example, when creating a relationship from Account to ClientProject, you'll get a new relationship property on Account called ClientProjects (notice the 's' because an Account can have more than one ClientProject). Later in your customization, you'll bind the related ClientProjects through that relationship. If you choose to include the ClientProjectId and the Id on the ClientProject entity, the Account might have a binding of Account.ClientProjects.ClientProjectId OR Account.ClientProjects.Id. Both of those examples are accurate but it's ambiguous and some would argue the latter is clearer.

## Part 2: Create a new ClientProject relationship to the Account entity

Each Infor CRM account can have many client projects. You must create a new relationship using your AccountId property from our ClientProject entity and the AccountId from the existing Account entity.

1. Expand **LFS > Entity Model > Packages > Parent Properties**.
2. Right-click the **Parent Properties** node and select **New Relationship**. The **New Client Project Relationship** window opens.
3. Set the following relationship properties:
  - **Parent Entity:** Account
  - **Parent Entity Property:** Account ID
  - **Cardinality:** 1:M
  - **Child Entity:** Client Project
  - **Child Entity Property:** AccountID
4. Click **OK**. The details of the relationship appear in the workspace.

5. Click **Save All**.
6. Click **Build Interfaces**.

Once the build is complete, you may begin to use the relationship between Client Project and Account.



### Exercise 3.2: Creating a Detail form

In this exercise, you will create a form to show the details of a ClientProject object, which can also be used to edit or insert information.

#### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Add a ProjectDetails form

This form displays the details for a project. Build this form once and reuse it on both the Project Details page and the Insert Project page (both of which will be created).

1. Open **Application Architect**.
2. Open the **Project Explorer**.
3. Expand **LFS > Entity Model > Packages > Training > ClientProject**.
4. Right-click **Forms** and select **New Form Wizard**. The **New Form Wizard** window opens.
5. Select the **Create Detail Form** radio button.
6. Type *ProjectDetails* in the **Form Name** field.
7. Click **Next**. The **Properties** screen opens.
8. Select the following properties to display on the form:
  - AccountID
  - Description
  - EndDate
  - EstimatedCost (Estimated Cost)
  - ManagerID
  - StartDate
  - Title
9. Click **Next**. The **Details Form Layout** screen opens.
10. Verify **3** is selected in the suggested **Number of columns** field.
11. Click **Next**. The **Summary** screen opens.
12. Click **Finish**. The form opens in the workspace.

**Note:** You can rearrange the controls on the form by dragging-and-dropping the different cells so they resemble the following figure:

Quickform Details <a href="#">Click to modify form resources</a>					
Title	Title	Description	Description	StartDate	StartDate
Estimated Cost	\$1000.00 (EstimatedCost)	ManagerID	ManagerID	EndDate	EndDate
		AccountID	AccountID		

Notice the area below the form for Non-Visual Controls. You'll explore this area later. It contains items such as Data Source (Hql, Mashup, SData) and Hidden Text.

13. Click **Save All**.

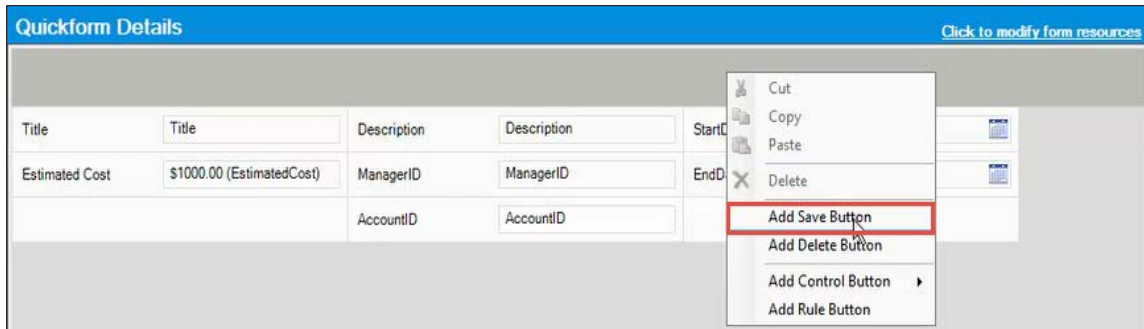
## Part 2: Add a Save button to the ProjectDetails form

There are two ways to add a button to a form:

- Through the quick Form Designer
- Through the form's Toolbar property

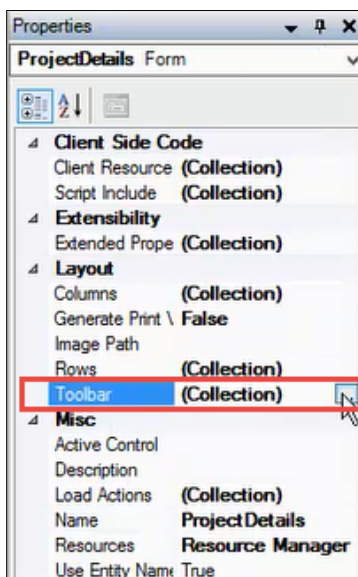
Let's explore both methods.

1. Right-click inside the top gray area of the **ProjectDetails** form and select **Add Save Button**.



A new **Save** button appears on the form. The button automatically includes an image and the built-in **Save** business rule as part of its On Click Action. To set additional properties, such as the tooltip or refresh value, use the **Toolbar** property on the form. Let's do this now.

2. Select **View > Properties**. The **Properties** pane opens on the right side of the screen.
3. Click the **ellipsis** near the **Toolbar** property. The **Toolbar Editor** window opens.



The button you just added is listed as a sub-node under either the **Left**, **Center**, or **Right** node. This depends on where you right-clicked when adding the **Save** button. Right-click any of these nodes to add additional buttons from the **Toolbar Editor**.

### Part 3: Remove this button and manually recreate it

1. Expand **Center** and then select **tbrSave (QFButton)**.
2. Click **Remove**.
3. Select **Right**.
4. Select **Add > Button**.
5. Select **Icon** from the **Button Type** drop-down list.
6. Type **tbrSave** in the **Caption** field.
7. Click **Image** and then click the **ellipsis**.
8. Double-click **Save\_16x16**.

9. Expand the **On Click Action** property.
10. Click **Action Name** and then click the **ellipsis**. The **Action Item Designer** dialog box opens.
11. Select **Business Rule**. The business rule options open on the right side of the dialog box.
12. Select **Save** from the **Business Rule** drop-down list.
13. Click **OK** to close the **Action Item Designer** dialog box.
14. Click **OK** again to close the **Toolbar Editor** window.
15. Click **Save All**.
16. Click **Build Web Platform**.

Until this point you have stayed inside the Entity Model. You've created a **ProjectDetails** form which is compiled as an **.ascx** file during the **Web Platform** build. **Note:** This **.ascx** file is also known as a smart part or a user control. However, this **.ascx** file can't be displayed on its own. If you deployed the SLXClient portal now and tried to navigate to this file inside a browser, there would be nothing to see because it requires a page (**.aspx** file) on which to show itself. It also needs a menu item to launch the page.



### Exercise 3.3: Creating an Insert page

In this exercise, you will create a new page for ClientProjects, that can be used to insert new projects to the database, and add Smart Parts to the page.

#### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

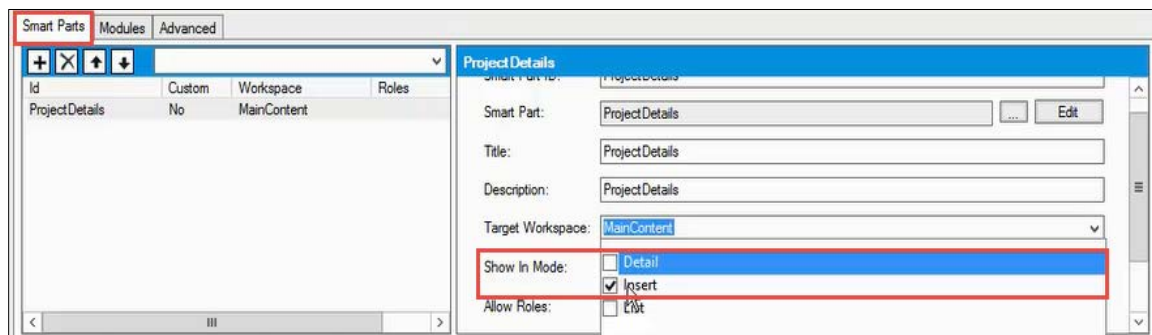
## Exercise steps

### Part 1: Create an Insert Project page

1. Open **Application Architect**.
2. Open the **Project Explorer**.
3. Expand **LFS > Portal Manager > SLXClient > Pages**.
4. Right-click the **Pages** node and select **New Page Wizard**. The **Portal Page Wizard** opens.
5. Select **Entity Page** from the **Page Types** drop-down list.
6. Click **Next**. The **Select Page Entity** screen opens.
7. Select **Client Project** from the list.
8. Click **Next**. The **New Page Details** screen opens.
9. Type *Insert Project* in the **Page Title** and then press **Tab**. This automatically populates the **Page Description** field.



10. Type *InsertProject* in the **Page Alias** field.
11. Click **Next**. The **Select Page Content** screen opens.
12. Expand **QuickForms > ClientProject**.
13. Drag-and-drop the **ProjectDetails** smart part into the **MainContent** area.  
**Note:** If you don't see the **ProjectDetails** smart part listed, you must build the Web Platform.
14. Click **Next**. The **Portal Page Wizard Summary** screen opens.
15. Click **Finish**. The **Page** opens in the workspace.
16. Click the **Smart Parts** tab.
17. Select **Insert** from the **Show in Mode** drop-down list within the **Project Details** area.

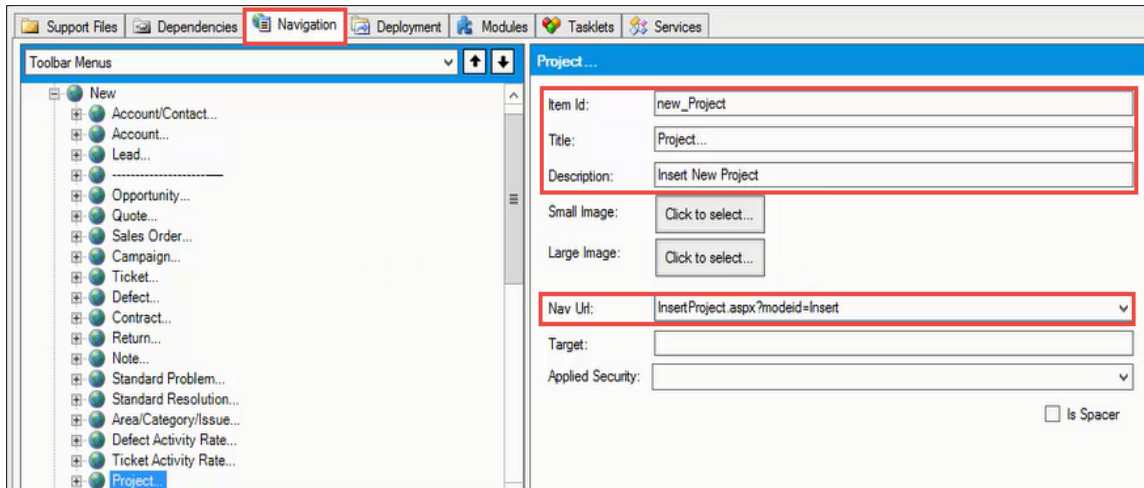


18. Click **Save All**.

## Part 2: Create a Project menu item

1. Open the **Project Explorer**.
2. Expand **LFS > Portal Manager**.
3. Double-click the **SixClient** node.
4. Click the **Navigation** tab.
5. Select **Toolbar Menus** from the drop-down list.
6. Expand **Menus**.
7. Right-click **New** and select **Add Menu Item**.
8. Change the following properties for the new menu item:
  - **Item ID:** new\_Project
  - **Title:** Project...
  - **Description:** Insert New Project
  - **Nav Url:** InsertProject.aspx?modeid=Insert





9. Move the **Project** menu item above the **separator line** “-----“ menu item using the upward facing arrow.
10. Click **Save All**.

You may run the website now and test the **Insert Projects** page to verify it's working. However, you do not have a way within the Web Client to view the information you inserted yet.



### Exercise 3.4: Creating a View page

In this exercise, you will create a main view page for a new entity and create new navigation bar options to access the new pages.

#### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

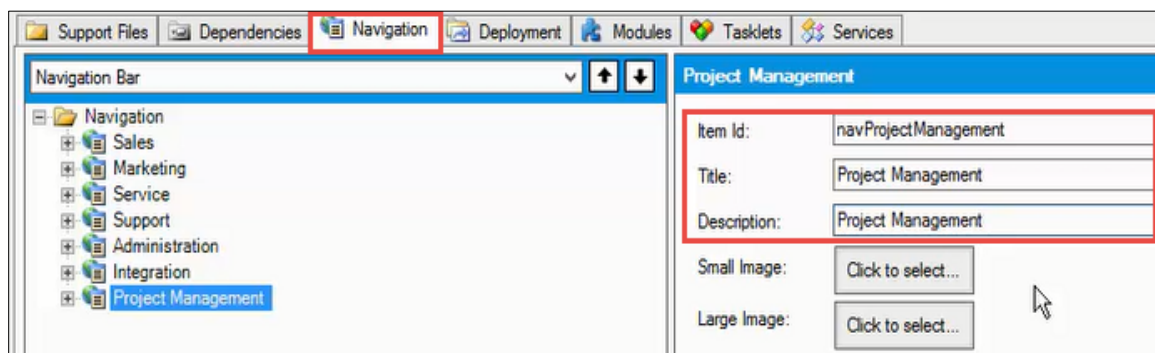
### Part 1: Create a Project View page

1. Open the **Project Explorer**.
2. Expand **LFS > Portal Manager > SlxClient**.
3. Right-click the **Pages** node and click **New Page Wizard**. The **Portal Page Wizard** opens.
4. Select **Main View** from the **Page Types** drop-down list.
5. Click **Next**. The **Select Page Entity** screen opens.
6. Select **Client Project**.
7. Click **Next**. The **New Page Details** screen opens.

8. Type *Project Detail* in the **Page Title** field and then press **Tab**. This automatically populates the **Page Description** field.
9. Verify **ClientProject** is selected in the **Page Alias** field.
10. Click **Next**. The **Select Page Content** screen opens.
11. Expand **QuickForms > ClientProject**.
12. Drag-and-drop the **ProjectDetails** smart part into the **MainContent** area.
13. Click **Next**. The **Portal Page Wizard Summary** screen opens.
14. Click **Finish**. The page opens in the workspace.
15. Click the **Smart Parts** tab and then click the **ProjectDetails** smart part. Notice the **Show in Mode** property is set to **Detail**.
16. Click the **LiveGroupViewer** smart part.
17. Verify **List** is selected in the **Show In Mode** field.
18. Click **Save All**.

## Part 2: Create a Nav Bar Group

1. Open the **Project Explorer**.
2. Expand **LFS > Portal Manager**.
3. Double-click **SlxClient**. The **SlxClient** portal opens.
4. Click the **Navigation** tab.
5. Right-click the **Navigation** folder and select **Add Navigation Group**.
6. Change the following properties for the new navBar1 group:
  - **Item Id:** navProjectManagement
  - **Title:** Project Management
  - **Description:** Project Management



7. Move the **Project Management Nav** group item up so it's immediately below the **Sales** item.
8. Click **Save All**.
9. Right-click the new **Project Management Nav** group and select **Add Navigation Item**.
10. Change the following properties for the new **Nav** item:
  - **Item Id:** navEntitiesProjects
  - **Title:** Projects

## 42 Lesson 3: Creating a custom entity

- **Nav Url:** ClientProject.aspx

11. Click **Save All**.

### Part 3: Run the website

1. Click **Run** from the **Application Architect** toolbar.

Wait for Application Architect project to build, deploy, and open the portal. When the browser opens, minimize Application Architect. The **Log On** page opens.



### Exercise 3.5: Inserting and viewing a project

In this exercise, you will test the Insert Projects page, the Client Projects main view, and create a group to view all projects.

#### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Insert a Project

1. Open the **Infor CRM Web Client**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **Sign In**.
4. Select **New > Project** from the menu bar. The **Insert Project** page opens.

The Group List is not able to load yet because you haven't created one but the form should still load in the main window. You want to enter a few different sample projects linked to various accounts in the database but because you don't have an Account lookup, let's hard-code an AccountID manually using the value *AGHEA0002669*.

5. Enter the following values for the first project belonging to the Abbott Ltd. account:

Insert Project			
ProjectDetails			
Title	<input type="text" value="Project 1"/>	Description	<input type="text" value="This is the first pro..."/>
Estimated Cost	<input type="text" value="\$1,000.00"/>	ManagerID	<input type="text"/>
		AccountID	<input type="text" value="AGHEA0002669"/>
StartDate	<input type="text" value="3/29/2016"/>	EndDate	<input type="text" value="6/29/2016"/>



You can enter in more than just one project if you want.

## Part 2: Create a group to view projects

1. Select the **Project Management Nav Bar > Projects**. The **Projects** main view opens.



None of your project records show up in the view yet because you haven't created a group.

2. Click **Add Group**. The **Query Builder** opens.
3. Type *All Projects* in the **Name** and **Display Name** fields.
4. Click the **Layout** tab and select the **CLIENT\_PROJECT** folder to highlight it.
5. Double-click the following properties to add them as columns in the group. Double-click each column one at a time and modify the following properties:
  - **Title:** Select the **Entity Linked** check box.
  - **StartDate:** Type *Start Date* in the **Caption** field and select **Date/Time** from the **Format Type** drop-down list.
  - **EndDate:** Type *End Date* in the **Caption** field and select **Date/Time** from the **Format Type** drop-down list.
  - **Description:** No changes.
  - **Id:** Clear the **Visible** check box.

Properties   Conditions <u>Layout</u> Sorting   Defaults				
Double click a field from the view above to place it in the grid below. Use the Move Left and Move Right buttons to change the order of the fields in the grid.				
Title	Start Date	End Date	Description	Id
CLIENT_PROJECT.TITLE	CLIENT_PROJECT.STARTDATE	CLIENT_PROJECT.ENDDATE	CLIENT_PROJECT.DESCRIP	CLIENT_PROJECT.CLIENT
120	120	120	120	120
Left	Left	Left	Left	Left
None	Date/Time	Date/Time	None	None
Visible	Visible	Visible	Visible	Hidden
Yes	No	No	No	No

6. Expand **CLIENT\_PROJECT** and click the **Account** node.
7. Double-click the **Account Name** column to add it to the right of the **Id** column in your layout.
8. Double-click the **Account Name** column.
9. Select the **Entity Linked** check box.
10. Click **OK**.
11. Click **OK** again to close the **Query Builder**. The main view shows the list of projects you added earlier.

### Part 3: Backup your work

1. Open **Application Architect**.
2. Select **View > Project Workspace Manager**.
3. Right-click the **LFS** project workspace and select **Backup Project**. The **Backup Project** window opens.
4. Click the **ellipsis**. The **Set Project Backup FileName** window opens.
5. Type *LFS03.backup.zip* in the **File name** field.
6. Click **Save**.
7. Click **OK**.



You've created a new page (ClientProject.aspx) for existing project details (ProjectDetails.ascx) and could reuse the ProjectDetails smart part you created for the Insert Project page (InsertProject.aspx) by changing the **Show In Mode** field from **Insert** to **Detail**. You also created a new **Nav** bar group and **Nav** item to launch the main view page. Finally, every main view requires a group so it knows what properties to list.

## Check your understanding



What are some common properties of a new entity?



---

---

---

---

---

---

---

---



What items should I be aware of when creating a relationship in the New Entity Wizard?



---

---

---

---

---

---

---

---



What are the main components in a Main View?



---

---

---

---

---

---

---

---



# Lesson 4: Workspace enhancements

## Estimated time

3 hours

## Learning objectives

In this lesson, you will:

- Customize a main view page to mimic out of the box features.
- Add context menus to navigation items.
- Redirect to the detail page after inserting an object.

## Topics

- Create Project Detail page filters
- Create Project Detail page common tasks
- Add a GroupNavigator to the ProjectDetails form
- Add a context menu to the Project Nav Item that launches the Insert Project page



# Workspace enhancements

Keep in mind for when creating a customization that you want end users to easily adopt the new changes. One way to help this adoption process is to ensure the customization looks like the out-of-the-box Infor CRM product that they are used to working with. This can include context menus, filters, and more.

Now that you can insert and view the Client Project entity within the Web Client, let's enhance its usability by creating filters for the detail page which allow for easier data location. Also, let's create a group navigator to switch between objects within a group and group/list views. Lastly, let's add context menus to the navigation bar items so they mimic the out of the box functionality of other navigation bar items.



## Exercise 4.1: Enhancing the View page

In this exercise, you will enhance the Main View page of a custom Infor CRM entity.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Create two ClientProject filters

A filter appears in the **List** view of a page when a user wants to filter records based on criteria to view a subset of an entire group.

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Open the **Project Explorer**.
5. Expand **LFS > Entity Model > Packages > Training > ClientProject**.
6. Right-click the **Filters** node and select **New Filter**. A new filter displays in the workspace.

Let's create a **StartDate** filter. This filter lists all projects with a start date in a month. For example, "show me all projects with a start date in January, February, or March."

7. Set the following properties for the new filter:
  - **Filter Name:** Start
  - **Display Name:** Start
  - **Type:** Range
  - **Property:** StartDate
  - **Characters:** 1
  - **Ranges:** SlxMonthly (For more valid range values, explore some of the existing entities.)

8. Click **Save**.
9. Right-click the **Filters** node and select **New Filter**. A new filter displays in the workspace.

Let's now create an **EndDate** filter. This filter lists all projects with an end date in a month. For example, "show me all projects with an end date in January, February, or March."

10. Set the following properties for the new filter:

- **Filter Name:** End
- **Display Name:** End
- **Type:** Range
- **Property:** EndDate
- **Characters:** 1
- **Ranges:** SlxMonthly

11. Click **Save**.

## Part 2: Create ClientProject common tasks

The common tasks pane displays in both **List** and **Detail** views. It gives quick access to common tasks related to an entity. To create common tasks, you can modify the code (.cs) for the Common Tasks Smart part (CommonTasksTasklet.ascx file).

1. Open the **Project Explorer**.
2. Expand **LFS > Portal Manager > SlxClient > SupportFiles > SmartParts > TaskPane > CommonTasks**.
3. Double-click **CommonTasksTasklet.ascx.cs**.

4. Scroll to the following method: **private void FillListViewDictionaries**.

**Note:** Press **CTRL+F** to find this method quickly.

5. Select and copy the code for the **accountListTasks** and then paste it on the empty line below.

```

1227 #region Fill Dictionaries
1228
1229 private void FillListViewDictionaries()
1230 {
1231     string[,] accountListTasks =
1232     {{"tskAddToGroup", "TaskText_AddToGroup", "javascript:commonTaskActions.showAdHocList(event);", "false"},
1233     {"tskSaveAsNewGroup", "TaskText_SaveAsNew", "javascript:commonTaskActions.saveSelectionsAsNewGroup();", "false"},
1234     {"tskRemoveFromGroup", "TaskText_Remove", "javascript:commonTaskActions.removeSelectionsFromGroup();", "false"},
1235     {"tskPromote", "TaskText_Promote", "javascript:Sage.Utility.Dashboard.promoteGroupToDashboard();", "false"},
1236     {"tskImportAccount", "TaskText_Import", "javascript:commonTaskActions.importFromFile();", "false"},
1237     {"tskBulkUpdateAccount", "TaskText_BulkActionUpdate", "javascript:commonTaskActions.bulkActionUpdateJob();", "f"},
1238     {"tskDeleteAccount", "TaskText_BulkDelete", "javascript:commonTaskActions.bulkActionDeleteJob();", "false"},
1239     {"tskExportToExcel", "TaskText_Export", "javascript:commonTaskActions.exportToFile();", "false"}
1240     };
1241     tasksByEntityList.Add("IAccount", accountListTasks);
1242

```

6. Rename the following three items in the copied code as shown:

```

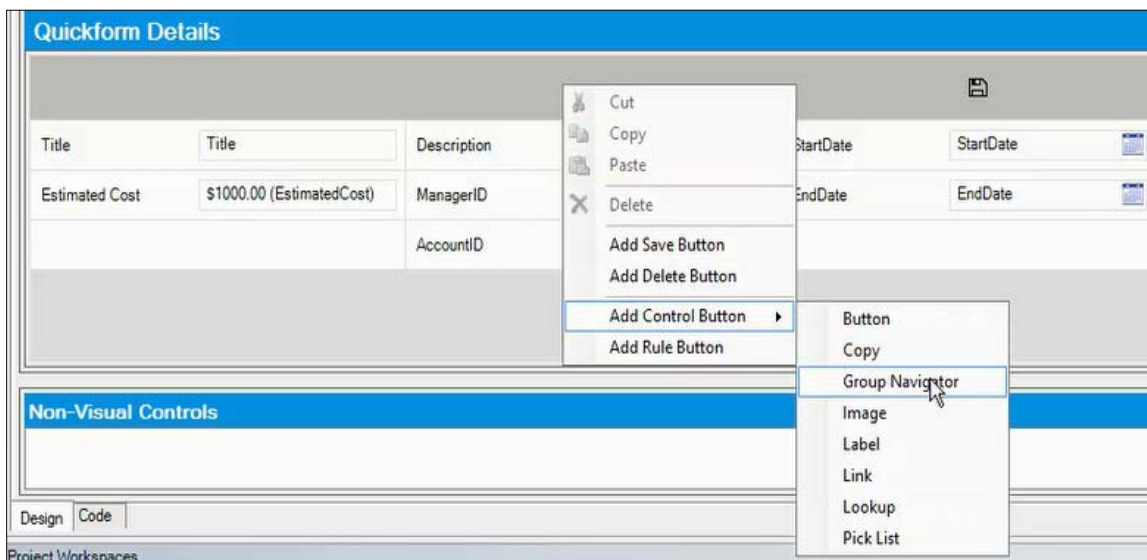
string[,] clientprojectListTasks =
{{"tskAddToGroup", "TaskText_AddToGroup", "javascript:commonTaskActions.showAdHocList(event);", "false"},
{"tskSaveAsNewGroup", "TaskText_SaveAsNew", "javascript:commonTaskActions.saveSelectionsAsNewGroup();", "false"},
{"tskRemoveFromGroup", "TaskText_Remove", "javascript:commonTaskActions.removeSelectionsFromGroup();", "false"},
{"tskPromote", "TaskText_Promote", "javascript:Sage.Utility.Dashboard.promoteGroupToDashboard();", "false"},
{"tskImportAccount", "TaskText_Import", "javascript:commonTaskActions.importFromFile();", "false"},
{"tskBulkUpdateAccount", "TaskText_BulkActionUpdate", "javascript:commonTaskActions.bulkActionUpdateJob();", "f"},
{"tskDeleteAccount", "TaskText_BulkDelete", "javascript:commonTaskActions.bulkActionDeleteJob();", "false"},
{"tskExportToExcel", "TaskText_Export", "javascript:commonTaskActions.exportToFile();", "false"}
};
tasksByEntityList.Add("IClientProject", clientprojectListTasks);

```

7. Click **Save**.

### Part 3: Add a group navigator to the ProjectDetails form

1. Open the **Project Explorer**.
2. Expand **LFS > Entity Model > Packages > Training > ClientProject > Forms**.
3. Double-click the **ProjectDetails** form.
4. Right-click the center area of the form's toolbar, point to **Add Control Button**, and then click **Group Navigator**. The navigator displays on the form.



To see the properties for this control, click the **ellipsis** in the form's **Toolbar** property.

5. Click **Save All**.



### Exercise 4.2: Enhancing the Navigation Bar

In this exercise, you will enhance a custom Navigation Bar within Infor CRM entity.

#### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

### Exercise steps

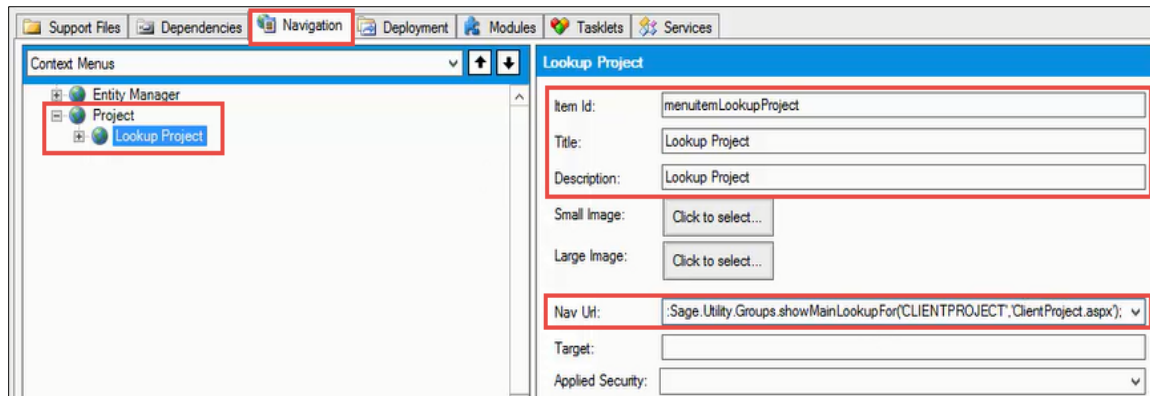
#### Part 1: Add a context menu to the Projects Nav Item

The context menu appears when a user right-clicks the nav item from the **Nav Bar**.

1. Open the **Project Explorer**.
2. Expand **LFS > Portal Manager > SlxClient > Context Menus**.

3. Right-click the **Context Menus** node and select **Add Menu Item**.
4. Set the following properties within the **Project** pane for the new context menu:
  - **Item Id:** contextProject
  - **Title:** Project
5. Click **Save**.
6. Right-click the **Project** node within the **Navigation** tab and select **Add Menu Item**. A new item is added to the Project context menu (Lookup Project).
7. Set the following properties for the new context menu:
  - **Item Id:** MenuItemLookupProject
  - **Title:** Lookup Project
  - **Description:** Lookup Project
  - **Nav Url:**  
javascript:Sage.Utility.Groups.showMainLookupFor('CLIENTPROJECT','ClientProject.aspx ');

This code is copied from the **Lookup Account** menu item and modified for ClientProject entity:



8. Click **Save**.
9. Right-click the **Project** node in the tree view and select **Add Menu Item**.
10. Set the following properties for the new context menu (New Project):
  - **Item Id:** MenuItemNewProject
  - **Title:** New Project...
  - **Description:** New Project
  - **Nav Url:** InsertProject.aspx?modeid=Insert
11. Click **Save**.
12. Right-click the **Project** node in the tree view and select **Add Menu Item**.
13. Set the following properties for the new context menu (Create New Group):
  - **Item Id:** MenuItemCreateNewGroup
  - **Title:** Create New Group
  - **Description:** Create New Group
  - **Nav Url:** javascript:Sage.Groups.GroupManager.CreateGroup('CLIENTPROJECT');

This code is copied from the **Lookup Account** menu item and has been modified for Client Project.

14. Click **Save**.
15. Select **Navigation Bar** from the **Context Menus** drop-down list.
16. Expand **Navigation > Project Management**.
17. Click the **Projects** node.
18. Select **contextProject** from the **Contact Menu** drop-down list.
19. Click **Save**.

## **Part 2: Add more Project Management nav items such as Account, Contact, and Ticket**

1. Right-click the **Project Management** nav group and select **Add Navigation Item**.
2. Set the following properties for the new nav item (Accounts):
  - **Item Id:** navProjectManagementAccounts
  - **Title:** Accounts
  - **Description:** Accounts
  - **Nav Url:** Account.aspx
  - **Context Menu:** contextAccount
3. Click **Save**.
4. Right-click the **Project Management Nav** group and select **Add Navigation Item**.
5. Set the following properties for the new nav item (Contacts):
  - **Item Id:** navProjectManagementContacts
  - **Title:** Contacts
  - **Description:** Contacts
  - **Nav Url:** Contact.aspx
  - **Context Menu:** contextContact
6. Click **Save**.
7. Right-click the **Project Management Nav** group and select **Add Navigation Item**.
8. Set the following properties for the new nav item (Tickets):
  - **Item Id:** navProjectManagementTickets
  - **Title:** Tickets
  - **Description:** Tickets
  - **Nav Url:** Ticket.aspx
  - **Context Menu:** contextTicket
9. Click **Save All**.



### Exercise 4.3: Enhancing the Insert page

In this exercise, you will create a redirect after insert on the insert page.

#### Notes:

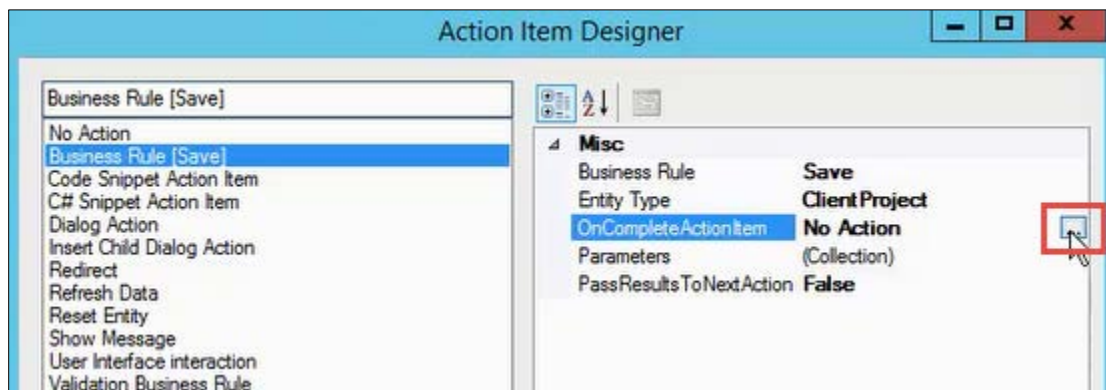
- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

### Exercise steps

1. Open the **Project Explorer**.
2. Expand **LFS > Entity Model > Packages > Training > ClientProject > Forms**.
3. Double-click **ProjectDetails**.
4. Click **Save** within the **ProjectDetails** form.
5. Select **View > Properties**.
6. Click **Save** within the **ProjectDetails** form again so it shows in the **Properties** pane.
7. Expand **On Click Action** and then click the **ellipsis** near the **Action Name** field. The **Action Item Designer** window opens.
8. Select the **Business Rule (Save) Action Item** in the left pane and then click the **ellipsis** near the **OnCompleteActionItem** field. A second **Action Item Designer** window opens.



9. Click the **Redirect Action Item** and then select **ClientProject** from the **MainView Entity Name** drop-down list.
10. Click **OK** to close the second **Action Item Designer** window.
11. Click **OK** again to close the first **Action Item Designer** window.
12. Click **Save** within the **Quickforms Details** form.



## Exercise 4.4: Verifying the enhancements

In this exercise, you will build and run the Infor CRM web client to verify the changes made.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

### Exercise steps

1. Open **Application Architect**.
2. Click **Run** from the toolbar.  
Wait for the Application Architect project to build, deploy, and open the portal. When the browser opens, minimize the Application Architect. The **Log On** page opens.
3. Type *admin* in the **Username** field. No password is required.
4. Click **Sign In**.
5. Expand the **Project Management Nav Bar**.
6. Right Click **Projects** and select **New Project**. The **Insert Project** screen should display.
7. Enter the following information into the **Insert Projects** page.
  - **Title:** Second Project
  - **Description:** This is the second project
  - **Amount:** \$1000.00
  - **Start Date:** today
  - **End Date:** 3-months from today
  - **AccountID:** AGHEA0002669
8. Click **Save**  
The save should redirect you to the **Project Details** page.
9. Click **List View** to switch to the list view.
10. Verify the new filters display as well as the **Common Tasks**.
11. Close the browser.
12. Create a backup of the project (LFS04.backup.zip) in Application Architect:
  - a. Select **View > Project Workspace Manager**.
  - d. Right-click **LFS** and select **Backup Project**.

## Check your understanding



When should I create a custom navigation bar?



---

---

---

---

---

---

---

---



Why should I create a context menu for a custom navigation bar?



---

---

---

---

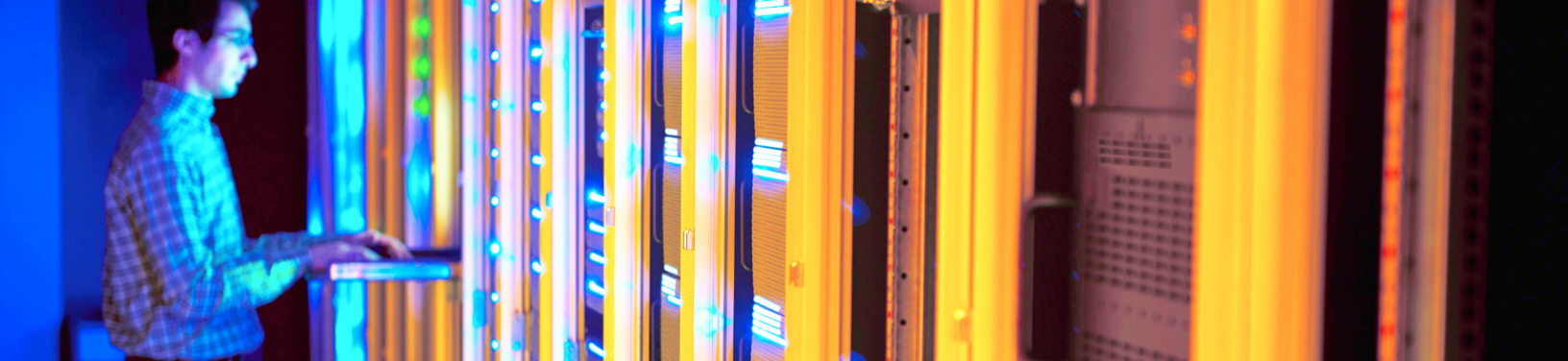
---

---

---

---





# Lesson 5: Adding automation to a form

## Estimated time

2 hours

## Learning objectives

In this lesson, you will:

- Create a form to show related entities in their parent's main view.
- Create a lookup to easily address relationships.
- Create automation to enhance security.

## Topics

- Create an AccountProjects form
- Create new relationships between ClientProject and UserInfo
- Create the Account and Project Manager lookups
- Add post-execute events to OnAfterInsert and OnAfterUpdate
- Add form load code to ProjectDetails

# Adding Automation to a Form

Now that most of the structure is created for the project, add automation to help end users adopt the changes. First, create a form to show the ClientProjects within the Account details page; Allow users to add new projects from this page. Second, change the UserInfo and Account lookup on the ClientProject detail page to a lookup control to assist with the relationships; Select a specific project manager and account instead of typing in Ids. Lastly, add code to set up ownerships, security, some events, and for creating new projects.



## Exercise 5.1: Associating an account to a project

In this exercise, you will customize the Web client to allow a user to see projects for an account.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Create an AccountProjects form

This form (or smart part) displays projects belonging to an account. When added to the **Account Detail** page it displays as a tab inside the **More Tabs** area.

1. Open the **Project Explorer**.
2. Expand **LFS > Entity Model > Packages > Saleslogix Application Entities > Account**.
3. Right-click **Forms** and select **New Form Wizard**. The **New Form Wizard** opens.
4. Select the **Create Form with a Grid** radio button.
5. Type *AccountProjects* in the **Form Name** field.
6. Click **Next**. The **Grid DataSource** screen opens.
7. Select the **Entity** radio button.
8. Select **ClientProjects** from the **Method or property** drop-down list. This value represents the relationship property that exists under the **Account** entity **Child** Properties.
9. Select **ClientProject** from the **Entity Type** drop-down list.
10. Click **Next**. The **Properties** screen opens.
11. Select the following properties to display on the form:
  - Description
  - EndDate
  - StartDate

- Title

12. Click **Next**. The **Summary** screen opens.

13. Click **Finish**. The form opens in the workspace.

Let's rearrange these columns and change the formatting.

Description	EndDate	StartDate	Title

14. Select **View > Properties**.

15. Click the **ellipsis** next to the **Columns** field under **Misc**. The **QFDataGridCol Collection Editor** window opens.

16. Select **Add > DateTime Column**.

17. Select **DataField** in the right pane under **Appearance** and then click the **ellipsis**. The **DataField Selector** window opens.

18. Select **StartDate**.

19. Click **OK**.

20. Select **ColumnHeading** in the right pane under **Appearance** and type *Start Date* in the field to the right of it.

21. Select the **BoundField (StartDate)** member and click **Remove**.

22. Click **Add** and then click **DateTime Column**.

23. Select **ColumnHeading** in the right pane under **Appearance** and type *End Date* in the field to the right of it.

24. Select **DataField** in the right pane under **Appearance** and then click its **ellipsis**. The **DataField Selector** window opens.

25. Select **EndDate**.

26. Click **OK**.

27. Select the **BoundField (EndDate)** member and click **Remove**.

28. Click **Add** and select **Link Column**.

29. Select **ColumnHeading** in the right pane under **Appearance** and type *Title* in the field to the right of it.

30. Select **DataField** in the right pane under **Appearance** and then click the **ellipsis**. The **DataField Selector** window opens.

31. Select **Title**.

32. Click **OK**.

33. Select **Entity TypeName** in the right pane under **Entity Linking** and then select **ClientProject** from its drop-down list in the field to the right of it.

34. Select the **BoundField (Title)** member and then click **Remove**. After viewing the results of these steps, you can choose to re-order these columns by using the up and down arrows.

35. Click **OK**.

36. Click **Save All**.

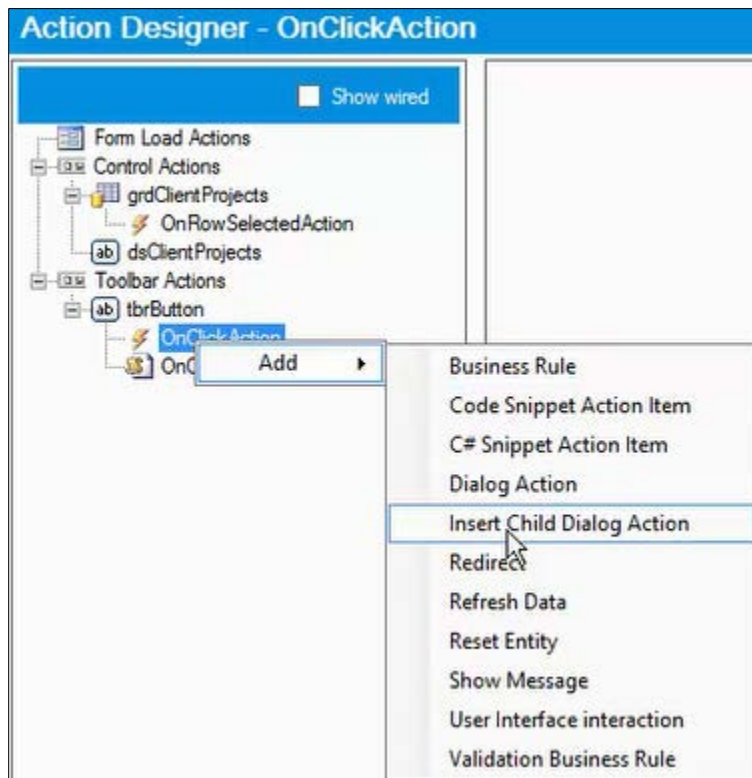
## Part 2: Create an Add button to the AccountProjects form

At this point the user is on the Account Detail page looking at the projects associated with an account. To add an **Add** button on the AccountProjects tab that allows a user to insert a new project, you could do a redirect action. Let's take it a step further to open the ProjectDetails.ascx in a workspace.

1. Open the **Project Explorer**.
2. Expand **LFS > Entity Model > Packages > Account > Forms**.
3. Click **AccountProjects**.
4. Right-click in the gray area above the columns and select **Add Control Button > Button**.
5. Modify the following button properties in the Properties pane:
  - **Button Type:** Icon
  - **Caption:** Add New Project
  - **Image:** plus\_16x16

You can also add a button to the form through the Toolbar property of the form.

6. Click **Save**.
7. Click the **Code** tab near the bottom of the **Quick Form Designer**.
8. Right-click **OnClickAction**, point to **Add**, and select **Insert Child Dialog Action**.



9. Set the following properties in the Insert **Child Dialog Action** area in the workspace:
  - **Data Source:** dsClientProjects
  - **Entity Type:** ClientProject

- **Parent Relationship Property:** Account
- **Smart Part:** ProjectDetails

10. Click **Save All**.
11. Expand **LFS > Portal Manager > SlxClient > Pages**.
12. Double-click **Account Detail**.
13. Click the **Smart Parts** tab.
14. Click **Add**. The **Select Smart Part** window opens.
15. Expand **Quick Forms > ClientProject**.
16. Click **ProjectDetails**.
17. Click **OK**. The smart part is added to the list at the bottom.
18. Select **DialogWorkspace** from the **Target Workspace** drop-down list in the **ProjectDetails** pane.
19. Click **Save All**.

### Part 3: Add the AccountProjects smart part to the Account Detail page

To create a quick form, build the Web Platform to compile them into usable .ascx files (smart parts). Again, you cannot browse to an .ascx file inside a browser so display that file inside a page (.aspx). Use the existing Account Detail page.

1. Click **Build Web Platform** and wait for the build to complete.
2. Close the **Output** window if open.
3. Click the **Smart Parts** tab and then click **Add**. The **Select Smart Part** window opens.
4. Expand **Quick Forms > Account** and then click **AccountProjects**.
5. Click **OK**.  
  
The smart part is added to the list at the bottom. Notice the Target Workspace is set to TabControl. Click the **Up** arrow to move the smart part to the second spot on the list, just below **AccountDetails**. **Note:** Moving it here will put the tab into the Tab Area without having to look under the **More Tabs** tab.
6. Ensure the **Show In Mode** field is set to **Detail** within the **AccountProjects** smart part properties.
7. Move the **AccountProjects** smart part up and under **AccountDetails**.
8. Click **Save All**.



You added a new AccountProjects smart part (AccountProjects.ascx) to the existing Account Details page (AccountDetails.aspx). Inside the data grid on the smart part, you added a link column to link back to the Project Detail page for the selected project. You also created a button to create a new ClientProject from the Account Detail page.



## Exercise 5.2: Creating lookups

In this exercise, you will customize the Project Details form to use lookups.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



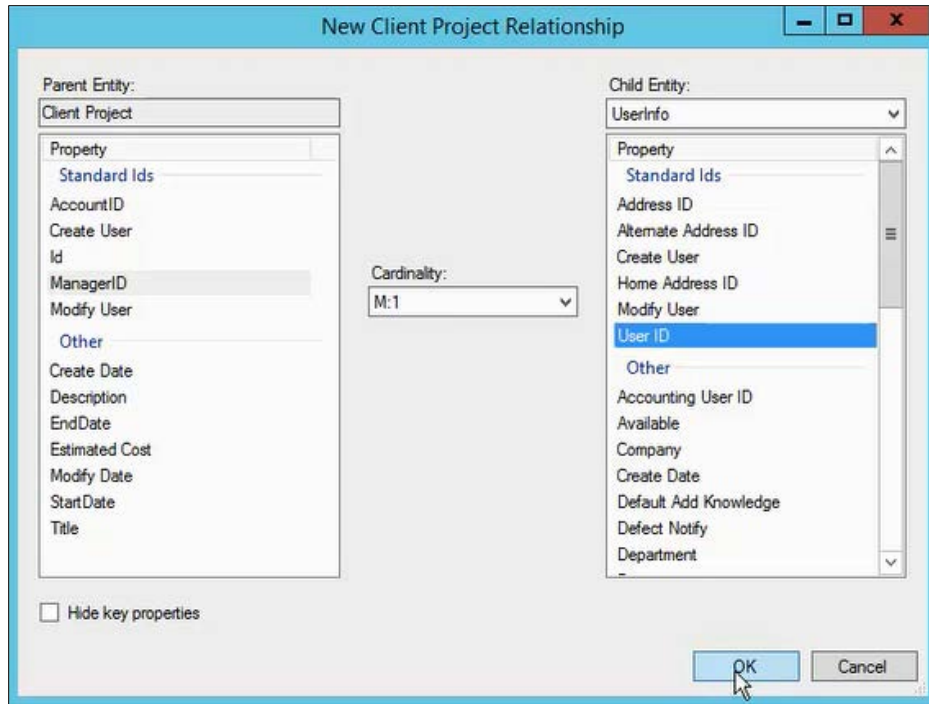
[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Create a new ClientProject relationship to the UserInfo entity

The Project Manager is always going to be an Infor CRM user so you must create a new relationship using the **ManagerId** property from the **ClientProject** entity and the **User Id** from the existing Infor CRM **UserInfo** entity.

1. Open the **Project Explorer**.
2. Expand **LFS > Entity Model > Packages > Training > ClientProject**.
3. Right-click the **Child Properties** node and select **New Relationship**.
4. Set the following relationship properties:
  - **Parent Entity:** Client Project
  - **Parent Entity Property:** ManagerID
  - **Cardinality:** M:1
  - **Child Entity:** UserInfo
  - **Child Entity Property:** User ID
5. Click **OK**. The details of the relationship display in the workspace.



6. Type *ProjectManager* in the **Property Name** and **Display Name** fields to make it more intuitive.
7. Click **Save All**.
8. Click **Build Interfaces**.

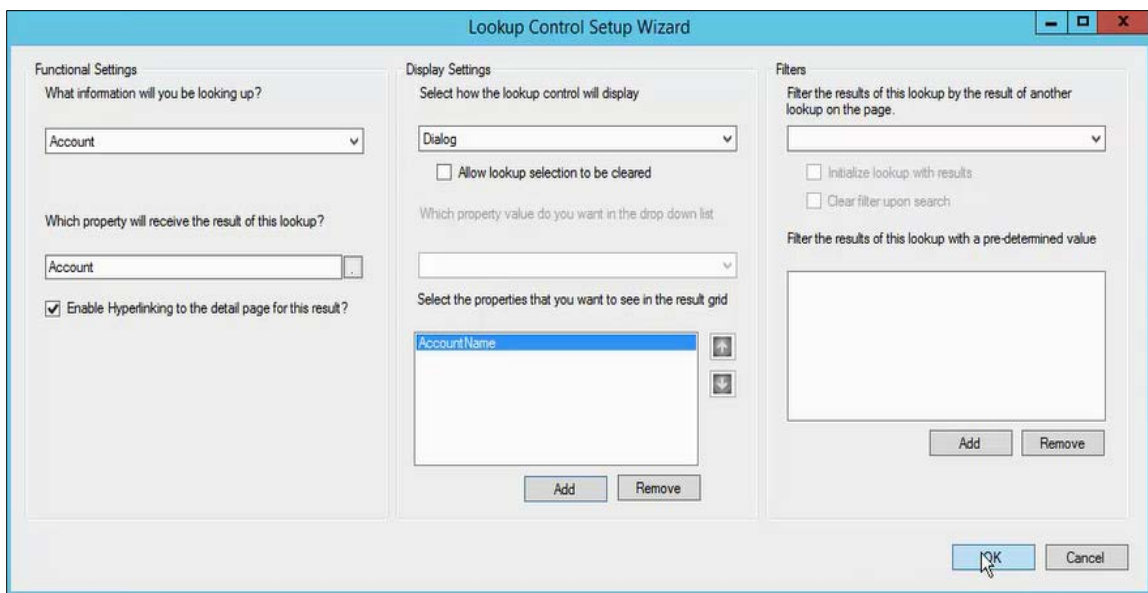
## Part 2: Create the Project Manager lookup

1. Expand **LFS > Entity Model > Packages > Training > ClientProject, > Forms**.
2. Click **ProjectDetails**.
3. Delete the **ManagerID** control by selecting the control and pressing **Delete**; This control is a placeholder.
4. Right-click in the now empty cell and select **Insert > Lookups > Lookup**. The **Lookup Control Setup Wizard** opens.
5. Set the following properties:
  - **What information will you be looking up?** User Info
  - **Which property will receive the result of this lookup?** ManagerID
  - **Enable Hyperlinking to the detail page for this result?** Do not select
  - **Select how the lookup control will display:** Dialog
  - **Select the properties that you want to see in the result grid:** UserName
6. Click **OK**. The control is added to the form but does not have a caption.
7. Select **Caption** within the **Appearance** area and type *Project Manager* in the field to the right of it.
8. Press **Tab**. The caption displays on the form.
9. Click **Save All**.



### Part 3: Create the Account lookup

1. Delete the **AccountID** control; This control is also a placeholder.
2. Right-click in the now empty cell and select **Insert > Lookups > Lookup**. The **Lookup Control Setup Wizard** opens.
3. Set the following properties:
  - **What information will you be looking up?** Account
  - **Which property will receive the result of this lookup?** Account
  - **Enable Hyperlinking to the detail page for this result?** Select
  - **Select how the lookup control will display:** Dialog
  - **Select the properties that you want to see in the result grid:** AccountName
4. Click **OK**. The control is added to the form but does not have a caption.



The screenshot shows the 'Lookup Control Setup Wizard' dialog box with three main sections: Functional Settings, Display Settings, and Filters. In Functional Settings, 'What information will you be looking up?' is set to 'Account', 'Which property will receive the result of this lookup?' is set to 'Account', and 'Enable Hyperlinking to the detail page for this result?' is checked. In Display Settings, 'Select how the lookup control will display' is set to 'Dialog', 'Allow lookup selection to be cleared' is unchecked, 'Which property value do you want in the drop down list' is empty, and 'AccountName' is selected in the 'Select the properties that you want to see in the result grid' list. The Filters section is empty. At the bottom right, the 'OK' button is highlighted with a mouse cursor.

5. Select the new **Account** lookup control.
6. Select **Caption** under **Appearance** in the right pane and type **Account** in the field next to it.
7. Click **Save All**



You've successfully added lookups to the Project Details page to assist end users with entering the correct information. Now you don't need the 12-character ID to properly add a Project Manager or an Account.

Now, let's add a new lookup for the Project Manager on the **ProjectDetails** form. This requires a relationship from the **ClientProject** entity to the **UserInfo** entity. These also enhance the AccountID to a lookup so an end user won't need the AccountID to properly add an Account to a Project.





## Exercise 5.3: Adding events to automate input

In this exercise, you will customize the Web client to allow a user to see projects for an account.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

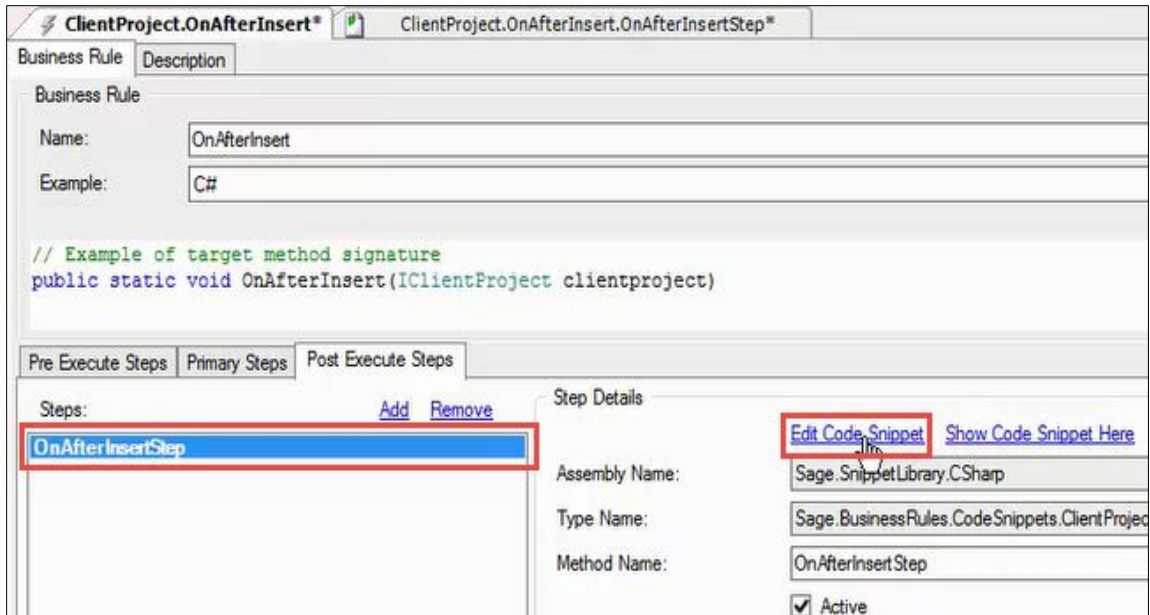
### Part 1: Add a post execute step to OnAfterInsert and OnAfterUpdate events

SECCODEID (Security Code ID) is the primary key in the SECCODE (Security Code) table of the Saleslogix database. Each team, department, and user you create is assigned a SECCODEID in the database. For example, when users create a new account record in the Infor CRM Client, Infor CRM uses the SECCODEID from the selected user or team to identify the owner.

Because we included SECCODEID in our CLIENTPROJECT entity, we need to handle how Infor CRM users can view projects created by other users. If you want projects to be viewable by everyone, consider using the **OnCreate** event. For example, if you want projects to be viewable only by the project manager (if set) or the account's default owner, use the **OnAfterInsert** and **OnAfterUpdate** events.

Depending on how security is set up for your implementation, users may be able to see projects owned by another user if they're part of that user's "user team."

1. Open the **Project Explorer**.
2. Expand **LFS > Entity Model > Packages > Training > ClientProject > Rules > Events**.
3. Double-click **OnAfterInsert**.
4. Click the **Post Execute Steps** tab.
5. Click **Add**. The **New Business Rule Step** window opens.
6. Select the **C# Code Snippet** radio button.
7. Click **OK**.
8. Select the **OnAfterInsertStep** you just created and click **Edit Code Snippet**.



9. Replace the //TODO code inside the code snippet with the following:

```
if(clientproject.UserInfo != null)
{
    clientproject.Owner = clientproject.UserInfo.User.DefaultOwner;
}
else
{
    clientproject.Owner =
    clientproject.Account.AccountManager.DefaultOwner;
}
clientproject.Save();
```

10. Click **Save**.
11. Double-click **OnAfterUpdate**.
12. Click the **Post Execute Steps** tab and then click **Add**. The **New Business Rule Step** window opens.
13. Select the **C# Code Snippet** radio button.
14. Click **OK**.
15. Select the **OnAfterUpdateStep** you just created and click **Edit Code Snippet**.
16. Click **OK**.
17. Replace the //TODO code inside the code snippet with the following:

```
if(clientproject.UserInfo != null)
{
    clientproject.Owner = clientproject.UserInfo.User.DefaultOwner;
```

```

    }
    else
    {
        clientproject.Owner =
        clientproject.Account.AccountManager.DefaultOwner;
    }
    clientproject.Save();

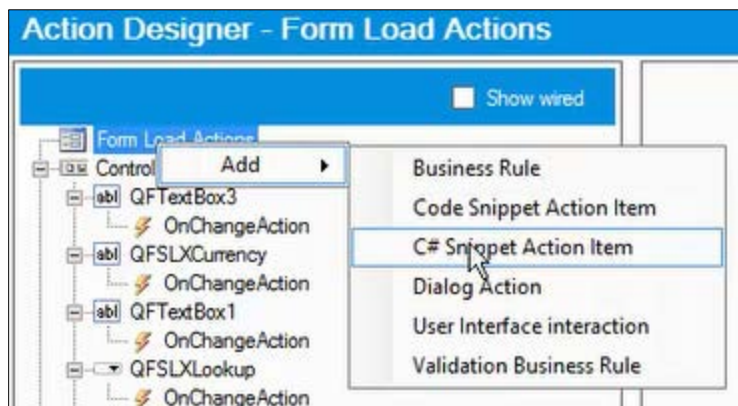
```

18. Click **Save All**.

## Part 2: Add form load code to the ProjectDetails form

You plan to launch the **ProjectDetails** form from the **AccountProjects** form. For a better user experience, let's add code to the **ProjectDetails** form where the parent entity defaults to the appropriate account name on the form.

1. Open the **Project Explorer**.
2. Expand **LFS > Entity Model > Packages > Training > Forms**.
3. Click **ProjectDetails**.
4. Click the **Code** tab near the bottom of the **ProjectDetails** form.
5. Right-click the **Form Load Actions** node and select **Add > C# Snippet Action Item**.



6. Add the following code in the **C# Code Snippet Action** workspace:

```

Sage.Entity.Interfaces.IClientProject project =
(Sage.Entity.Interfaces.IClientProject)this.BindingSource.Current;

object parent = this.GetParentEntity();

if(parent is Sage.Entity.Interfaces.IAccount)
{
    project.Account = (Sage.Entity.Interfaces.IAccount)parent;
}

```

This code gets the current project, then gets the project's parent entity. If the parent is account, it sets the account on the form to the same as the parent entity's ID.

7. Click **Save All**.



### Exercise 5.4: Verifying the automation changes

In this exercise, you will customize the Web client to allow a user to see projects for an account.

#### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)


## Exercise steps

### Part 1: Run the website

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Click **Run** from the toolbar.

Wait for the project to build, deploy, and open the portal. When the Infor CRM Web Client browser opens, minimize **Application Architect**. The **Log On** page opens.

5. Type *admin* in the **Username** field. No password is required.
6. Click **Sign In**.
7. Expand the **Sales Nav Bar** and click **Accounts**.
8. Click the **Abbott Ltd.** account record. The **Abbott Ltd. Account Detail** view opens.
9. Click **Add** within the **AccountProjects** tab.



AccountProjects					Details	Contacts	Opportunities	Activities	More Tabs
▼	Title	Description	Start Date	End Date					
	Project 1	this is the first project	2/29/2016	6/30/2016					
	Project 2	This is the second project	3/30/2016	6/30/2016					

Notice the modified **Account lookup** control and the new **Project Manager** lookup on the **Project Detail** view. Next, verify the form load code is working and the account shows up without data entry.

10. Close this window and the browser.

11. Create a backup of the project (**LFS05.backup.zip**) in **Application Architect**.

## Check your understanding



What is the benefit of using Lookups?



---

---

---

---

---

---

---

---



What OnClickActions are available for a control button?



---

---

---

---

---

---

---

---



What are the differences between the events?



---

---

---

---

---

---

---

---



What are the differences between Pre-Execute, Post-Execute, Primary, and Post Flush steps?



---

---

---

---

---

---

---

---



# Lesson 6: Associating multiple objects to an entity

## Estimated time

3 hours

## Learning objectives

In this lesson, you will:

- Create an Entity to allow many to many relationships.
- Create a form to allow for adding and editing data.
- Add code snippets to events to automate some data entry.
- Create forms to see the relationships.

## Topics

- Create a custom entity for a many to many relationship
- Create relationships to show a many to many
- Add events to the OnAfterUpdate and OnAfterInsert for the new entity
- Create forms to be used as smart parts for the many to many relationship



# Associating multiple objects to an entity

When you need to utilize a many to many relationship between two entities, the typical solution is to create a third entity to hold the information. This new entity will have two one to many relationships with the two entities that need many to many relationships. This allows you to add additional details to the entity if you want to track more details about the relationship.



## Exercise 6.1: Creating a many-to-many relationship

In this exercise, you will associate contacts to a project and to see projects for a contact.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Create a ClientProjectContact entity/table

From the design sketch, you determined the Contact to ClientProject has a many-to-many relationship. To resolve this, create an associated table with a one-to-many relationship on each side by first creating the entity and table.

1. Open the **Project Explorer**.
2. Expand **LFS > Entity Model > Packages**.
3. Right-click the **Training** node and select **New Entity**. The **New Entity Wizard** opens.
4. Select the **Create a business entity and a new table** radio button and click **Next**. The **New Entity From New Table** screen opens.
5. Type *ClientProjectContact* in the **Display Name** field and then press **Tab** to populate the remaining fields.
6. Select the **Enable Field-Level Security** check box.
7. Click **Next**. The **Primary Relationship** screen opens.

This is an association table between two entities: **ClientProject** and **Contact**; hence the name **ClientProjectContact**.

Although you must relate this entity to both the **ClientProject** and **Contact** entities, you'll create these relationships outside this wizard later. First, let's set up the keys to which these entities will eventually relate.

8. Verify the **Relate to an existing entity** check box is cleared.
9. Click **Next**. The **Enter Properties** screen opens.

This screen creates the properties you want the ClientProjectContact to have which includes only the common properties of both the **ClientProject** and **Contact** tables. These fields are already created for you:

The **CreateDate**, **ModifyDate**, **CreateUser**, and **ModifyUser** properties are included with every entity or table to support synchronization with Remote Clients. If you choose to enable field security, the **Seccodeid** property is also included after the wizard finishes.

10. Click **Add Property** or press **Alt+A** to add the following properties:

DisplayName	Property Name	Column Name	Data Type	Length	
Create User	CreateUser	CREATEUSER	Standard Id	12	
Create Date	CreateDate	CREATEDATE	Date Time	0	
Modify User	ModifyUser	MODIFYUSER	Standard Id	12	
Modify Date	ModifyDate	MODIFYDATE	Text	64	
ClientProjectID	ClientProjectID	CLIENTPROJEC...	Standard Id	12	
ContactId	ContactId	CONTACTID	Standard Id	12	
Role	Role	ROLE	Text	64	

11. Click **Next**. The **Additional Options** screen opens.
12. Select **Role** from the **Display Property** drop-down list.  
The selected Display Property is what appears on the title bar of the main view in the Web Client.
13. Click **Next**. The **Create Entities** screen opens.
14. Click **Next** again.
15. Click **Finish** when the process finishes. The **ClientProjectContact** entity appears in the workspace with all properties listed.
16. Click the **Entity** tab.
17. Type *Client Project Contact* in the **Display Name** field.
18. Type *Client Project Contacts* in the **Plural Display Name** field.
19. Clear the **Include** check box next to the **ClientProjectContactId** property.

**ClientProjectContact\***

Entity Description Code Templates Saleslogix Extended SData

Name: ClientProjectContact

Display Name: Client Project Contact

Plural Display Name: Client Project Contacts

Table Name: CLIENTPROJECTCONTACT

☐ Is Extension

Cache Usage: None

Entity Properties					
	Property Name	Display Name	Data Type	Include	Nullable
...	ClientProjectCont...	Id	Standard Id	<input type="checkbox"/>	<input type="checkbox"/>
	ClientProjectID	ClientProjectID	Standard Id	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	ContactId	ContactId	Standard Id	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Always go to an Id property through a relationship rather than referring to it directly. Choosing not to include this property prevents you from seeing it on any drop-down lists later to avoid confusion. In this scenario, you could choose not to include the **ClientProjectId** and **ContactId** properties.

20. Click **Save All**.

21. Click **Build Interfaces**.

## Part 2: Create the ClientProjectContact relationships

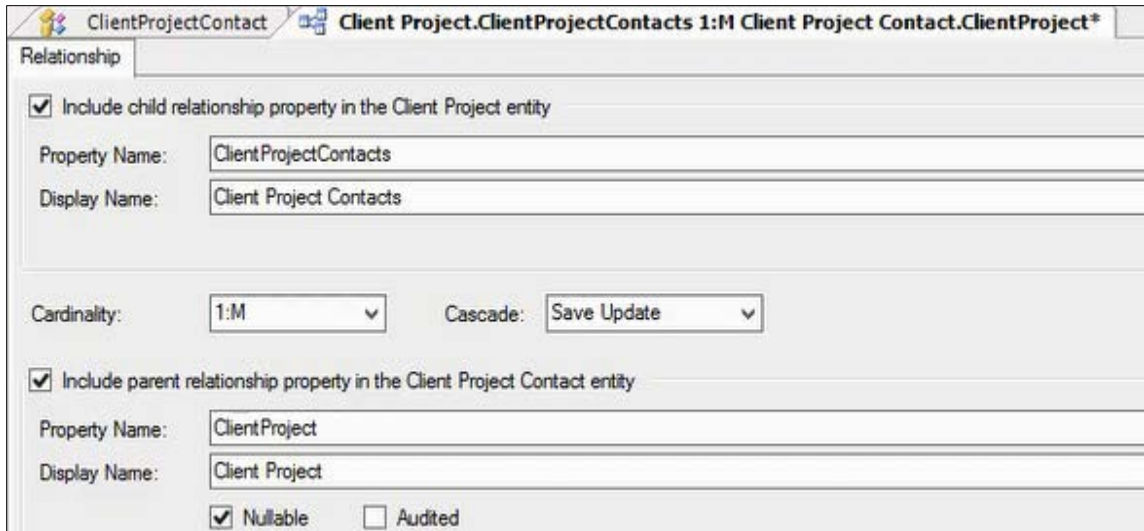
You must create two (2) one-to-many relationships.

1. Open the **Project Explorer**.
2. Expand **LFS > Entity Model > Packages > Training > ClientProjectContact**.

**Note:** Verify you're not on ClientProject.

3. Right-click **Parent Properties** and select **New Relationship**.
4. Set the following relationship properties:
  - **Parent Entity:** ClientProject
  - **Parent Entity Property:** Id
  - **Cardinality:** 1:M
  - **Child Entity:** Client Project Contact
  - **Child Entity Property:** ClientProjectID

5. Click **OK**. The details of the relationship appear in the workspace.



ClientProjectContact Client Project.ClientProjectContacts 1:M Client Project Contact.ClientProject\*

Relationship

☒ Include child relationship property in the Client Project entity

Property Name: ClientProjectContacts

Display Name: Client Project Contacts

Cardinality: 1:M Cascade: Save Update

☒ Include parent relationship property in the Client Project Contact entity

Property Name: ClientProject

Display Name: Client Project

☒ Nullable ☐ Audited

6. Click **Save All**.

This new relationship property appears in the **Child Properties** node under the **ClientProject** entity in the **Project Explorer**. You can also find it in the **Parent Properties** node under the **ClientProjectContact** entity.

7. Expand **LFS > Entity Model > Packages > Training > ClientProject**.

**Note:** Verify you're not on ClientProjectContact.

8. Right-click **Parent Properties** and select **New Relationship**.
9. Set the following relationship properties:

- **Parent Entity:** Contact
- **Parent Entity Property:** Contact ID
- **Cardinality:** 1:M
- **Child Entity:** Client Project Contact
- **Child Entity Property:** ContactId

10. Click **OK**. The details of the relationship appear in the workspace.

11. Click **Save All**.

This new relationship property appears in the **Child Properties** node under the **Contact** entity in the **Project Explorer**. You can also find it in the **Parent Properties** node under the **ClientProjectContact** entity.

12. Click **Build Interfaces**.

### Part 3: Add a post execute step to OnAfterInsert and OnAfterUpdate events

Similar to what you did with the **CLIENTPROJECT** entity, you must handle how Infor CRM users view project contact records created by other users. In this example, you want project contacts to be viewable only by the owner(s) of the project itself. If no project owner is set (which is not likely in this instance because you already coded this earlier), then you must set the project contact owner to the owner of the contact.

Depending on how security is set up for your implementation, users may be able to see projects owned by another user if they're part of that user's "user team."

1. Open the **Project Explorer**.

2. Expand **LFS > Entity Model > Packages > Training > ClientProjectContact > Rules > Events**.
3. Double-click **OnAfterInsert**.
4. Click the **Post Execute Steps** tab.
5. Click **Add**. The **New Business Rule Step** window opens.
6. Select the **C# Code Snippet** radio button.
7. Click **OK**.
8. Select the **OnAfterInsertStep** you just created and click **Edit Code Snippet**.
9. Replace the `//TODO` code inside the code snippet with the following:

```
//We do this because when we create a new clientprojectcontact from
SData (like with activities)

// the default entity created has no ClientProject
if(clientprojectcontact.ClientProject != null)
{
    if(clientprojectcontact.ClientProject.Owner != null)
    {
        clientprojectcontact.Owner = clientprojectcontact.ClientProject.Owner;
    }
    else if(clientprojectcontact.Contact != null)
    {
        //If we do not have an owner on the clientproject (which would be bad
        at
        //this point) then we will use the contact's owner.
        clientprojectcontact.Owner = clientprojectcontact.Contact.Owner;
    }
    clientprojectcontact.Save();
}
```

10. Click **Save**.
11. Expand **LFS > Entity Model > Packages > Training > ClientProjectContact > Rules > Events** again.
12. Double-click **OnAfterUpdate**.
13. Click the **Post Execute Steps** tab.
14. Click **Add**. The **New Business Rule Step** window opens.
15. Select the **C# Code Snippet** radio button.
16. Click **OK**.
17. Click the **OnAfterUpdateStep** you just created and then click **Edit Code Snippet**.
18. Replace the `//TODO` code inside the code snippet with the following:

```
//We do this because when we create a new clientprojectcontact from
SData (like with activities) the default entity created has no
ClientProject

if(clientprojectcontact.ClientProject != null)
{
if(clientprojectcontact.ClientProject.Owner != null)
{
clientprojectcontact.Owner = clientprojectcontact.ClientProject.Owner;
}
else if(clientprojectcontact.Contact != null)
{
//If we do not have an owner on the clientproject (which would be bad
at
//this point) then we will use the contact's owner.
clientprojectcontact.Owner = clientprojectcontact.Contact.Owner;
}
clientprojectcontact.Save();
}
```

19. Click **Save All**.



## Exercise 6.2: Creating an Add and Edit form

In this exercise, you will associate contacts to a project and to see projects for a contact.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Add an AddEditProjectContacts form

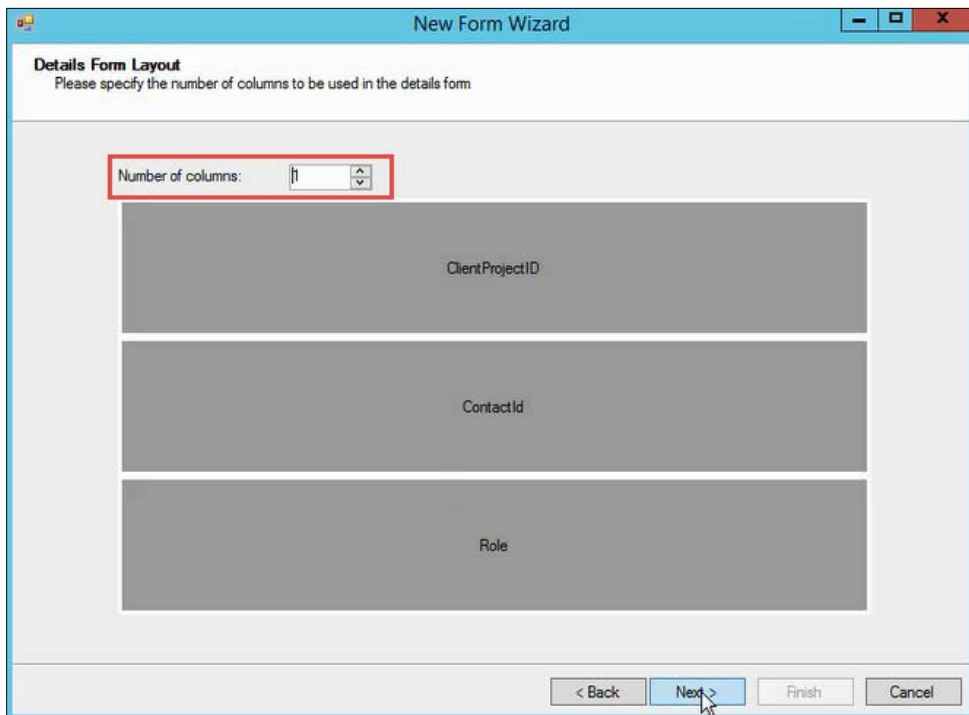
This form displays details for the contact who is associated with a project. You'll use the same form to add new and edit existing project contacts once you build the **Add/Edit Project Contacts** page (to be created).

1. Open the **Project Explorer**.
2. Expand **LFS > Entity Model > Packages > ClientProjectContact**.

3. Right-click **Forms** and select **New Form Wizard**. The **New Form Wizard** opens.
4. Select the **Create Detail Form** radio button.
5. Type *AddEditProjectContacts* in the **Form Name** field.
6. Click **Next**.
7. Select the following properties to display on the form in the **Properties** screen:
  - ClientProjectID
  - ContactId
  - Role

**Note:** The properties you choose to display on the form don't matter much because you'll delete and add them back as lookup controls. For now, keep them as placeholders for where the lookups will go.

8. Click **Next**. The **Details Form Layout** screen opens.
9. Set the **Number of columns** to 1.
10. Click **Next**. The **Summary** screen opens.

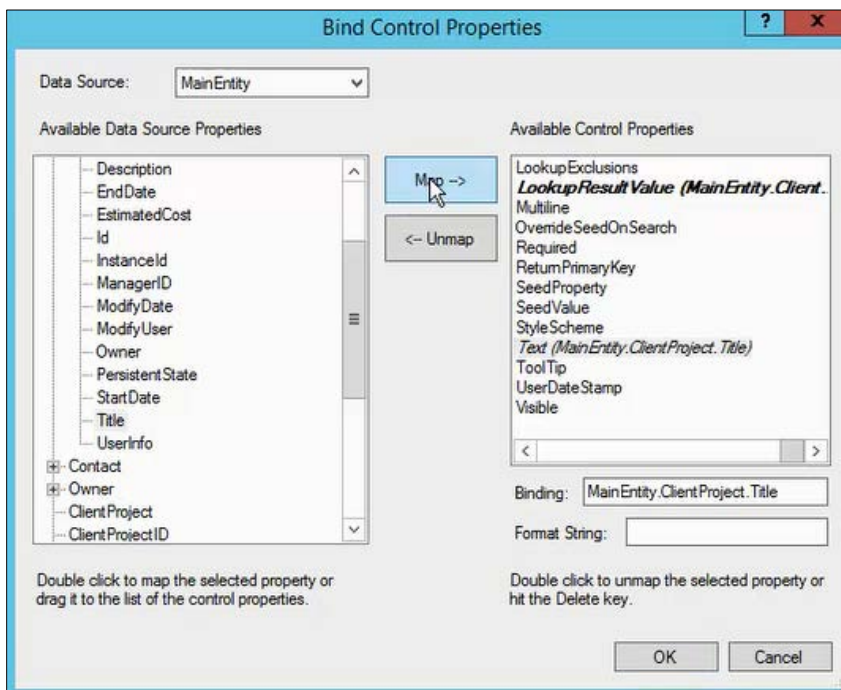


11. Click **Finish**. The form opens in the workspace.
12. Click **Save All**.

## Part 2: Create the Client Project & Contact lookups

1. Delete the **ClientProjectID** control.
2. Right-click in the now empty cell and select **Insert > Lookups > Lookup**. The **Lookup Control Setup Wizard** opens.
3. Set the following properties:

- **What information will you be looking up?** Client Project
  - **Which property will receive the result of this lookup?** ClientProject
  - **Enable Hyperlinking to the detail page for this result?** Select
  - **Select how the lookup control will display:** Dialog
  - **Select the properties that you want to see in the result grid:** Title, Description, StartDate, EndDate
4. Click **OK**. The control is added to the form, but does not have a caption.
  5. Select **View > Properties** to open the **Properties** pane.
  6. Select the new **ClientProject** lookup control.
  7. Select **Caption** within the **Appearance** area and type *Project* in the field to the right of it.  
**Note:** Click out of the field so the caption appears on the form.
  8. Select **Data Bindings** under **Appearance** in the right pane and click its ellipsis. The **Bind Control Properties** window opens.
  9. Expand the **ClientProject** node in the left pane and select **Title**.
  10. Select **Text** in the right pane and click **Map**.



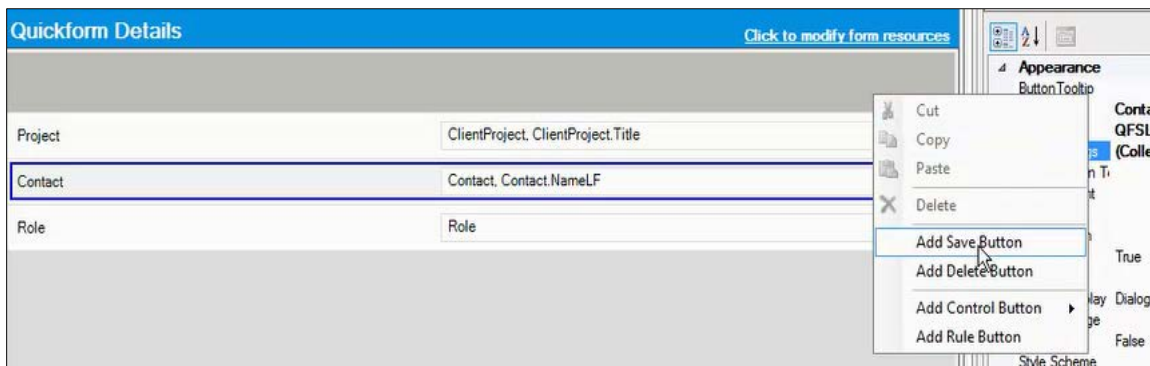
11. Click **OK**.
12. Delete the **ContactId** control.
13. Right-click in the now empty cell and select **Insert > Lookups > Lookup**. The **Lookup Control Setup Wizard** opens.
14. Set the following properties:
  - **What information will you be looking up?** Contact
  - **Which property will receive the result of this lookup?** Contact
  - **Enable Hyperlinking to the detail page for this result?** Select
  - **Select how the lookup control will display:** Dialog



- **Select the properties that you want to see in the result grid:** Account.AccountName, FirstName, LastName
15. Click **OK**. The control is added to the form but doesn't have a caption.
  16. Select the new **Contact** lookup control.
  17. Select **Caption** within the **Appearance** area and type *Contact* in the field to the right of it.
  18. Select **Data Bindings** with the **Appearance** area and click the **ellipsis**. The **Bind Control Properties** window opens.
  19. Expand the **Contact** node in the right pane and select **NameLF**.
  20. Select **Text** in the right pane and click **Map**.
  21. Click **OK**.
  22. Click **Save All**.

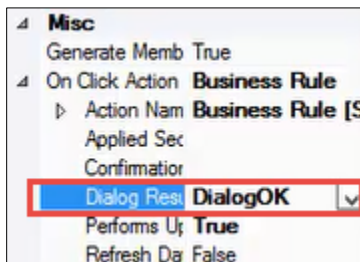
### Part 3: Add a Save button to the AddEditProjectContacts form

1. Right-click inside the top, gray area of the **AddEditProjectContacts** form and select **Add Save Button**.



A new **Save** button appears on the form. The button automatically includes an image and the built-in **Save** business rule as part of its on-click action. You still need to set additional properties through the Toolbar property on the form. Let's do this now.

2. Click **Save**.
3. Select **Caption** within the **Appearance** area and type *Save* in the field to the right of it.
4. Click **Save** within the **AddEditProjectContacts** form.
5. Expand **On Click Action** in the right pane property and select **Dialog Result**.
6. Select **DialogOK** from the **Dialog Result** drop-down list.



7. Click **Save All**.

#### Part 4: Add form load code to the AddEditProjectContacts form

Plan to launch this form from two locations: The Contact Detail view (Contact Projects tab) and the Project Detail view (Project Contacts tab). For a better experience for your user, let's add code to this form that will get the parent entity (i.e. the view from which it was launched) so it can default the appropriate project name or contact name on the form.

1. Click the **Code** tab near the bottom of the **AddEditProjectContacts** form.
2. Right-click the **Form Load Actions** node and select **Add > C# Snippet Action Item**.
3. Add the following code in the C# Code Snippet Action workspace:

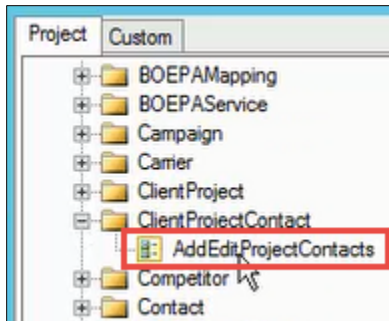
```
Sage.Entity.Interfaces.IClientProjectContact project =  
this.BindingSource.Current as  
Sage.Entity.Interfaces.IClientProjectContact;  
  
object parent = this.GetParentEntity();  
if(parent is Sage.Entity.Interfaces.IContact)  
{  
project.Contact = (Sage.Entity.Interfaces.IContact)parent;  
}  
else if (parent is Sage.Entity.Interfaces.IClientProject)  
{  
project.ClientProject = (Sage.Entity.Interfaces.IClientProject)parent;  
}
```

This code gets the current project, then gets the project's parent entity. If the parent is a contact, it will set the contact on the form to the same as the parent entity's ID. If the parent entity is a project, it will set the project on the form to the same as the parent entity's ID.

4. Click **Save All**.
5. Click **Build Web Platform**.

#### Part 5: Add the AddEditProjectContacts smart part to the Project Detail page

1. Open the **Project Explorer**.
2. Expand **LFS > Portal Manager > SlxClient > Pages**.
3. Double-click **Project Detail**.
4. Click the **Smart Parts** tab and then click **Add a new smart part**. The **Select Smart Part** window opens.
5. Expand **QuickForms > ClientProjectContact** and then double-click **AddEditProjectContacts**.



6. Click **OK**.
7. Select **DialogWorkspace** from the **Target Workspace** drop-down list within the **AddEditProjectContacts** pane.
8. Click **Save All**.

You just built the new smart part (.ascx) and chose to house it inside the existing **Project Detail** page (.aspx). This time, rather than including this smart part as a tab in the view, let's choose to open it inside a dialog window. This way, when a user adds or edits a project contact, they can close the dialog and return to where he/she came from.

Where did the user come from? This part hasn't been completed yet, so let's do this next. Start by creating the **ProjectContacts** form with a data grid added as a tab on the **Project Detail** page. This way, a user can launch the **AddEditProjectContacts** dialog box to add new project contacts. You'll also repeat a similar process to create a **ContactProjects** tab allowing users to add a contact to an existing project from the **Contact Detail** page.



### Exercise 6.3: Creating a form to view and add ClientProjectContacts

In this exercise, you will associate contacts to a project and to see projects for a contact.

#### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Create a ProjectContacts form

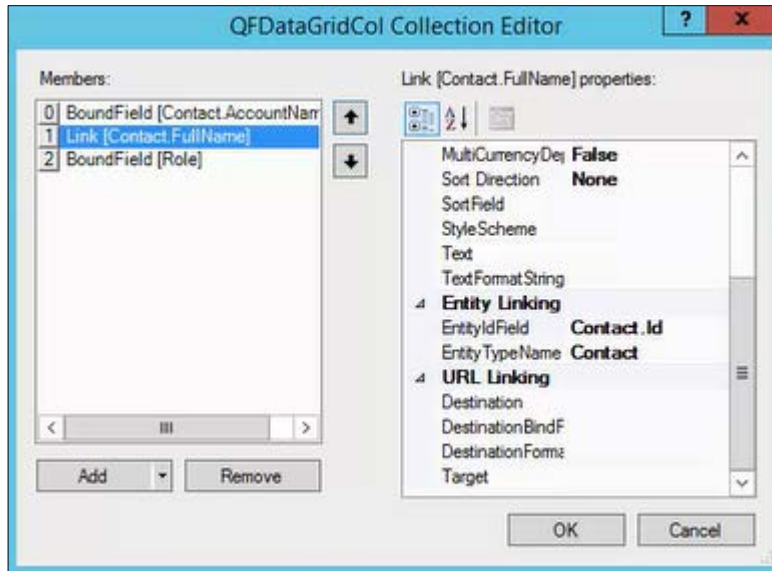
This form (or smart part / .ascx file) displays contact details inside a data grid for contacts that belong to a project. When added to the Project Detail page, it appears as another tab inside the More Tabs area.

1. Open the **Project Explorer**.
2. Expand **LFS > Entity Model > Packages > ClientProject**.
3. Right-click **Forms** and select **New Form Wizard**. The **New Form Wizard** opens.

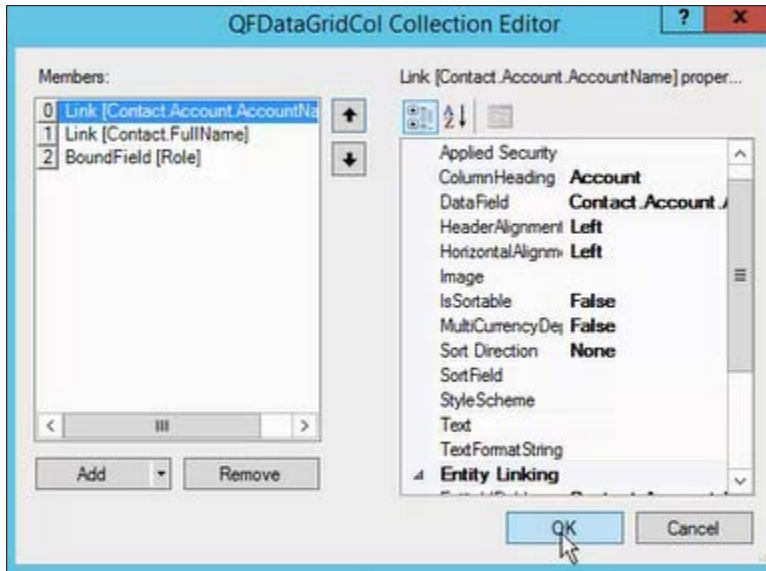
4. Select the **Create Form with a Grid** radio button.
5. Type *ProjectContact* in the **Form Name** field.
6. Click **Next**. The **Grid DataSource** screen opens.
7. Click the **Entity** radio button.
8. Select **ClientProjectContacts** from the **Method or property** drop-down list.
9. Select **ClientProjectContact** from the **Entity Type** drop-down list.
10. Click **Next**. The **Properties** screen opens.
11. Select the following properties to display in the grid:
  - **Contact (expand the node) > AccountName** (Account Name)
  - **Contact (expand the node) > FullName** (Full Name)
  - **Role**
12. Click **Next**. The **Summary** screen opens.
13. Click **Finish**. The form opens in the workspace.

Let's rearrange the columns and change the formatting.
14. Select **View > Properties**.
15. Click the data grid columns.
16. Select **Columns** under **Misc** in the right pane and click the **ellipsis**. The **QFDataGridCol Collection Editor** window opens.
17. Select **BoundField [Contact.FullName]** from the **Members** pane and click **Remove**.

**Note:** This is a Bound Field which displays the contact name as plain text. You want to display the name as a link back to the contact's details, so you must re-add the column as a **Link Column**.
18. Select **Add > Link Column**.
19. Select **ColumnHeading** in the right pane and type *Name* in the field to the right of it.
20. Select **DataField** in the right pane and click the **ellipsis**. The **DataField Selector** window opens.
21. Expand **Contact** and select **FullName**.
22. Click **OK**.
23. Select **EntityTypeName** in the right pane and then select **Contact** from its drop-down list.
24. Select **EntityIdField** in the right pane and type *Contact.Id* in the field to the right of it.
25. Move the **Link [Contact.FullName]** above **BoundField [Role]**.



26. Select **BoundField [Contact.AccountName]** in the **Members** area and click **Remove**.
27. Select **Add > Link Column**.
28. Move the new link column above **BoundField [Role]**.
29. Select **ColumnHeading** in the right pane and type *Account* in the field to the right of it.
30. Select **DataField** in the right pane and then click its **ellipsis**. The **DataField Selector** window opens.
31. Expand **Contact > Account** and select **AccountName**.
32. Click **OK**.
33. Select **EntityType** in the right pane and then select **Account** from its drop-down list.
34. Select **EntityIdField** in the right pane and type *Contact.Account.Id* in the field to the right of it.
35. Move the **Link [Contact.Account.AccountName]** above **Link [Contact.FullName]**.
36. Click **OK**.

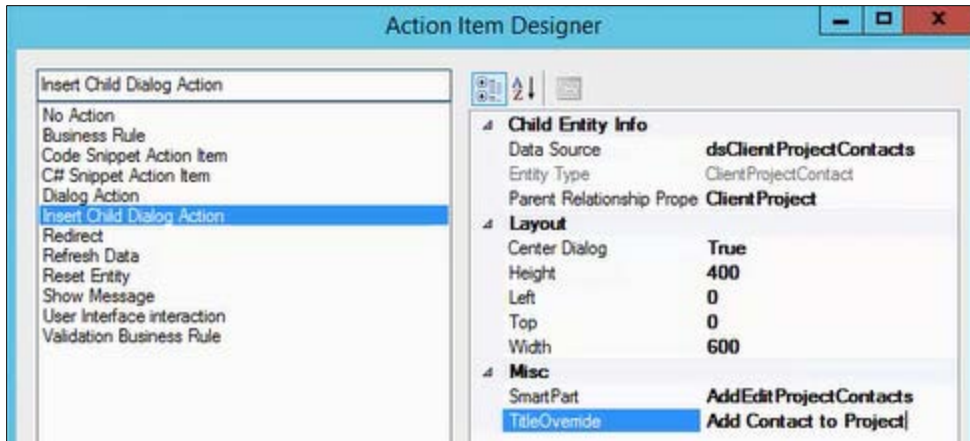


37. Click **Save All**.

## Part 2: Add an Add button to the ProjectContacts form

At this point, the user is on the **Project Detail** page looking at the contacts associated with a project. You want to add an **Add** button on the ProjectContacts tab allowing a user to insert a new contact.

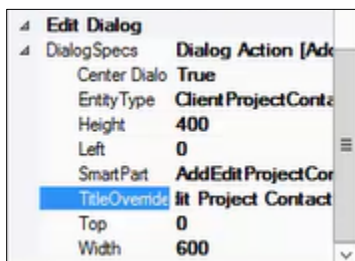
1. Open the **ProjectContacts** form and then click **View > Properties** to display its properties.
2. Right-click in the top, gray area of the **ProjectContacts** form and then click **Add Control Button > Button**.
3. Set the following properties:
  - **Button Type:** Icon
  - **Caption:** Add contact
  - **Image:** plus\_16x16
  - **Tooltip:** Add New Contact to Project
4. Expand **On Click Action** under **Misc** in the right pane
5. Select **Action Name** and click the **ellipsis**. The **Action Item Designer** window opens.
6. Select **Insert Child Dialog Action** in the left pane.
7. Set the following properties:
  - **Data Source:** dsClientProjectContacts
  - **Entity Type:** ClientProjectContact
  - **Parent Relationship Property:** ClientProject
  - **SmartPart:** AddEditProjectContacts
  - **TitleOverride:** Add Contact to Project



8. Click **OK** to close the **Action Item Designer**.
9. Click **Save All**.

### Part 3: Add an Edit and Delete column to the ProjectContacts form

1. Open the **ProjectContacts** form.
2. Click the data grid columns to view its properties.
3. Select **Columns** within the **Misc** area and click the **ellipsis**. The **QFDataGridCol Collection Editor** window opens.
4. Select **Add > Edit Column**.
5. Select **ColumnHeading** within the **Appearance** area and type *Edit* in the field to the right of it.
6. Select **Text** within the **Appearance** area and type *Edit* in the field to the right of it.
7. Expand **DialogSpecs** within the **Edit Dialog** area.
8. Select **SmartPart** and then select **AddEditProjectContact** from its drop-down list.
9. Select **TitleOverride** in the **Edit Dialog** area.
10. Select **Dialog Specs** and type *Edit Project Contact* in the field to the right of it.



11. Select **Add > Delete Column**.
12. Change the following properties:
  - **ColumnHeading**: Delete
  - **Text**: Delete
  - **BusinessRuleObjectName**: ClientProjectContact
  - **BusinessRule**: Delete
13. Click **OK**.

14. Click **Save All**.

#### Part 4: Add the ProjectContacts smart part to the Project Detail page

As with the other quick forms you created, you must build the Web Platform to compile them into usable .ascx files (smart parts). Again, you cannot browse to an .ascx file inside a browser so you need to display this file inside a page (.aspx). Use the existing **Project Detail** page.

1. Click **Build Web Platform**.
2. Open the **Project Explorer**.
3. Expand **LFS > Portal Manager > SlxClient > Pages**.
4. Double-click **Project Detail**.
5. Click the **Smart Parts** tab.
6. Click **Add**. The **Select Smart Part** window opens.
7. Expand **QuickForms > ClientProject** and then double-click **ProjectContacts**. The smart part is added to the list at the bottom. Notice the **Target Workspace** is set to **TabControl** and the **Show In Mode** is set to **Detail**.
8. Click **Save All**.



#### Exercise 6.4: Creating a form to view Contact Projects

In this exercise, you will associate contacts to a project and to see projects for a contact.

##### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

### Exercise steps

#### Part 1: Create a ContactProjects form

This form (or smart part / .ascx file) displays project details inside a data grid for projects associated with a contact. When added to the **Contact Detail** page, it appears as another tab inside the **More Tabs** area.

1. Open the **Project Explorer**.
2. Expand **LFS > Entity Model > Packages Saleslogix Application Entities > Contact**.
3. Right-click **Forms** and select **New Form Wizard**. The **New Form Wizard** window opens.
4. Select the **Create Form with a Grid** radio button.
5. Type *ContactProjects* in the **Form Name** field.
6. Click **Next**. The **Grid DataSource** screen opens.
7. Select the **Entity** radio button.



8. Select **ClientProjectContacts** from the **Method or property** drop-down list.
9. Select **ClientProjectContact** from the **Entity Type** drop-down list.
10. Click **Next**. The **Properties** screen opens.
11. Select the following properties to display in the grid:
  - ClientProject (expand the node) > EndDate
  - ClientProject (expand the node) > UserInfo
  - ClientProject (expand the node) > StartDate
  - ClientProject (expand the node) > Title
12. Click **Next**. The **Summary** screen opens.
13. Click **Finish**. The form opens in the workspace.

Let's rearrange the columns and change the formatting.
14. Click the data grid columns and open the **Properties** pane.
15. Select **Columns** within the **Misc** area and click the **ellipsis**. The **QFDataGridCol Collection Editor** window opens.
16. Select **BoundField [ClientProject.Title]** and click **Remove**.

**Note:** This is a Bound Field which displays the contact name as plain text. Because you want to display the name as a link which returns the user to the contact's details, you must re-add the column as a Link Column.
17. Select **Add > Link Column**.
18. Change the following properties:
  - **ColumnHeading:** Title
  - **DataField:** ClientProject.Title
  - **SortField:** ClientProject.Title
  - **EntityTypeName:** ClientProject
  - **IsSortable:** True
  - **EntityIdField:** ClientProject.Id
19. Select **BoundField [ClientProject.StartDate]** and click **Remove**.
20. Select **Add > DateTime Column**.
21. Change the following properties:
  - **ColumnHeading:** Start Date
  - **DataField:** ClientProject.StartDate
  - **IsSortable:** True
  - **SortField:** ClientProject.StartDate
22. Select **BoundField [ClientProject.EndDateDate]** and click **Remove**.
23. Select **Add > DateTime Column**.
24. Change the following properties:
  - **ColumnHeading:** End Date
  - **DataField:** ClientProject.EndDate
  - **IsSortable:** True
  - **SortField:** ClientProject.EndDate

25. Select **BoundField [ClientProject.UserInfo]** and move it below **Link [ClientProject.Title]**.
26. Change the following properties:
  - **ColumnHeading:** Project Manager
  - **DataField:** ClientProject.UserInfo.NameLF
27. Select **Add > Edit Column**.
28. Change the following properties:
  - **ColumnHeading:** Edit
  - **Text:** Edit
  - **SmartPart:** AddEditProjectContacts
29. Select **Add > Delete Column**.
30. Change the following properties:
  - **ColumnHeading:** Delete
  - **Text:** Delete
  - **BusinessRuleObjectName:** ClientProjectContacts
  - **BusinessRule:** Delete
31. Click **OK**.
32. Click **Save All**.

## Part 2: Add an Add button to the ContactProjects form

At this point, the user is on the **Contact Detail** page looking at the projects associated with a contact. You want to add an **Add** button on the **ContactProjects** tab allowing a user to insert a new project.

1. Open the **ContactProjects** form.
2. Right-click in the top, gray area of the **ContactProjects** form and select **Add Control Button > Button**.
3. Set the following properties:
  - **Button Type:** Icon
  - **Caption:** Add project
  - **Image:** plus\_16x16
  - **Tooltip:** Add New Project to Contact
4. Expand **On Click Action** within the **Misc** area.
5. Select **Action Name** and click the **ellipsis**. The **Action Item Designer** window opens.
6. Select **Insert Child Dialog Action**.
7. Set the following properties:
  - **Data Source:** dsClientProjectContacts
  - **Entity Type:** ClientProjectContact
  - **Parent Relationship Property:** Contact
  - **SmartPart:** AddEditProjectContacts
  - **TitleOverride:** Add New Project to Contact
8. Click **OK**.
9. Click **Save All**.

### Part 3: Add the ContactProjects and AddEditProjectContacts smart parts to the Contact Detail page

As with the other quick forms you created, you must build the Web Platform to compile them into usable .ascx files (smart parts). Again, you cannot browse to an .ascx file inside a browser so you must display this file inside a page (.aspx). You'll use the existing **Contact Detail** page.

1. Click **Build Web Platform**.
2. Open the **Project Explorer**.
3. Expand **LFS > Portal Manager > SlxClient > Pages**.
4. Double-click **Contact Detail**.
5. Click the **Smart Parts** tab and then click **Add**.
6. Expand **QuickForms > Contact** and then double-click **ContactProjects**.

The smart part is added to the list at the bottom. Notice the Target Workspace is set to **TabControl** and the **Show In Mode** is set to **Detail**.

Id	Custom	Workspace	Roles
AddEditTargetRespo...	Yes	DialogWorkspace	
LiveGroupViewer	Yes	MainContent	
MergeRecords	Yes	DialogWorkspace	
MergeChildren	Yes	DialogWorkspace	
SyncResultsHistory	Yes	TabControl	
MergeContactAddress	Yes	DialogWorkspace	
MergeAddress	Yes	DialogWorkspace	
ErpContactAccounts	No	TabControl	
ContactProjects	No	TabControl	

Contact Projects	
Smart Part ID:	ContactProjects
Smart Part:	ContactProjects
Title:	ContactProjects
Description:	ContactProjects
Target Workspace:	TabControl
Show In Mode:	Detail

7. Move the **ContactProjects** smart part up and below **ContactDetails**.
8. Click the **Smart Parts** tab and then click **Add**.
9. Expand **QuickForms > ClientProjectContact** and then double-click **AddEditProjectContacts**.
10. Select **DialogWorkspace** from the **Target Workspace** drop-down list within the **AddEditProjectContacts** area.
11. Click **Save All**.



#### Exercise 6.5: Verifying the many-to-many association

In this exercise, you will associate contacts to a project and to see projects for a contact.

##### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Test within the Projects Detail view

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Click **Run** from the toolbar.  
Wait for the project to build, deploy, and open the portal. When the browser opens, minimize **Application Architect**. The **Log On** page opens.
5. Type *admin* in the **Username** field. No password is required.
6. Click **Sign In**.
7. Expand the **Project Management Nav Bar** and click **Projects**.
8. Click one of the projects to view the **Project Details** view. The **ProjectContacts** tab opens but no contacts have been added yet.
9. Click **Add New Contact to Project** to add a new project contact. The **Add Contact to Project** dialog box opens.  
**Note:** The **Project** lookup defaults to the current project. Use the **Contact** lookup to select a contact to associate.
10. Search and select **Abbott, John** in the **Contact** field.
11. Type *SME* in the **Role** field.
12. Click **Save**.



Although it would be ideal to default the current project and show only contacts in the lookup who belong to the associated account, you'll skip this customization in class as it's too time-consuming. If you're interested in learning how to do this, explore another out-of-the-box form that implements the same kind of functionality or watch a video in the video library.

13. Click **Add New Contact to Project** to add another user. The **Add Contact to Project** dialog box opens. You'll test the **Edit** and **Delete** links to confirm they work.
14. Search and select **Drew, Dean** in the **Contact** field.
15. Type *TBD* in the **Role** field.
16. Click **Save**.
17. Delete the row for **Drew, Dean**.
18. Click **Edit** within the row for **Abbott, John**.

19. Type *Project Lead* in the **Role** field and delete **SME**.
20. Click **Save**.
21. Close the browser.

#### **Part 2: Test within the Contact Detail view**

1. Click the link for **Abbott, John**. The **Contact Detail** view opens.
2. Click **Add** within the **ClientProjects** tab. The **Add New Project to Contact** window opens.
3. Search and select **Project 2** in the **Project** field.
4. Type *SME* in the **Role** field.
5. Click **Save**. Both projects now show in **John Abbotts Contact Detail** view.
6. Create a backup of the project (**LFS06.backup.zip**) in Application Architect.

## Check your understanding



How can I create a many-to-many relationship?



---

---

---

---

---

---

---

---



How can I use the new entity to show the relationship?



---

---

---

---

---

---

---

---



# Lesson 7: Modifying out-of-the-box entities

## Estimated time

2 hours

## Learning objectives

After completing this lesson, you will be able to <terminal learning objective for lesson X>. In this lesson, you will:

- Modify an out of the box Infor CRM entity.
- Modify an out-of-the-box detail form.

## Topics

- Modifying an out of the box entity
- Create a new relationship for an out of the box entity
- Add fields to an out-of-the-box form

# Modifying out of the box entities

Modifying out-of-the-box entities can greatly increase the amount of flexibility you have in your customizations. Adding a foreign key to a custom entity allows new relationships to be created. Adding a pick list property can allow for more control of business rules, including those from external assemblies.



## Exercise 7.1: Modifying an out-of-the-box entity

In this exercise, you will associate tickets to a project and to see projects for a ticket.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Modify the out-of-the-box Ticket entity

You must create a foreign key in the ticket table to relate it back to the ClientProjectId. If you're going to add a new field to an out-of-the box entity, you must use the **Database Manager** and update the property in Application Architect.

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Expand **LFS > Entity Model > Packages > Saleslogix Application Entities**.
5. Double-click **Ticket**.

Let's create a new ClientProjectId.

6. Select **Start > All Programs > Microsoft SQL Server 2012 > SQL Server Management Studio**.
7. Verify **SRV10\SQLEXPRESS** is selected in the **Server name** field.
8. Click **Connect**.
9. Expand **SRV10\SQLEXPRESS > Databases > saleslogix\_eval > Tables in the Object Explorer**.
10. Right-click **sysdba.TICKET** and select **Design**. The **Ticket** table opens.  
Notice there is no **CLIENTPROJECTID** field.
11. Close the **Ticket** table and minimize **SQL Server Management Studio**.
12. Close the **Ticket** project workspace within **Application Architect**.



13. Right-click the **Ticket** entity and select **Modify Entity**. The **New Entity Wizard** window opens.
14. Click **Add property** and set the following properties:
  - **Display Name:** ClientProjectID
  - **Name:** ClientProjectID
  - **Data Type:** Standard Id
  - **Column Name:** CLIENTPROJECTID
15. Click **Next** twice. Wait for the **Ticket** entity to save.
16. Click **Finish**.
17. Maximize **SQL Server Management Studio**.
18. Right-click **sysdba.Ticket** within the **Object Explorer** and select **Design**.

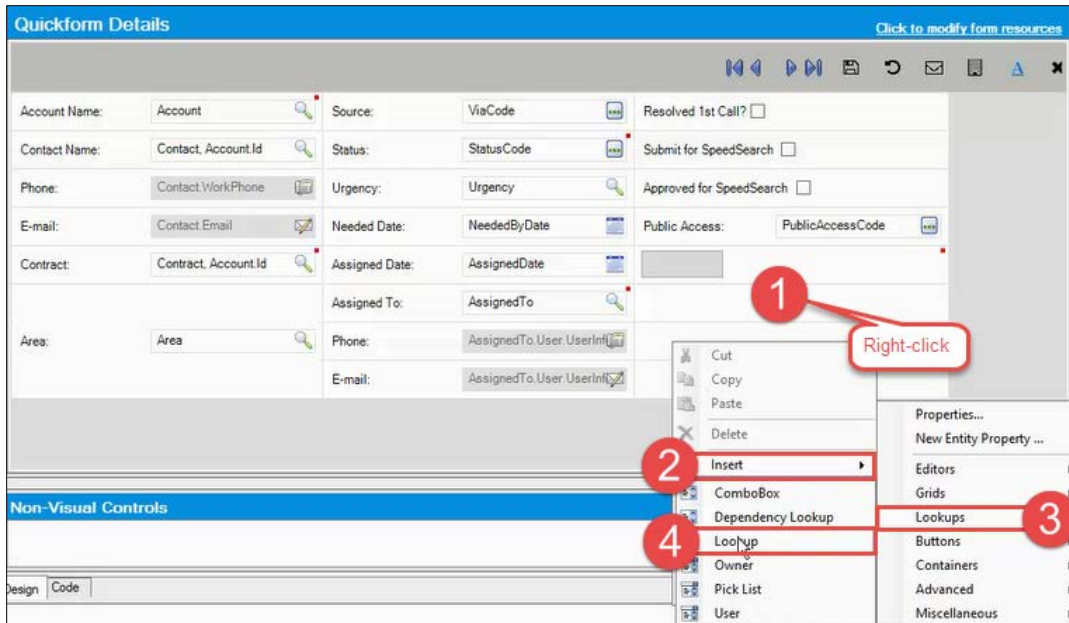
Notice the **CLIENTPROJECTID** field now displays at the bottom of the list. **Note:** This adds the field to the database and entity.
19. Close **SQL Server Management Studio**.
20. Click **Build Interfaces**.

## **Part 2: Associate the Ticket entity with the ClientProject custom entity**

1. Expand **LFS > Entity Model > Packages > Saleslogix Application Entities > Ticket**.
2. Right-click **Parent Properties** and select **New Relationship**. The **New Ticket Relationship** window opens.
3. Set the following relationship properties:
  - **Parent Entity:** ClientProject
  - **Parent Entity Property:** Id
  - **Cardinality:** 1:M
  - **Child Entity:** Ticket
  - **Child Entity Property:** ClientProjectID
4. Click **OK**. The new relationship shows in the project workspace.
5. Click **Save All**.
6. Click **Build Interfaces**.

## **Part 3: Update the Tickets form to show the new field**

1. Expand **LFS > Entity Model > Packages > Saleslogix Application Entities > Ticket > Forms**.
2. Double-click **TicketDetails**. The **TicketDetails** form opens.
3. Right-click in any empty field, point to **Insert > Lookups**, and select **Lookup**. The **Lookup Control Setup Wizard** opens.



4. Set the following properties:
  - **What information will you be looking up?** Client Project
  - **Which property will receive the result of this lookup?** ClientProject
  - **Enable Hyperlinking to the detail page for this result?** Select
  - **Select how the lookup control will display:** Dialog
  - **Select the properties that you want to see in the result grid:** Title
5. Click **OK**. The control is added to the form, but does not have a caption.
6. Select **View > Properties**.
7. Select **Caption** within the **Appearance** area and type *Project* in the field to the right of it.
8. Click **Save**.



## Exercise 7.2: Creating a ProjectTickets form

In this exercise, you will associate tickets to a project and to see projects for a ticket.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Create a ProjectTickets form

This form (or smart part / .ascx file) displays ticket details inside a data grid for tickets that belong to a project. When added to the **Project Detail** page, it appears as another tab inside the **More Tabs** area.

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Open the **Project Explorer**.
5. Expand **LFS > Entity Model > Packages > Training > ClientProject**.
6. Right-click **Forms** and select **New Form Wizard**. The **New Form Wizard** opens.
7. Select the **Create Form with a Grid** radio button.
8. Type *ProjectTickets* in the **Form Name** field.
9. Click **Next**. The **Grid DataSource** screen opens.
10. Select the **Entity** radio button.
11. Select **Tickets** from the **Method or property** drop-down list.
12. Select **Tickets** from the **Entity Type** drop-down list.
13. Click **Next**. The **Properties** screen opens.
14. Select the following properties to display in the grid:
  - **AssignedDate**
  - **NeededByDate**
  - **TicketNumber**
  - **Urgency**
15. Click **Next**. The **Summary** screen opens.
16. Click **Finish**. The form opens in the workspace.

Let's rearrange the columns and change the formatting.
17. Click the data grid columns.
18. Select **View > Properties**.
19. Select **Columns** under **Misc** in the right pane and then click its ellipsis. The **QFDataGrid Col Collection Editor** window opens.
20. Select **BoundField [TicketNumber]** and click **Remove**.
21. Select **Add > Link Column**.
22. Change the following properties:
  - **ColumnHeading**: Ticket
  - **DataField**: TicketNumber
  - **EntityType**: Ticket
  - **EntityIdField**: Id
23. Move **Link [TicketNumber]** to the top of the list.

24. Remove the **BoundField [AssignedDate]** and **BoundField [NeededByDate]** members.
25. Select **Add > DateTime Column**.
26. Change the following column's properties:
  - **ColumnHeading:** Assigned Date
  - **DataField:** AssignedDate).
27. Click **Add > DateTime Column**.
28. Change the column's properties
  - **ColumnHeading:** Needed By Date
  - **DataField:** NeededByDate
29. Move **BoundField [Urgency]** to the bottom of the list.
30. Click **OK**.
31. Click **Save All**.

## Part 2: Add an Add button to the ProjectTickets form

At this point, the user is on the **Project Detail** page looking at the tickets associated with a project. You want to add an **Add** button on the **ProjectTickets** tab allowing users to insert a new ticket.

1. Open the **ProjectTickets** form and then click the form to see its properties.
2. Right-click in the gray area above the **Urgency** column and select **Add Control Button > Button**.
3. Modify the following properties:
  - **Button Type:** Icon
  - **Caption:** Add New Ticket
  - **Image:** Plus\_16x16
  - **Tooltip:** Add New ticket
4. Expand **On Click Action** within the **Misc** area.
5. Select **Action Name** and click the **ellipsis**. The **Action Item Designer** window opens.
6. Select **Redirect**.
7. Set the following properties:
  - **MainView Entity Mode:** Insert
  - **URL:** InsertTicket.aspx?modeid=Insert
  - **Use Current Id in Link:** True
8. Click **OK**.
9. Click **Save All**.

## Part 3: Add the ProjectTickets smart part to the Project Detail page

As with the other quick forms you created, you must build the Web Platform to compile them into usable .ascx files (smart parts). Again, you cannot browse to an .ascx file inside a browser so you must display this file inside a page (.aspx). You'll use the existing **Project Detail** page.

1. Click **Build Web Platform**.
2. Open the **Project Explorer**.

3. Expand **LFS > Portal Manager > SlxClient > Pages**.
4. Double-click **Project Detail**.
5. Click the **Smart Parts** tab and then click **Add**.
6. Expand **QuickForms > ClientProject** and then double-click **ProjectTickets**.

The smart part is added to the list at the bottom. Notice the **Target Workspace** is set to **TabControl** and the **Show In Mode** is set to **Detail**.

7. Click **Save All**.



### Exercise 7.3: Verifying the Ticket associations

In this exercise, you will associate tickets to a project and to see projects for a ticket.

#### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Run the website

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Click **Run** from the toolbar.  
Wait for the project to build, deploy, and open the portal. When the browser opens, minimize Application Architect. The **Log On** page opens.
5. Type *admin* in the **Username** field. No password is required.
6. Click **Sign In**.
7. Expand the **Project Management Nav Bar** and click **Tickets**.
8. Open the **Ticket Detail** view for the first ticket in the list.
9. Search and select **Project 1** in the **Project** field.
10. Click **OK**.
11. Click **Save**.

Let's verify everything works.

12. Click the **Project 1** link. Notice there is a new **ProjectTickets** tab.
13. Click the **ProjectTickets** tab and then click **Add New Ticket**. The **Insert Ticket** page opens.

Now that you've confirmed everything works, create a backup of the project (**LFS07.backup.zip**) within Application Architect.

## Check your understanding



How can I add a new property to an out of the box entity?



---

---

---

---

---

---

---

---



Can I create new relationships using out of the box entities?



---

---

---

---

---

---

---

---



# Lesson 8: Using pick lists

## Estimated time

1 hour

## Learning objectives

In this lesson, you will:

- Create a custom pick list.
- Apply a custom pick list to a form.

## Topics

- Create a custom picklist
- Add a picklist property to an existing entity
- Add the picklist to a detail form



# Using Pick Lists

Occasionally you will have a property on an entity that has a specific set of values that are valid. For example, within a **Defect** entity, there may be a severity property with the only acceptable values being; High, Medium, Low, and Urgent. Using a pick list is a great way to ensure these values are the only ones entered. Pick lists can also prevent data entry errors, as you are unable to make any typos.

Pick lists are also good for helping ensure business rules run properly. By creating a pick list to force the use of a select number of values, you can control validation by checking on those known values. This way, when you're creating a business rule, you know the valid value to check against.



## Exercise 8.1: Creating Pick Lists

In this exercise, you will create pick lists, entities, and variables in the forms so business rules can verify tickets and projects are closed properly.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

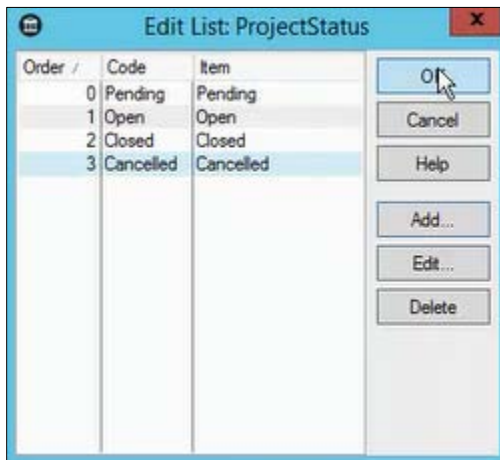
### Part 1: Modify the ProjectTickets form

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Expand **LFS > Entity Model > Packages > Training > ClientProject > Forms**.
5. Double-click **ProjectTickets**.
6. Click the data grid.
7. Select **View > Properties**.
8. Select **Columns** within the **Misc** area and click the **ellipsis**. The **QFDataGrid Col Collection Editor** window opens.
9. Select **Add > SLX Pick List Column**.
10. Change the following properties:
  - **ColumnHeading:** Status
  - **DataField:** StatusCode
  - **IsSortable:** True
  - **SortField:** StatusCode
  - **Pick List Name:** Ticket Status

11. Click **OK**.
12. Click **Save All**.

## Part 2: Add a Project Status pick list

1. Minimize **Application Architect**.
2. Select **Start > Programs > Saleslogix > Administrator**.
3. Type *admin* in the **Username** field. No password is required.
4. Click **Sign In**.
5. Select **Manage > PickLists**. The **Pick List Manager** window opens.
6. Click the **Custom** tab and then click **Add**. The **Pick List Name** window opens.
7. Type *ProjectStatus* in the **Name** field.
8. Click **OK**.
9. Add the following pick list items using the **Edit List: ProjectStatus** window:
  - **Order:** 0 / **Code:** Pending / **Item:** Pending
  - **Order:** 1 / **Code:** Open / **Item:** Open
  - **Order:** 2 / **Code:** Closed / **Item:** Closed
  - **Order:** 3 / **Code:** Cancelled / **Item:** Cancelled
10. Click **OK**.



11. Select the **Set Default** check box and then select the **Pending** item as the default.
12. Click **Close**.
13. Close the **Administrator**.

## Part 3: Add a Status property to the ClientProject entity

You must add a property to the **ClientProject** entity and field to the table in which to store the value of one of the **Project Status** pick list items. Unlike what you did with the out-of-the-box ticket entity, to add a new field to a custom entity for which you've also created the schema in Application Architect, you can modify the entity.

1. Open **Application Architect**.

2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Expand **LFS > Entity Model > Packages > Training**.
5. Right-click **ClientProject** and select **Modify Entity**. The **New Entity Wizard** opens.
6. Click **Add Property** or press **Alt+A** to modify the following properties:

7. Click **Next**. The **Summary** screen opens.  
**Note:** Despite what the message says, it's not creating a new entity; it's modifying it.
8. Click **Next** again. Wait for the entity to save.
9. Click **Finish**. The new **ProjectStatus** property is added to the entity. Double-click the **ClientProject** entity to view the property details to verify it was added.
10. Click **Build Web Platform**.

#### Part 4: Modify the ProjectDetails form

Now that you have a new **Status** property to store project status pick list items, you can add the pick list control to the **ProjectDetails** form.

1. Expand **LFS > Entity Model > Packages > Training > ClientProject > Forms**.
2. Double-click **ProjectDetails**.
3. Right-click in the empty cell (1st column) and select **Insert > Lookups > Pick List**.
4. Select **View > Properties**.
5. Select **Caption** within the **Appearance** area and type *Status* in the field to the right of it.
6. Select **Data Bindings** under **Appearance** area and click the **ellipsis**. The **Bind Control Properties** window opens.
7. Select **Status** and then click **Map**.
8. Click **OK**.
9. Select **Pick List Name** within the **Misc** area and then select **Project Status** from its drop-down list.

10. Click **Save**. The form should resemble the following figure:

Quickform Details <span>Click to modify form resources</span>					
<div> <span>⏮</span> <span>⏪</span> <span>⏩</span> <span>⏭</span> <span>💾</span> </div>					
Title	Title	Description	Description	StartDate	StartD
EstimatedCost	\$1000.00 (EstimatedCost)	Project Manager	ManagerID 🔍	EndDate	EndD
Status	Status 📄	Account	Account 🔍		



## Exercise 8.2: Verifying Pick Lists

In this exercise, you will create pick lists, entities, and variables in the forms so business rules can verify tickets and projects are closed properly.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Test the change

1. Open **Application Architect**.
2. Click *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Click **Run** from the toolbar.

Wait for the project to build, deploy, and open the portal. When the browser opens, minimize Application Architect. The **Log On** page opens.



You can use the no deploy feature for the **ProjectDetails** quick form since you've already built it at least once; however, choose to build and deploy so you can see the status property in your business rule inside Visual Studio for the next exercise.

5. Type *admin* in the **Username** field. No password is required.
6. Click **Sign In**.
7. Expand the **Project Management Nav Bar** and click **Projects**.
8. Open **Project 1**.

Notice the new **Status** pick list is on the form and fully functional within the **ProjectTickets** tab.

9. Select **Open** from the **Status** drop-down list.
10. Click **Save**.
11. Create a backup of the project (**LFS08.backup.zip**) within **Application Architect**.

## Check your understanding



What is a Pick List?



---

---

---

---

---

---

---

---



What are some benefits to using a Pick List?



---

---

---

---

---

---

---

---



# Lesson 9: Creating business rules

## Estimated time

4 hours

## Learning objectives

In this lesson, you will:

- Create a project in Visual Studio to hold external assemblies.
- Add an external assembly into Application Architect.
- Create business rules to use external assemblies.

## Topics

- Setup Visual Studio to create an external assembly
- Add new assembly references into Application Architect
- Add the business rule to be used on the OnChange action of our picklist
- Create a business rule to check the value of a Ticket's status
- Use a business rule on the OnSuccess action of our first business rule
- Use a business rule on the save action of our form

# Creating Business Rules

The ability to create custom validations and other business rules can greatly increase the functionality of Infor CRM. Though creating code snippets within Infor CRM's Application Architect has many uses and can be quite powerful itself, the ability to create external DLLs used as business rules gives even more flexibility in your rule creation. Using a program like Microsoft Visual Studio helps to create dynamic rules or use C# or other languages, which can then be added into Infor CRM.



## Exercise 9.1: Creating an external DLL with business rules

In this exercise, you will add logic to prevent a user from saving the project status as "Closed" or "Cancelled" if any of the project's tickets aren't yet closed and how to accomplish this logic with two different business rules that will be used first on the control and again on an entity event.

**IsProjectStatusClosed:** Checks to see if the user selected Closed or Cancelled from the Project Status pick list (returns true). If true, run the next business rule.

**CanCloseProject:** Checks to see if project tickets have a status of Open or In Progress (returns false). If false, show a message to the user.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



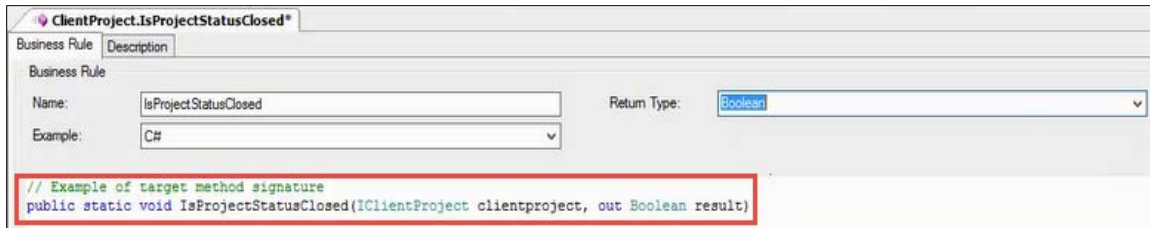
[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Create a business rule and Visual Studio project

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Expand **LFS > Entity Model > Packages > Training > ClientProject > Rules**.
5. Right-click **Business Rule Methods** and select **New Business Rule**.
6. Type *IsProjectStatusClosed* in the **Name** field.
7. Select **Boolean** from the **Return Type** drop-down list. Notice the method signature is automatically filled out for you as you make selections.





8. Copy the signature to **Notepad++**.

9. Click **Save**.

At this point, you need to create the actual assembly this method will call. To do this, you'll use Microsoft Visual Studio to create the assembly outside of Application Architect. By doing so, you can easily preserve this custom logic to make Infor CRM upgrades easier for the customer.

10. Minimize **Application Architect**.

11. Select **Start > All Programs > Visual Studio 2015**.

12. Select **File > New > Project**. The **New Project** window opens.

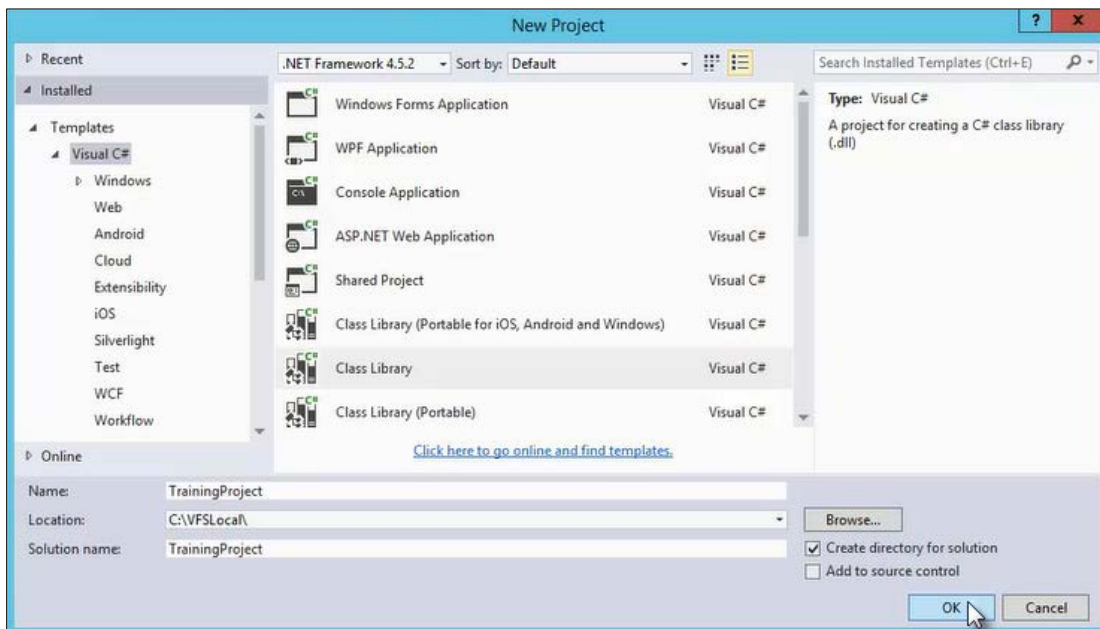
13. Make the following selections:

- .NET Framework 4.5.2
- Visual C#
- Class Library

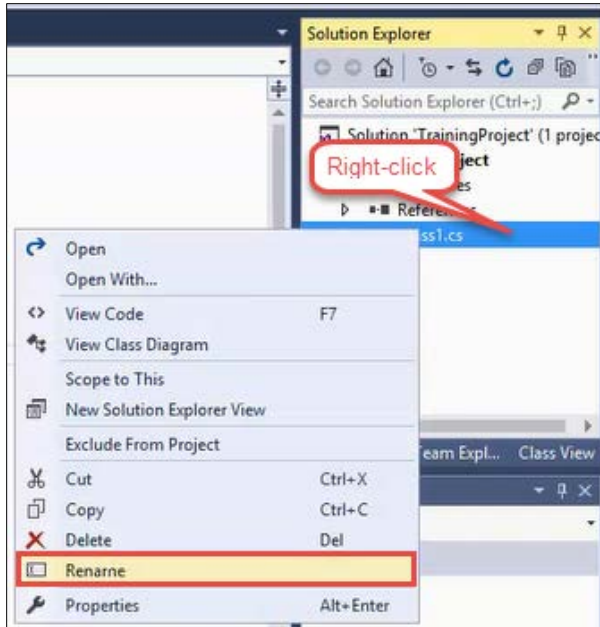
14. Type *TrainingProject* in the **Name** and **Solution name** fields.

15. Select **C:\VFSLocal\** in the **Location** field.

16. Click **OK**. The new project opens in the **Solution Explorer** as **TrainingProject** with a **Class1.cs** class library.



17. Right-click **Class1.cs** and select **Rename** to rename the file to **ClientProject.cs**.



18. Click **Yes** when prompted to rename all project references.



When naming a project, a good practice is to use the name of your company or the company for which you're creating the customization. For this training, we chose "TrainingProject," but we could have used "Phoenix Computers" (the name of our customer) so we keep DLLs separate. Then for the Class name, choose the entity for which all methods inside the class apply. It not only helps to keep the contents of the class library organized, but it also makes it more intuitive to reference the methods using [Entity].[Method] notation.

19. Select **Project > Add Reference** from the menu bar. The **Reference Manager – Training Project** window opens.

20. Click the **Browse** tab and navigate to **C:\inetpub\wwwroot\SlxClient\Bin\**.

21. Select the following files:

- Sage.Entity.Interfaces.dll
- Sage.Platform.dll
- Sage.SalesLogix.BusinessRules.dll

22. Click **Add**.

23. Click **OK**.

24. Paste the method signature you copied from **Application Architect** inside the **ClientProject** class.

25. Modify the signature to include a reference to the **Sage.Entity.Interfaces.IClientProject**. Alternatively, you could add this as a using statement at the top.

26. Type the remaining code to fill out the class.

If you prefer not to type, you can paste the code from the completed **ClientProject.cs** file inside the **ClassFiles** folder. However, Infor strongly recommends you learn from typing.

```

ClientProject.cs
TrainingProject
    TrainingProject.ClientProject
        IsProjectStatusClosed(IClientProject clientproject, out Boolean result)
            using System;
            using System.Collections.Generic;
            using System.Linq;
            using System.Text;
            using System.Threading.Tasks;

            namespace TrainingProject
            {
                public class ClientProject
                {
                    public static void IsProjectStatusClosed(Sage.Entity.Interfaces.IClientProject clientproject, out Boolean result)
                    {
                        result = Convert.ToBoolean(clientproject.Status.ToUpper() == "CLOSED") ||
                            Convert.ToBoolean(clientproject.Status.ToUpper() == "CANCELLED");
                    }
                }
            }

```

```

public static void
IsProjectStatusClosed(Sage.Entity.Interfaces.IClientProject
clientproject, out Boolean result){

result = Convert.ToBoolean(clientproject.Status.ToUpper() == "CLOSED")
|| Convert.ToBoolean(clientproject.Status.ToUpper() == "CANCELLED");
}

```

27. Click **Save All**.

28. Select **Build > Build Solution** from the menu bar.



You created a business rule place-holder in Application Architect then built a new class library that includes code to check whether the Project Status was set to either Closed or Cancelled. You referred to the out-of-the-box IsTicketStatusClosed business rule to create this. When you compiled the library, it became a DLL and PDB file in the default Debug folder for Visual Studio (C:\Documents and Settings\Administrator\My Documents\Visual Studio 2008\Projects\TrainingProject\TrainingProjectbin\Debug). Eventually, these files need to get added into the SlxClient deployment portal (C:\inetpub\wwwroot\SlxClient\Bin).

There are a few options for moving these files into the deployment portal. The first is to add the files in the Application Architect. When deployed, the files move into the appropriate directory. Alternatively, you can manually copy/paste the files into the deployment directory after a build or you can configure settings in Visual Studio to move the final files into the deployment directly after a build. This process doesn't save you much time right now since you still need to build and deploy in Application Architect anyway to reference the new DLL. Moving forward though, if there is a change to the code, you can open it in Visual Studio and build without opening Application Architect.

## Part 2: Configure Visual Studio to perform a post build

Let's utilize the XCOPY command included with all Windows systems. This way, whenever the Visual Studio solution is built, it automatically updates our files within the Web Client.



The directories listed here use the defaults for Microsoft Visual Studio Community 2015 and for Infor CRM's SLXClient deployments. If you're using a different directory, you may need to alter them based on the directories you are using.

1. Select **Start > Run**.
2. Type *cmd* and then press **Enter**.
3. Return to the root "C:\\" directory by typing *cd..* and pressing **Enter** as many times as necessary.
4. Type or paste the first **xcopy** line as shown below and then press **Enter**:

```
xcopy /y "C:\users\administrator\documents\visual studio
2015\Projects\TrainingProject\TrainingProject\bin\Debug\
TrainingProject.dll "
"C:\Inetpub\wwwroot\slxclient\bin\TrainingProject.dll "
```

5. Type *f* when prompted for a file or directory.
6. Right-click, type or paste the second **xcopy** line as shown below, and then press **Enter**:

```
xcopy /y " C:\users\administrator\documents\visual studio
2015\Projects\TrainingProject\TrainingProject\bin\Debug\
TrainingProject.pdb"
"C:\Inetpub\wwwroot\slxclient\bin\TrainingProject.pdb"
```

7. Type *f* when prompted for a file or directory.
8. Close the command prompt when finished.
9. Right-click **TrainingProject** within the **Solution Explorer** and select **Properties**.
10. Click the **Build Events** tab.
11. Paste the following command inside the **Post Build Command Line** field:

```
xcopy /y " C:\users\administrator\documents\visual studio
2015\Projects\TrainingProject\TrainingProject\bin\Debug\
TrainingProject.dll "
"C:\Inetpub\wwwroot\slxclient\bin\TrainingProject.dll "

xcopy /y " C:\users\administrator\documents\visual studio
2015\Projects\TrainingProject\TrainingProject\bin\Debug\
TrainingProject.pdb"
"C:\Inetpub\wwwroot\slxclient\bin\TrainingProject.pdb"
```

12. Click **Save All**.
13. Clear the **Output** window.
14. Close the **Properties** window.
15. Select **Build > Build Solution** from the menu bar.

Notice the copied file messages appear in the Output window.

16. Close **Visual Studio**.



## Exercise 9.2: Creating business rules for closing a project

In this exercise, you will add logic to prevent a user from saving the project status as “Closed” or “Cancelled” if any of the project’s tickets aren’t yet closed and how to accomplish this logic with two different business rules that will be reused first on the control and again on an entity event.

**IsProjectStatusClosed:** Checks to see if the user selected Closed or Cancelled from the Project Status pick list (returns true). If true, run the next business rule.

**CanCloseProject:** Checks to see if project tickets have a status of Open or In Progress (returns false). If false, show a message to the user.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

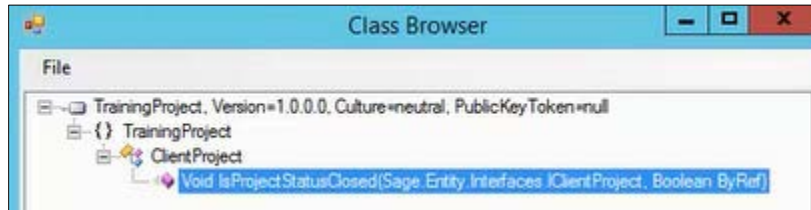
### Part 1: Add the assembly references to Application Architect

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Expand **LFS > Portal Manager > SlxClient > SupportFiles**.
5. Right-click **Bin** and select **Add Existing**. The **Open** dialog box opens.
6. Browse to **C:\inetpub\wwwroot\SlxClient\Bin\**.
7. Press **Ctrl** + click **TrainingProject.dll** and **TrainingProject.pdb**.
8. Click **Add**.
9. Locate **ClientProject.IsProjectStatusClosed** if it’s not already open within the **Project Explorer** by expanding **LFS > Entity Model > Packages > Training > ClientProject > Rules > Business Rule**.
10. Double-click **IsProjectStatusClosed**.
11. Click the **Primary Steps** tab and then click **Add**. The **New Business Rule Step** window opens.
12. Click the **External Assembly** radio button.

The other options allow you to define the method by typing code right here in Application Architect as opposed to saving it in a DLL you created using Visual Studio. The advantage of using an external assembly is to make upgrades easier.

13. Click the **Choose type and method from assembly link**. The **Class Browser** window opens.
14. Select **File > Open**.

15. Browse to the location of the **TrainingProject.dll** and double-click the file. You have the file in both the default **C:\Documents and Settings\...\Debug** folder and the **C:\inetpub\...\Bin** folder in the **SlxClient** portal. Use either location.
16. Expand the **TrainingProject namespace > ClientProject** and select the **IsProjectStatusClosed** method.



17. Click **OK**.
18. Click **OK** again to close the **New Business Rule Step** window.
19. Click **Save All**.
20. Select **Build > Rebuild Web Platform** from the menu bar.

If you receive an “Unable to compile T4 template” error, confirm you have Visual Studio closed. Also, try closing Application Architect and then reopening it.

## Part 2: Use the business rule in the OnChange action of the Status control

1. Expand **LFS > Entity Model > Packages > Training > ClientProject > Forms**.
2. Double-click **ProjectDetails**.
3. Select **View > Properties**.
4. Select the **Status data** field.
5. Expand **OnChangeAction** within the **Misc** node to the right.
6. Select **Action Name** and click the **ellipsis**. The **Action Item Designer** window opens.
7. Select **Validation Business Rule** in the left pane.
8. Select **IsProjectStatusClosed** from the **ValidationMethod** field.
9. Select **OnSuccess** and click the **ellipsis**. A second **Action Item Designer** window opens.
10. Select **Show Message** in the left pane.
11. Type *Testing 1-2-3* in the **Text** field.
12. Click **OK**.

Type anything you want. Since you don't have a second business rule created yet (the rule which checks a ticket's status), confirm this one is firing when the project status is set to closed. Remove the ShowMessage text after you confirm this is working and before starting on the next business rule.

13. Click **OK**.
14. Click **Save All**.

## Part 3: Run the website

1. Click **Run** from the toolbar.

Wait for the project to build, deploy, and open the portal. When the browser opens, minimize the Application Architect. The **Log On** page opens.

2. Type *admin* in the **Username** field. No password is required.
3. Click **Sign In**.
4. Select the **Project Management Nav Bar > Projects**.
5. Open **Project 1**.
6. Select **Closed** from the **Status** drop-down list. The Testing 1-2-3 message should appear indicating the business rule is operating correctly.

The screenshot shows the 'Client Project - Project 1' form in Infor CRM. The form has a blue header with the title and a search bar. Below the header, there are tabs for 'Lookup Results' and 'All Projects'. The main form area contains several fields: 'Title' (Project 1), 'Description' (this is the first proj...), 'StartDate', 'EstimatedCost' (\$1,000.00), 'Project Manager' (Hutchinson, B.), 'EndDate', 'Status' (Closed), and 'Account' (Abbott Ltd.). A 'ProjectContacts' section is visible at the bottom, showing a list of contacts. A blue tooltip or message box is overlaid on the 'Status' field, displaying 'Infor CRM' and 'Testing 1-2-3'. A hand cursor is pointing at the 'Testing 1-2-3' message.

7. Change the status to **Pending** or **Open** to confirm the status doesn't appear.



What happens if it doesn't function as expected?



Infor suggests visiting the Appendix and watching the "Debugging an External Assembly" video at this point to see how to open the SlxClient website in Visual Studio and run it in debug mode. This technique is critical to improving your troubleshooting and debugging skills when working with any code customizations for Infor CRM.

8. Close the browser.





### Exercise 9.3: Creating business rules for ticket status

In this exercise, you will add logic to prevent a user from saving the project status as “Closed” or “Cancelled” if any of the project’s tickets aren’t yet closed and how to accomplish this logic with two different business rules that will be reused first on the control and again on an entity event?”

**IsProjectStatusClosed:** Checks to see if the user selected Closed or Cancelled from the Project Status pick list (returns true). If true, run the next business rule.

**CanCloseProject:** Checks to see if project tickets have a status of Open or In Progress (returns false). If false, show a message to the user.

#### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Add a business rule to check the ticket status value

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Expand **LFS > Entity Model > Packages > Training > ClientProject > Rules**.
5. Right-click **Business Rule Methods** and then click **New Business Rule**.
6. Type *CanCloseProject* in the **Name** field.
7. Select **Boolean** from the **Return Type** drop-down list. The method signature is automatically filled out for you as you make selections.
8. Copy the signature to **Notepad++**.
9. Click **Save**.

At this point, you must create the actual assembly this method will call. To do this, use Microsoft Visual Studio to create the assembly outside of Application Architect. Since you already have a class library created for the ClientProject entity in the TrainingProject solution, you can add a new method to this file.

10. Select **Start > All Programs > Microsoft Visual Studio 2012 > Microsoft Visual Studio 2012**.
11. Select **File > Open > Project**.
12. Open the **TrainingProject** solution.
13. Paste the method signature you built in **Application Architect** inside the **ClientProject** class.
14. Modify the signature to include a reference to the **Sage.Entity.Interfaces.IClientProject** before the clientproject parameter. Alternatively, you can add this as a using statement at the top.



15. Type the remaining code to fill out the class.

If you prefer to not type, paste the code from the completed ClientProject.cs file (includes code comments) inside the unzipped TrainingProject.7z in the ClassFiles folder. Infor strongly recommends you learn from typing though.

```
public static void
CanCloseProject(Sage.Entity.Interfaces.IClientProject clientproject,
out Boolean result)
{
    result = true;

    foreach (Sage.Entity.Interfaces.ITicket ticket in
clientproject.Tickets)
    {
        if ((ticket.StatusCode ==
Sage.SalesLogix.BusinessRules.BusinessRuleHelper.GetPickListItemIdByName
e ("Ticket Status", "OPEN")
|| ticket.StatusCode ==
Sage.SalesLogix.BusinessRules.BusinessRuleHelper.GetPickListItemIdByName
e("Ticket Status", "IN PROCESS"))
        {
            result = false;
        }
    }
}
```

16. Click **Save All**.
17. Select **Build > Build Solution**.
18. Close **Visual Studio**.

## Part 2: Add the assembly references to Application Architect

1. Close and reopen **Application Architect**.
2. Open the **Project Explorer**.
3. Expand **LFS > Entity Model > Packages > Training > ClientProject > Rules > Business Rule Methods**.
4. Double-click **CanCloseProject**.
5. Click the **Primary Steps** tab and then click **Add**. The **New Business Rule Step** window opens.
6. Select the **External Assembly** radio button.
7. Click the **Choose type and method from assembly** link. The **Class Browser** window opens.
8. Select **File > Open**. The **Open** dialog box opens.
9. Browse to **C:\inetpub\wwwroot\SlxClient\Bin\**.
10. Double-click **TrainingProject.dll**.

11. Expand the **TrainingProject namespace > ClientProject**, and then select the **CanCloseProject** method.
12. Click **OK** twice.
13. Click **Save All**.
14. Click **Build Web Platform**.

**WARNING:** If you receive an “Unable to compile T4 template” error, confirm you have Visual Studio closed. Try saving everything in Application Architect, closing Application Architect, and then reopening it.

### **Part 3: Use the business rule in the OnSuccess action in the OnChange action of the Status control**

1. Open the **ProjectDetails** form.
2. Select the **Status** pick list control in the grid.
3. Select **View > Properties**.
4. Expand **OnChangeAction** within the **Misc** area.
5. Select **ActionName** and click the **ellipsis**. The **Action Item Designer** window opens.
6. Select **Validation Rule (IsProjectStatusClosed)** in the left pane.
7. Select **On Success** within the **Misc** area and click the **ellipsis**. A second **Action Item Designer** window opens.
8. Select **Validation Business Rule** in the left pane
9. Select **OnFail** within the **Misc** area and click the **ellipsis**. A third **Action Item Designer** window opens.
10. Select **ShowMessage** in the left pane.
11. Select **Text** within the **Misc** area and type “*You have chosen to close or cancel this project. It has open or in progress tickets. You will not be allowed to save the project until those tickets are closed.*”
12. Click **OK** twice.
13. Expand **OnSuccess** within the **Misc** area and select **Validation Method**.
14. Select **CanCloseProject** from its drop-down list.
15. Click **OK**.
16. Click **Save All**.

### **Part 4: Run the website**

1. Click **Run** from the toolbar.  
Wait for the project to build, deploy, and open the portal. When the browser opens, minimize the Application Architect. The **Log On** page opens.
2. Type *admin* in the **Username** field. No password is required.
3. Click **Sign In**.
4. Select the **Project Management Nav Bar > Projects**.

- Click **Project 1**. The **Project 1 Detail** view opens.
- Select **Closed** from the **Status** drop-down list.

A warning message should appear indicating the business rule is operating correctly.

- Close the message.
- Close the browser.



### Exercise 9.4: Modifying the client project before updating

In this exercise, you will re-use the same business rules and apply the validation on the entity, preventing a user from saving when the project status and ticket status are incompatible.

#### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Modify the ClientProject OnBeforeUpdate event

- Open **Application Architect**.
- Type *admin* in the **Username** field. No password is required.
- Click **OK**.
- Open the **Project Explorer**.
- Expand **LFS > Entity Model > Packages > Training > ClientProject > Rules > Events**.

6. Double-click **OnBeforeUpdate**.
7. Click the **Pre Execute Steps** tab and then click **Add**. The **New Business Rule Step** window opens.
8. Select the **C# Code Snippet** radio button.
9. Click **OK**.
10. Click **Edit Code Snippet**. The code snippet opens in the workspace. Notice the comment in the code - `//TODO: Complete business rule implementation`. This is where you'll add the code.

Type the following code inside the method brackets:

```
Sage.Platform.ChangeManagement.IChangedState objchanges =
clientproject as Sage.Platform.ChangeManagement.IChangedState;
if(objchanges != null)
{
Sage.Platform.ChangeManagement.ChangeSet objchangeset =
objchanges.GetChangedState();
if(objchangeset.HasMemberChange("Status"))
{
if(clientproject.IsProjectStatusClosed())
{
if(!clientproject.CanCloseProject())
{
throw new Sage.Platform.Application.ValidationException("You cannot
close this project until all tickets associated are closed.");
}
}
}
}
```

11. Click **Save All**.

## Part 2: Run the website

1. Click **Run** from the toolbar.  
Wait for the project to build, deploy, and open portal. When the browser opens, minimize the Application Architect. The **Log On** page opens.
2. Type *admin* in the **Username** field. No password is required.
3. Click **Sign In**.
4. Select the **Project Management Nav Bar > Projects**.
5. Click **Project 1**. The **Project 1 Detail** view opens.
6. Set the project status to **Closed**. A warning message should appear telling the user that he/she made an invalid selection and cannot save.

7. Close the warning message and then click **Save**. A new message appears to prevent the save from committing any changes.
8. Create a backup of the project (**LFS09.backup.zip**) in **Application Architect**.

## Check your understanding



What happens when I add validation to a form control and contrast this process with adding validation to the entity itself?



---

---

---

---

---

---

---

---



# Lesson 10: Bundling a customization

## Estimated time

2 hours

## Learning objectives

In this lesson, you will:

- Bundle a customization.
- Install a customization bundle.

## Topics

- Bundling customizations
- Installing bundles for customizations

# Bundling a customization

Infor CRM Web Client bundles are used for installing updates and customizations. Bundles include custom components, pick lists, tables, plugins, and more. Bundles contain both the customizations, referenced in a manifest, and a copy of the manifest. The manifest is a list of all the customizations included within a bundle. Manifests can either be created manually, by change date, by project differences, or by change date. Saved bundles are stored in a .ZIP file and can be password protected as well.



## Exercise 10.1: Creating a manifest and bundle

In this exercise, you will bundle your changes into a .zip file so your customer company can apply these changes to their development database.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



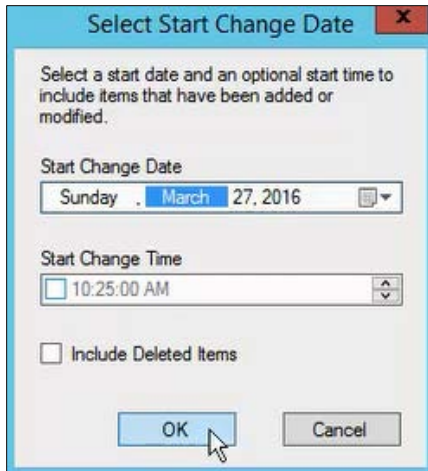
[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Create a new manifest and bundle

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Open the **Project Explorer**.
5. Expand **LFS > Bundle Model**.
6. Right-click **Bundle Model** and select **Create Manifest by Change Date**.
7. Select the date you started working on these changes in the **Start Change Date** field (For this training, select last Sunday).
8. Click **OK**. The manifest opens in your project workspace.





9. Type *TrainingBundle* in the **Name** field.
10. Type *Infor CRM Web Development Training Bundle* in the **Description** field.
11. Click **Save All**.
12. Click the **Bundle Items** tab.

Expand the different nodes in the model to see all the items included as part of the bundle when installed on a new database. These items are either new (custom) or modified (out-of-the-box). The only portal you modified should include the Infor CRM portal. If other portals are listed here, right-click and remove them if you don't want to include them in your bundle.

13. Clear the **Output** window.
14. Right-click the new bundle within the **Project Explorer** and select **Create Bundle**. The **Save Bundle** window opens.
15. Save the bundle to **C:\VFSLocal\**.



## Exercise 10.2: Installing a customization bundle

In this exercise, you will bundle your changes into a .zip file so your customer company can apply these changes to their development database.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

When installing the bundle to another database, Application Architect alerts you to any items conflicting with existing items in the database. This gives you the opportunity to double-check any items in red to confirm you want to overwrite them with the item in the new bundle.

In the training environment, let's see how this works by attempting to install the bundle you just created to the same database. **Note:** You'll cancel the process. These steps are similar to what the Business Analyst would do for installing it at the customer site.

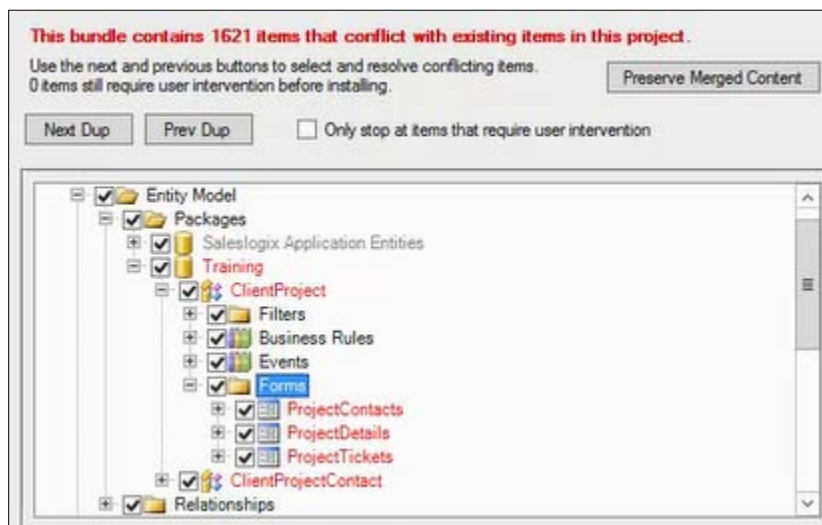
Watch the related video for more explanation of the different merge options when installing a bundle.

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Select **View > Bundle Manager** from the menu bar.
5. Click **Install**. The **Install Bundle** window opens.

**Note:** You may also right-click the project (**LFS**) within the **Project Explorer** and then click **Install Bundle**.

6. Browse to **C:\VFSLocal** and double-click the **TrainingBundle.zip**.
7. Click **Open**.
8. Click **Next**. The **Select Items** screen opens.
9. Expand the various nodes to see the items conflicting with any items in the database.

**Note:** In this case, everything you created or modified should show as red because you're importing in the same database.



10. Click **Next**. The bundle starts to install.
11. Click **Finish**.

## Check your understanding



What are the main actions that a bundle can have?



---

---

---

---

---

---

---

---



What are the different ways you can create a manifest?



---

---

---

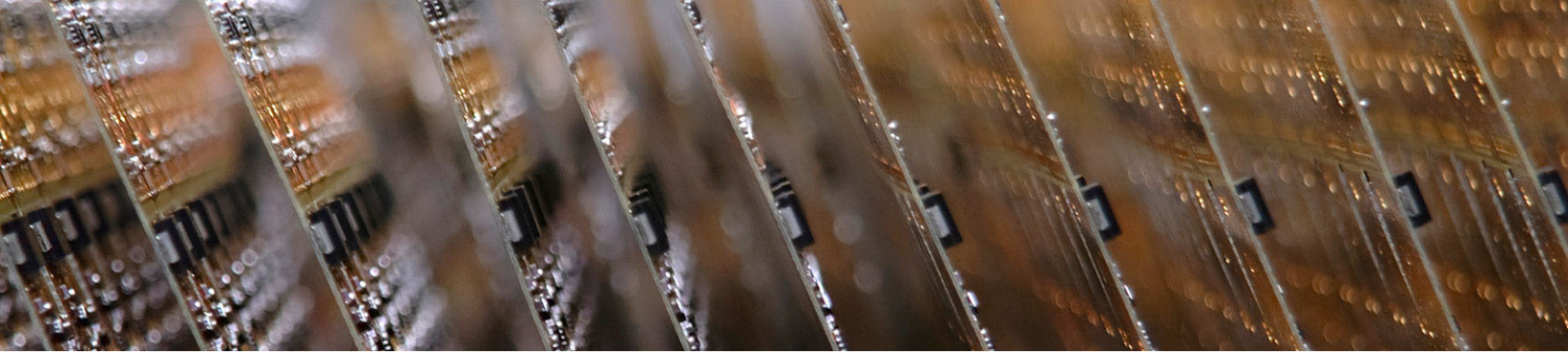
---

---

---

---

---



# Lesson 11: Customizing the Customer portal

## Estimated time

2 hours

## Learning objectives

In this lesson, you will:

- Deploy the Customer Portal.
- Customize the Customer Portal interface.

## Topics

- Deploy the Customer Portal
- Grant web access to a customer
- Create lookups within customer facing pages

# Customer portal

The Infor CRM Customer Portal is a limited version of the Web Client which allows you to grant your customers access to certain parts of your Infor CRM system, such as viewing and adding tickets. The Customer Portal can also be customized to allow for more robust access to your Infor CRM Web Client. Adding pages to allow access to your customized entities, adding additional fields to out-of-the-box pages, and nearly any other customization can be added to the Customer Portal.



## Exercise 11.1: Customizing the Customer Portal

In this exercise, you will prepare the Customer Portal for customization.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Deploy the Customer Portal

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Select **Tools > Web User Security**.
5. Select **Administrator** from the **Web Access User** drop-down list.
6. Click **OK**.



7. Select **View > Deployment Explorer**.
8. Expand **Deployments** and double-click **Customer Portal**.
9. Select the **SlxCustomerPortal** within the project workspace under **Deployment Targets > IIS**.
10. Select the **Use this deployment for debugging** check box.
11. Change the following IIS Target Settings:

- **Port:** 3333
- **App Pool:** Saleslogix



By default, when you install the Web Host on IIS from your installation media, an Application Pool is created under the name “Saleslogix”. On this training environment, an identical Application Pool was created called “InforCRM.” This way, any new user or IIS Admin will know the Application Pool is used by Infor CRM.

12. Select the **Startup Portal** check box within the **IIS Portal Configuration** area.
13. Click **Save All**.
14. Click **Run**. This process automatically includes three steps:
  - **Builds the Web Platform** - builds all the interface items and compiles all quick forms into .ascx files). These files are saved in a temporary location on the hard drive (C:\Users\Administrator\AppData\Roaming\Sage\Platform\Output).
  - **Deploys the core portals** - builds the .aspx Web pages for the site and stores them in the designated deployment directory on the Web Server). In this case, you’re developing on the Web server. The deployment directory is C:\inetpub\wwwroot\slxcustomerportal.
  - **Opens the startup portal** - opens a Web browser directed at the Infor CRM Web Log On screen. The URL for the Web client is http://SRVXX:3333/SlxCustomerPortal.
15. Minimize **Application Architect** when the browser launches.
16. Verify the **Infor CRM Customer Portal Log On** screen opens.

Let’s create a username and password for one of the contacts.

## Part 2: Add Web User access to a contact

1. Select **Start > All Programs > Saleslogix > Saleslogix Client** to open the Infor CRM Windows client. The feature to add Web User access is also available using the Web Client interface.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Open the **Contact Detail** view for **Abbott, John**. John is used because there’s a project already created for the Abbott Ltd. account. This is important for when you test the Customer Portal project filter later.
5. Click the **More Tabs** tab and then click **Web Access**.
6. Select the **Web Access** check box.
7. Type *John* in the **User Name** field.
8. Type *password* in the **New Password** and **Repeat New Password** fields.
9. Click **Save**.
10. Close the **Saleslogix Client**.

## Part 3: Sign in to the Customer Portal interface

1. Open a new browser window and browse to **http://SRVXX:3333/slxcustomerportal**.
2. Type *John* in the **User Name** field.
3. Type *password* in the **Password** field.

4. Click **Sign In**.
5. Explore the Customer Portal interface and refer to the related video.



If you cannot log on, open the connection.config in the customer portal in C:\wwwroot\inetpub\slxcustomerportal, and remove the following parameter:

```
<Password><![CDATA[40E8031E53849A4DB91A]]></Password>.
```



## Exercise 11.2: Allowing customers to add a ticket to a project

In this exercise, you will customize the Customer Portal to allow a customer to associate a project to a ticket.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Create the Project lookup

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Open the **Project Explorer**.
5. Expand **LFS > Entity Model > Packages > Saleslogix Application Entities > Ticket > Forms**.
6. Double-click **AddPortalTicket**.
7. Right-click inside an empty cell on the form and select **Rows > Add Row**.
8. Move the **Submit Ticket** button to the now new bottom row.
9. Right-click inside an empty cell above the **Submit** button and select **Insert > Lookups > Lookup**. The **Lookup Control Setup Wizard** opens.
10. Set the following properties:
  - **What information will you be looking up?** Client Project
  - **Which property will receive the result of this lookup?** ClientProject
  - **Enable Hyperlinking to the detail page for this result?** Select
  - **Select how the lookup control will display:** Dialog
  - **Select the properties that you want to see in the result grid:** Title
11. Click **OK**. The control is added to the form but doesn't have a caption.

12. Select **View > Properties**.
13. Select the new lookup in your form.
14. Select **Caption** within the **Appearances** area and type *Project* in the field to the right of it. When you click out of the field, the caption appears on the form.
15. Select **SeedProperty** within the **Misc** area and type *Account.Id* in the field to the right of it.
16. Select **Data Bindings** within the **Appearance** area and click the **ellipsis**. The **Bind Control Properties** window opens.
17. Expand the **Account** node in the left pane and select **Id**.
18. Select **SeedValue** in the right pane and click **Map**.
19. Expand the **ClientProject** node in the left pane and select **Title**.
20. Select **Text** in the right pane and click **Map**.
21. Click **OK**. Your form should resemble the following figure:

The screenshot shows the 'Quickform Details' window. At the top, there's a blue header with the title 'Quickform Details' and a link 'Click to modify form resources'. Below the header is a grey bar with a save icon. The main area contains a form with the following fields:
 

- Area:** A text field with the value 'Area' and a magnifying glass icon.
- Urgency:** A text field with the value 'Urgency' and a magnifying glass icon.
- Subject:** A text field with the value 'Subject'.
- Description:** A text field with the value 'TicketProblem.Notes'.
- Project:** A text field with the value 'ClientProject, Account.Id'. This field is highlighted with a blue border.

 At the bottom right, there is a legend titled 'QFControlsList (1 controls)' which includes 'cmdSubmit'.

22. Click **Save All**.

## Part 2: Run the website

1. Click **Run** from the toolbar.  
Wait for the project to build, deploy, and open portal. When the browser opens, minimize the Application Architect. The **Log On** screen opens.
2. Type *John* in the **User Name** field.
3. Type *password* in the **Password** field.
4. Click **Sign In**.
5. Click **Add Ticket**.
6. Click the **Project** lookup and verify that only the project for which the **Abbott Ltd.** account appears in the dialog.
7. Click **Cancel**.
8. Click **Save**.
9. Close the browser.
10. Create a backup of the project (**LFS08.backup.zip**) within **Application Architect**.



## Check your understanding



What's the purpose of a portal?



---

---

---

---

---

---

---

---



What happens when I set the data bindings of a control?



---

---

---

---

---

---

---

---



# Course summary

## Estimated time

5 minutes

## Learning objectives

Now that you have completed this course, you should be able to:

- Convert customer requests into project requirements.
- Identify the entities and relationships that must be included in the project and as they apply to the requirements.
- Identify interface items and business rules that must be included in the project and as they apply to the requirements.
- Categorize project tasks for customer sign-off.
- Configure a web development environment for Infor CRM.
- Describe a scenario in which you could take advantage of the no deploy process in the Application Architect.
- Describe what happens when making a change to a form using the Form Designer in the Web Administrator.
- Discuss common properties of a new entity and identify which ones are needed to create a relationship with another entity.
- Summarize the purpose of smart parts and pages.
- List components included in a main view.
- Identify what OnClickActions are available for a control button.
- Explain what happens during a full build and deploy versus the no deploy process.
- Discuss what happens when adding validation to a form control and contrast that process with adding validation to the entity itself.
- Explain the purpose of a portal.
- Discuss what happens when setting the data bindings of a control.

## Topics

- Course review



# Appendices

The following are included in this section:

- Appendix A: User accounts
- Appendix B: Frequently asked questions
- Appendix C: Incremental and full builds
- Appendix D: Additional debugging techniques
- Appendix E: Additional exercises

## Appendix A: User accounts

Your instructor will assign you a student user ID from the table listed below to use for class exercises.

**Note:** If you are taking this course as self-directed learning, refer to the Training Desktop Login Instructions on the Lab On Demand page.

Application	User name	Password
SQL	sa	password
SQL	sysdba	Ma\$t3rk3y
RDP	administrator	G!oba!08
RDP	Infor	!nfor08
Infor CRM	admin	<blank>

## Appendix B: Frequently asked questions

In a technical training class, it's easy to get lost in the details. Our goal is to make sure you not only understand the details of how something works but also WHY you need to know them. These FAQs are a good way to review topics not specifically addressed during the course.



What's the difference between a data link and a data connection?



Infor CRM uses the native Object Linking and Embedding for Databases (OLE DB) and its own custom Saleslogix OLE DB Provider to access the back-end database. On the server side, there's one server-side connection. This connection is created using the Saleslogix Connection Manager and is used by the Saleslogix OLE DB Provider. On the client side, there's one client-side connection for each client that accesses the Saleslogix database. This connection is created using the Data Link Manager.



In which situations would you want to export a project workspace? What is the main difference between an exported project workspace (a model) and the Virtual File System (VFS)?



Working with a Model (a project exported to the hard drive or network share) is easier to troubleshoot and may build and deploy faster. You also need a Model when using a source control utility. If you're not using source control, keeping the project inside the VFS allows you to back it up whenever the database is backed up. The main difference between a Model and the VFS is the metadata for your project is included in your Model when you export it; with the VFS, the metadata for your project is contained inside the database.



What happens when I do a project backup?



When creating a backup, all your project files are taken from your VFS or Model and moved to a zipped backup file. Creating a backup is the same as making a copy of your local Model (if you exported the project) except that a backup is zipped and you can restore it from within the Application Architect.



With deployments, what is the difference between clicking the Run button and deploying manually (Build Web Platform, Deploy, Open Portal)? Why would you use one over the other?



Clicking Run performs all three steps (build, deploy, open portal) but also is different in that Run uses the ASP.NET Development server as opposed to IIS. This difference is important for troubleshooting because you can bypass your IIS Server. After you finish troubleshooting, you should test your deployment using the Open Portal option before deploying to production to make sure your customizations work on your IIS Server with its configurations.



Other than the letter “p,” how is an .aspx file different from an .ascx file?



An .aspx file represents a page. It's what you can use to navigate to a page inside a browser URL. The .ascx file represents a user control also known as a smart part. This is an element that gets placed inside of a page and told how to display. You cannot browse to an .ascx file directly without placing it on a page.



What happens during a build process? What happens during a deploy process?



Well, a lot! The key things to remember are that a build process compiles code, interface items or forms into usable smart parts. There are many types of builds you can run. The deploy process on the other hand creates the actual pages and moves all the contents of the portal into the designated deployment folder on the Web server.



When you create a new entity, what two methods are always available for you?



Save and delete.



When you create a new page, you have 3 options: Portal Page, MainView, and Entity Page. What is the difference? If you create a page as one type, can you later change it to another? If so, how?



A MainView page is tied to an entity and it has the Lookup/Group functionality. An Entity page is tied to an entity and a Portal page is blank. Yes, a page can be changed to another page type just by adding the new functionality to the page. To add the Lookup/Group functionality, for example, add the LiveGroupViewer to your smart part. To bind to an entity, for example, use the Configure Base button on the Advanced tab of a page.



When changing code for a portal through Visual Studio, why do you have to bring those changes back into the Application Architect?



When you run the SlxClient portal from the deployment folder, everything you deploy from within the Application Architect will overwrite those files; therefore, you need to first put the changes into Application Architect before deploying again.



What are some differences between the Code Snippet Action Item and the C# Snippet Action Item?



The code snippet action item allows VB.NET, C#, or an external assembly. If you choose C# or VB.NET your code is compiled into Sage.SnippetLibrary.CSharp.dll. The C# snippet action item places your C# code into the quick form.



When you create an external assembly that will be tied to a business rule, what gives you access to the entity?



When creating an external assembly, you can have the entity interface such as IAccount or IContact in the signature and parameters of your method.



What does the Sage.Platform.ChangeManagement namespace allow us to do?



You can determine if an entity instance has been changed, as well as the old value and new value for a property.



After creating a bundle in the Application Architect, how would you apply the bundle? Can you apply a Web bundle using the Saleslogix Administrator?



Apply the bundle in another instance of the Application Architect. You cannot apply a Web bundle using the Saleslogix Administrator.



## Appendix C: Incremental and full builds

Within Application Architect there are two primary build options: Build Interfaces and Build Web Platform. By default, both options use an incremental build. This appendix will explain how you can do a full build instead of an incremental build if one is necessary.



### Exercise C.1: Forcing a full build

In this exercise, you will locate and debug an error in your Infor CRM customization.

#### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

### Exercise steps

To learn debugging concepts, let's create a sample form in the Application Architect to play with so you don't taint any of the actual website forms.

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Open the **Project Explorer**.
5. Expand **LFS > Entity Model > Packages > Training > ClientProject**.
6. Right-click **Forms** and double-click **ProjectDetails**.

Here are a few ways to force a full build:

- Select **Build > Rebuild Web Platform**.
  - Select **Build > Clean Build Folders** and click **Build Web Platform**.
  - Press **Ctrl** and click **Build Web Platform**.
7. Open the **Output** window.
  8. Press **Ctrl** and click **Build Web Platform**. Notice it's running more than an incremental build which only builds changes you've just made to the ProjectDetails form if you had made a change.



# Appendix D: Additional debugging techniques

This course touches on basic debugging techniques in Lesson 2. If you'd like more practice with debugging inside Visual Studio, use this section as a guide.



## Exercise D.1: Troubleshooting a build error

In this exercise, you will locate and debug an error in your Infor CRM customization.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Open the **Project Explorer**.
5. Expand **LFS > Entity Model > Packages > Training > ClientProject > Rules > Events**.
6. Double-click **OnAfterDelete**.
7. Click the **Primary Steps** tab and then click **Add**.
8. Select the **C# Code Snippet** radio button.
9. Click **OK**.
10. Select the new **OnAfterDeleteStep1** and click **Edit Code Snippet**.
11. Add the following line to the bottom of the code:

```
Int i = 0
```

12. Click **Save All**.
13. Close the step and **OnAfterDelete** event.
14. Click **Run**.

Notice in the Output Window, there is a line in red font telling you about the error, and a second line of red font with the information "Build Failed." Double clicking on the line that gives you the information about the error will open the file containing that error in your project workspace.

15. Click **OK**.
16. Double-click the red error line that lists the location of the error.
17. Delete **int i =0**.

18. Clear the **Output** window.
19. Click **Build Web Platform**. Notice the error no longer appears.



## Exercise D.2: Debugging with Visual Studio

In this exercise, you will locate and debug an error in your Infor CRM customization.

### Notes:

- If you are taking this course as classroom or virtual instructor-led training, observe as your instructor first demonstrates this exercise.
- If you are taking this course as self-directed learning, complete the steps below.



[Click here to view a demo and/or practice this task](#)

## Exercise steps

### Part 1: Set up the debugging scenario (C# snippet action item)

To learn debugging concepts, let's create a sample form in the Application Architect to play with so you don't taint any of the actual website forms.

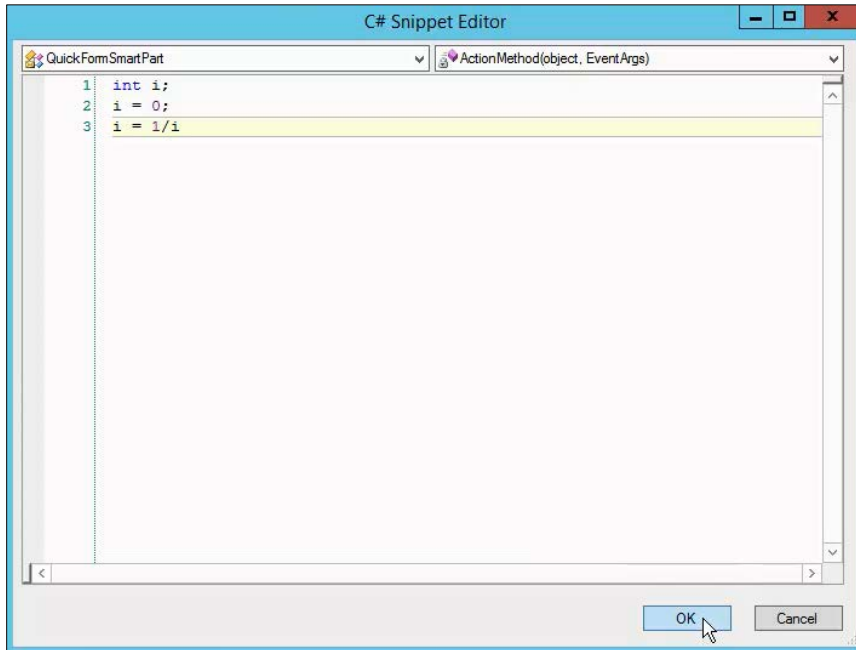
1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Open the **Project Explorer**.
5. Expand **LFS > Entity Model > Packages > Saleslogix Application Entities > Account > Forms**.
6. Right-click **Forms** and select **New Form Wizard**. The **New Form Wizard** opens.
7. Select the **Create Detail Form** radio button.
8. Type *AccountDebug* in the **Form Name** field.
9. Click **Next**. The **Properties** screen opens.
10. Select the **AccountName (Account Name)** property.
11. Click **Next**. The **Details Form Layout** screen opens.
12. Verify the suggested column number is **1**.
13. Click **Next**. The **Summary** screen opens.
14. Click **Finish**. The form opens in the workspace.
15. Right-click in the top, gray area of the workspace and select **Add Control Button > Button**.
16. Select **View > Properties**.
17. Select the new **ab** button in your workspace.
18. Select **Caption** within the **Appearance** area and type *Debug* in the field to the right of it.

19. Click **Save**.
20. Select **On Click Action** within the **Misc** area and then select **Action Name** and click the **ellipsis**. The **Action Item Designer** window opens.
21. Select **C# Snippet Action Item** in the left pane.
22. Select **CSharpCodeSnippet** in the right pane and click the **ellipsis**. The **C# Snippet Editor** window opens.
23. Type the following code:

```
int i;  
i = 0;  
i = 1/i
```

You're intentionally adding two errors in your code: A syntax error (missing semi-colon) and a logical error (division by zero).

24. Click **OK**.



25. Click **OK** again.
26. Click **Save All**.
27. Select **Build > Build Snippet Libraries**.

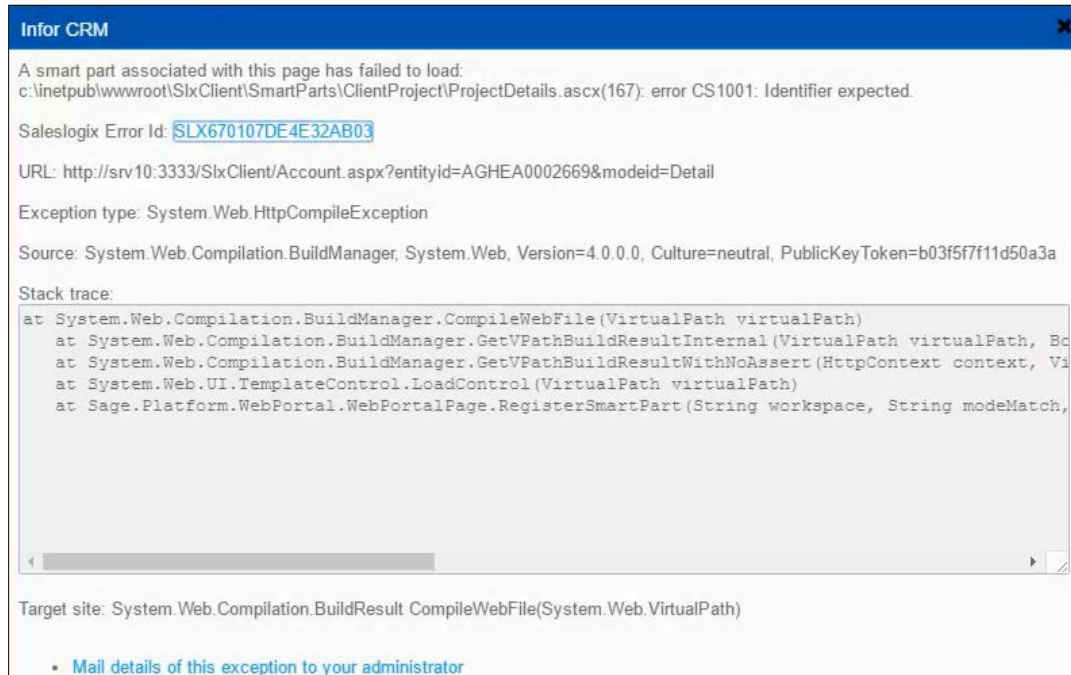
Notice no errors show in your Output Window even though you know they exist. You'll contrast this error output with the Code Snippet Action Item later.

28. Click **Build Interfaces**.

## Part 2: Add our AccountDebug form to the AccountDetail page

1. Open the **Project Explorer**.
2. Expand **LFS > Portal Manager > SlxClient > Pages**.

3. Double-click **Account Detail**.
4. Click the **Smart Part** tab and then click **Add Smart Part**. The **Select Smart Part** window opens.
5. Expand **Account** and then select **AccountDebug**.
6. Click **OK**.
7. Verify the **Target Workspace** field is set to **TabControl**.
8. Move the **AccountDebug** smart part to the top of the list.
9. Click **Save All**.
10. Click **Run**.
11. Open a browser to deploy the **SlxClient** portal and browse to **http://srvxx:3333/slxclient**.
12. Type *admin* in the **Username** field. No password is required.
13. Click **Sign In**.
14. Open the **Account Detail** view for **Abbott Ltd**. The Infor CRM website throws an error:



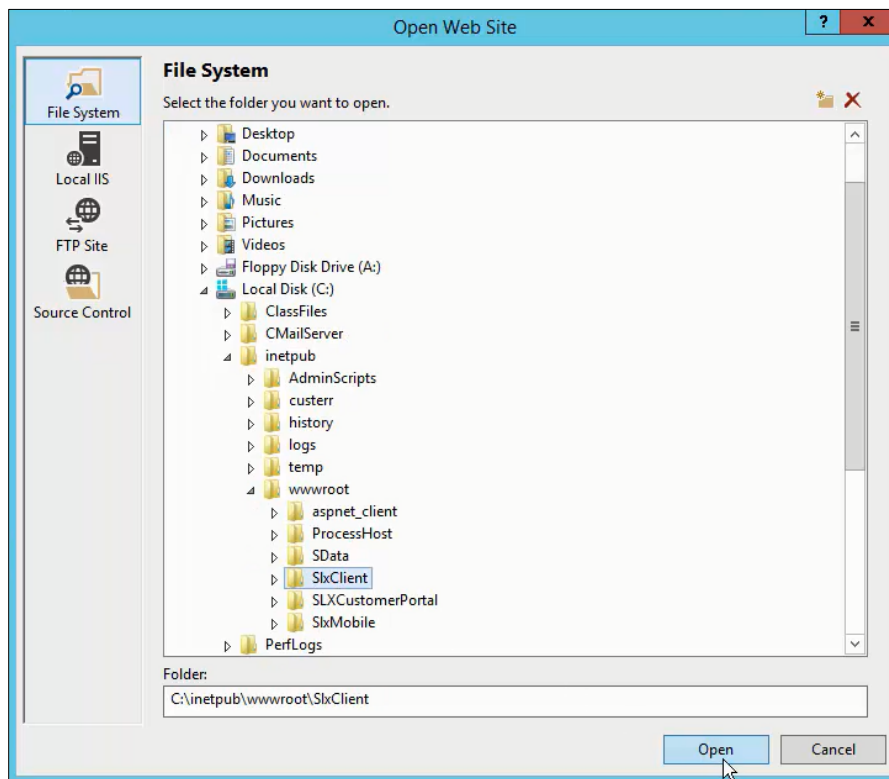
Now you have your scenario. The website throws an error and you need to figure out what's wrong. If you can't easily diagnose it in Application Architect, let's see how you can debug from Visual Studio.

### Part 3: Open the SlxClient website in Visual Studio

Opening the **SlxClient** website in **Visual Studio** allows you to take advantage of more debugging options. You can also rely on the "no deploy" benefit. When you're finished testing inside **Visual Studio**, you can add the changes back into **Application Architect**.

1. Close the **Infor CRM Web Client**.
2. Close **Application Architect**.
3. Select **Start > All Programs > Visual Studio 2015**.

4. Select **File > Open > Web Site**. The **Open Web Site** dialog box opens.
5. Click **File System** and browse to **C:\inetpub\wwwroot\SlxClient**.
6. Click **Open**. The **SlxClient** website opens in the **Solution Explorer**.



7. Select **Website > Start Options** from the menu bar.
8. Select **Build** in the left pane.
9. Select **No Build** from the **Before running startup page** drop-down list.
10. Click **Apply**.
11. Click **OK**.

#### Part 4: Edit a smart part in Visual Studio

1. Expand **SmartParts > Account** within the **Solution Explorer** and double-click **AccountDebug.ascx**.



If a prompt appears for an Inconsistent Line Endings message, clear the **Always show this dialog** check box when prompted and then click **No**.

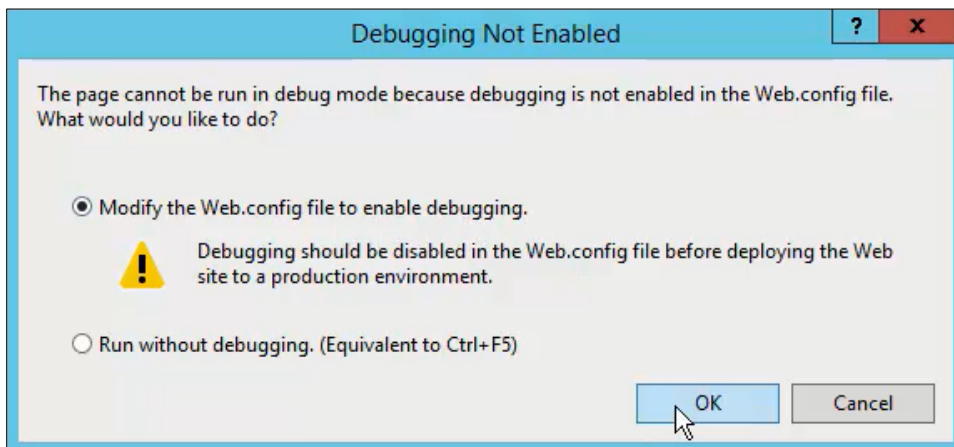
2. Search for **Line 64** or the red indication block within the **AccountDebug.acsx** code as per the error message.
3. Add a **semi-colon** to the end of the third line.

```
protected void tbrButton_ClickAction(object sender, EventArgs e) {
    int i;
    i = 0;
    i = 1 / i;
}
```

4. Click **Save All**.

#### Part 5: Start/stop the debugging website

1. Select **Debug > Start Debugging** or press **F5**. A **Debugging Not Enabled** dialog box opens.
2. Select the **Modify the Web.config file to enable debugging** radio button.
3. Click **OK**.

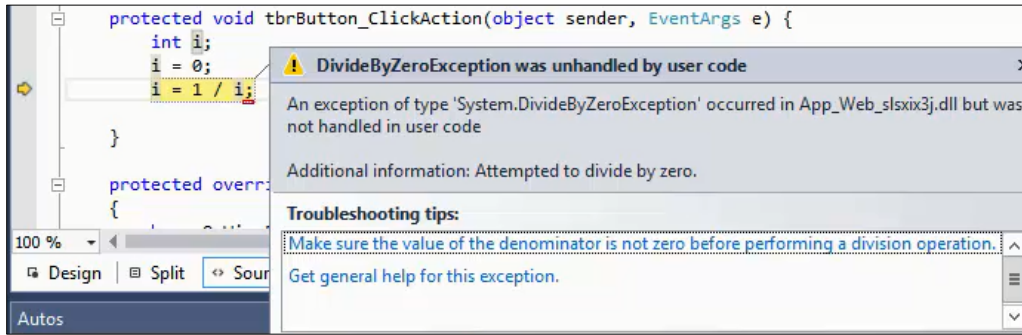


In your production environment, debugging shouldn't be enabled. This is a common setting for websites. When you click OK, the `<compilation debug>` value in the web.config will be set to "true."

The Infor CRM Web Client loads in a new browser window. Notice the URL points to the ASP.NET local development server. You could also browse to `srvxx:3333/slxclient`. Also, notice that a message appears indicating you have ASP.NET Debug Mode on.

4. Type *admin* in the **Username** field. No password is required.
5. Click **Sign In**.
6. Open the **Account Detail** view for **Abbott Ltd**. The error no longer appears.
7. Click the **AccountDebug** tab.
8. Click **Debug**. The `OnClickAction` is invoked and you are taken to the line item in Visual Studio that is throwing a `DivideByZero` error.





9. Select **Debug > Stop Debugging** or press **Shift+F5**.
10. Change the variable from **i = 0;** to **i = 1;**
11. Click **Save**.

Let's run the website again but this time you'll include a breakpoint so you can step through the code line-by-line at your breakpoint and see the values returned.

#### Part 6: Set a breakpoint and step through code

1. Click within the gray bar around the **int i;** code to add a breakpoint.
2. Right-click the breakpoint and select **Breakpoint > Insert Breakpoint**. The **File Breakpoint** window opens.
3. Add another breakpoint to the **i = 1;** code if one is not already created.
4. Click **Save All**.
5. Right-click the breakpoint for **i = 1;** and select **Conditions**.
6. Click the **Location** link.
7. Select the **Allow the source code to be different from the original version** check box.
8. Click **Close**.
9. Click **Save**.
10. Select **Debug > Start Debugging** or press **F5**. A new browser tab opens with the Infor CRM website.
11. Type *admin* in the **Username** field. No password is required.
12. Click **Sign In**.
13. Open the **Account Detail** view for **Abbott Ltd**.
14. Click the **AccountDebug** tab and then click **Debug**.

An error message appears indicating the source file is different from when the module was built.

15. Click **Yes**.

Visual Studio brings you to the breakpoint in the code. At this point, you can inspect and modify different variables. Hover over the variable to see the current value.

16. Right-click the **i** variable within the **i = 1;** code and then click **Add Watch** window.

Notice you can see the variable's current value. Another method to inspect a variable's current value is to use the **Immediate** window to query or set the value to something else.

17. Press **F11** to step into the code. Notice the variable has been set to **i = 1;**

The value is also updated in the **Watch** window. Now you can be confident the error is resolved.

18. Click **Continue** to return to the website in debug mode.
19. Stop debugging in **Visual Studio 2015** by pressing **Shift+F5**.
20. Close **Visual Studio 2015**.
21. Close the **Infor CRM Web Client**.

## **Part 7: Add changes into the Application Architect**

1. Open **Application Architect**.
2. Type *admin* in the **Username** field. No password is required.
3. Click **OK**.
4. Open the **AccountDebug** quick form.
5. Select the **ab** button within your workspace.
6. Expand **On Click Action** within the **Misc** area.
7. Select **Action Name** and click the **ellipsis**. The **Action Item Designer** opens.
8. Select **CSharpCodeSnippet** in the right pane and click the **ellipsis**. The **C# Snippet Editor** opens.
9. Correct the following errors in the code:
  - Set **i** equal to **1**
  - Insert a **semi-colon** to the end of the **i = 1/i** line
10. Click **OK** twice.
11. Save the form.
12. Select **Build > Clean Build Folders** from the menu bar.
13. Click **Run**.

# Appendix E: Sample project exercise

The below scenario offers a sample project based on a set of guidelines for those students who may want some additional hands-on training with Infor CRM. It utilizes a set of requirements that must be met to meet a client's needs.

This exercise is optional and not part of the main Infor CRM Developing Web training. However, it can be a valuable resource in learning more about the connectedness of the different aspects of Infor CRM customization and development that have been taught throughout this course.



For these exercises, continue using the Training package created earlier in class.

## Exercise goal:

Given a set of requirements, successfully build a set of customizations that satisfies all requirements without step-by-step instructions.

## Exercise requirements:



### Scenario

A customer has requested a custom payment tracking system for Infor CRM. After consulting with the customer, you have discovered the following requirements:

- A Payment must include the following:
  - Amount
  - Date
  - Account
  - Contact
  - Status
  - Invoice Number (maximum 20-character Alphanumeric)
- Payments can have either of the following for status:
  - Full
  - Partial
- An account can make many payments.
- A contact can also make many payments.
- When viewing an account a user should be able to:
  - View all payments made by the Account
  - Link to and view the details for a specific Payment
  - Create a new Payment
  - Edit an existing Payment
- When viewing a contact a user should be able:
  - View all payments made by the Contact
  - Link to and view the details for a specific payment

- Create a new Payment.
  - Edit an existing Payment
- Users should also be able to view a list of all Payments made for all Accounts they have access to.
- The New menu will need an option to create a Payment from it.
- A navigation bar should be created for payment processors with access to:
  - Payments
  - Accounts
  - Contacts
  - Tickets
- The new navigation bar should include context menus.