

CA Plex

Tutorial for Windows

r7.1



This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the "Documentation") is for your informational purposes only and is subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and may not be disclosed by you or used for any purpose other than as may be permitted in (i) a separate agreement between you and CA governing your use of the CA software to which the Documentation relates; or (ii) a separate confidentiality agreement between you and CA.

Notwithstanding the foregoing, if you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2014 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At <http://ca.com/support>, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at <http://ca.com/docs>.

Contents

Chapter 1: Introduction 9

Requirements	9
Additional Prerequisites for Windows Vista and Windows Server 2008	9
How Long Does It Take?	9
Using Online Help	10
Conventions in This Guide	11

Chapter 2: Your First Application in 20 Minutes 13

What Is CA Plex?	13
Reusable Business Patterns	13
Workgroup Development	14
Code Generation	14
Project Management Application	15
Opening the Tutorial Model	16
Objects and the Object Browser	16
CA Plex Object Types	17
Viewing the Object Browser	18
Defining the Project Entity	19
Specifying Attributes for the Project Entity	20
Defining Field Properties	23
Using Continuation Triples to Make Fields Optional	29
Using Inheritance to Define the Project Entity	31
CA Plex Feature: Patterns and Inheritance	32
Adding Functionality to the Project Entity	33
Generating and Building the Project Entity	36
CA Plex Feature: Native Platform Implementation	37
Checking for Errors	38
Checking for C++ Build Errors	38
Using Your Generated Application	39
Running the Project.Edit Function	39
CA Plex Feature: Default Panel Layouts	39
Adding a New Project	40
CA Plex Feature: 100% Code Generation	41
Preserving Data	41
Introducing the Panel Designer	42
CA Plex Feature: Visual Development	42

Opening the Panel Designer.....	43
Defining a Multi-line Edit Control	46
Adding Spin Controls.....	50
Regenerating the Function.....	51
Chapter Review	53

Chapter 3: Modeling Entity Relationships 55

Introducing the Diagrammer.....	55
Creating a Diagram.....	56
Defining Relationships with the Diagrammer	59
CA Plex Feature: Dynamic Diagrams	61
Defining the Task Entity	61
Generating and Building the Employee Entity	63
Adding Employee Records.....	64
Chapter Review	65

Chapter 4: Creating a Wizard 67

Defining the Wizard.....	68
Running the Basic Wizard	70
Modifying the Wizard Parent Panel	73
Action Diagrammer	74
Action Diagram Window	75
Action Diagram Palette	75
Variable Palette.....	76
Modifying the First Part of the Wizard.....	77
Adding a Field to a Variable	78
Updating the Shared Data.....	80
Making the Next Button Add a Record	82
Checking Your Modifications	84
Hiding the Apply Button.....	86
Modifying the Second Part of the Wizard.....	87
Adding Project ID to the SharedData and Restrict Variables	87
Defining Restrictor Processing	89
Restricting the Display of Tasks.....	89
Adding Commit and Rollback Processing	91
Generating and Building.....	93
Testing Your Wizard	93
CA Plex Feature: Automatic Selector Functions.....	95
Chapter Review	96

Chapter 5: Creating a Property Sheet **99**

What Is a Property Sheet?.....	99
Defining the Property Sheet.....	100
Creating a Function to Work with Projects and Tasks.....	101
Adding Action Diagram Code	106
Modifying the Panel	109
Adding the Function to the Property Sheet	112
Modifying the Parent Panel	112
Setting the Tab Text	115
Setting the Caption.....	116
Generating and Building.....	117
Testing Your Property Sheet	117
Chapter Review	120

Chapter 6: Putting It All Together **123**

Creating a Function to Tie the Application Together	123
Modifying the Project Manager's Panel	124
Creating and Mapping Logical Events	126
Event Handling in the Action Diagram	128
Calling the Wizard	128
Modifying the Property Sheet.....	130
Calling the Property Sheet	131
Generating and Building.....	134
Creating an Executable Program.....	134
Stopping the Select Data Source Dialog Appearing	135
Testing the Application	136
Chapter Review	137

Appendix A: The Tutorial Reference Model **139**

Using the Tutorial Reference Model	139
What Does It Contain?	140
What Does It Look Like?	143
Generating and Building.....	147

Index **149**

Chapter 1: Introduction

CA Plex is an e-business application development environment that enables teams of software developers to design and implement business applications.

This tutorial shows you how to create a simple Windows application that accesses data using open database components (ODBC). CA Plex supports a wide variety of platforms, but Windows-ODBC was chosen for this tutorial because little or no configuration is required (see the following requirements).

This section contains the following topics:

[Requirements](#) (see page 9)

[How Long Does It Take?](#) (see page 9)

[Using Online Help](#) (see page 10)

[Conventions in This Guide](#) (see page 11)

Requirements

To complete this tutorial you need to install:

- CA Plex r7.0
- Microsoft Visual Studio 2005 Standard edition or higher

If you do not have Visual Studio 2005 Standard edition or higher installed, you can still work through the tutorial but you cannot compile or run the application.

Additional Prerequisites for Windows Vista and Windows Server 2008

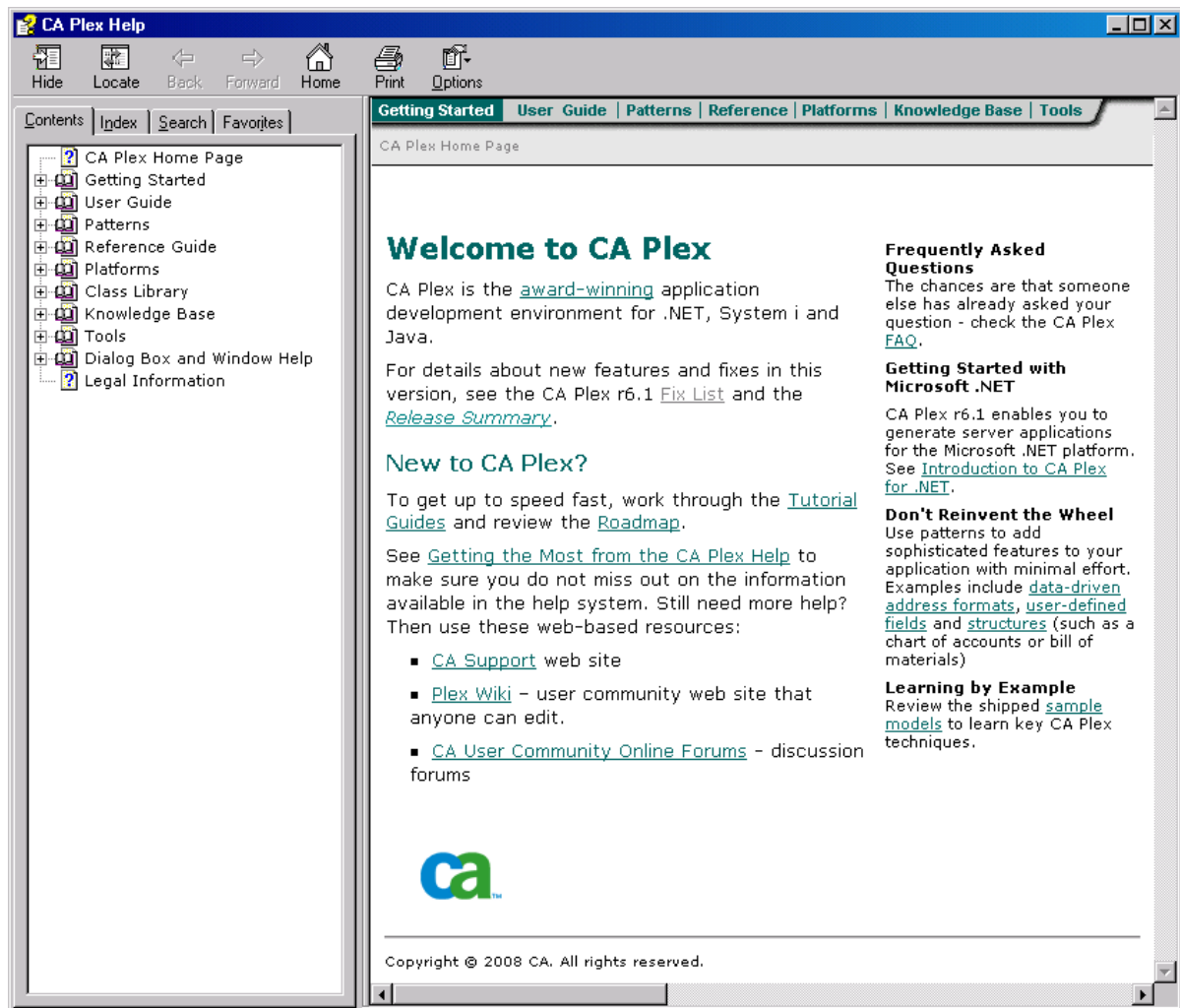
When you run the tutorial application on Windows Vista and Windows Server 2008, you may encounter a problem with updates to records failing. This occurs due to limitations associated with the use of the default Microsoft Access database. Though it does not prevent you from completing the tutorial, you can use an alternative database such as Microsoft SQL Server Express edition to avoid this limitation.

How Long Does It Take?

The first lesson is in the chapter, "Your First Application in 20 Minutes" takes about 20 minutes. Before you reach the end of that chapter, you will have a running application. The rest of the tutorial may take anywhere from two hours to two days depending on how much of the background information and associated online help you want to read as you go along.

Using Online Help

The online help system contains conceptual information, instructions about how to get started, step-by-step procedures for performing CA Plex tasks, and descriptions of the dialogs, menus, and toolbars in CA Plex.



- To locate a particular topic, click the Index tab, and then specify a text string.
- To print an individual topic, choose Print from the Options menu of the currently displayed help topic or the Print toolbar.
- To print a group of topics from the Help Topics window, select the topic, and click the Print button.

Note: You can access online help for CA Plex from any CA Plex Help menu or by pressing F1 while in the CA Plex application.

Conventions in This Guide

This section explains the conventions used to present information in this guide. We recommend that you take the time to familiarize yourself with them.

Boldface

Indicates a variable name or placeholder for which you must supply an actual value—this convention is used in explanatory text, as well as syntax.

Specific CA Plex verbs are always printed in bold and usually with their source and target object types (ENT known by FLD, for example).

Commands and Parameters

Commands and parameters are shown in code format. You enter these examples in CA Plex exactly as shown.

Variables

Italic text shown with a command indicates a user-defined variable. For example, in place of the variable *printer-id.data*, you might enter VPS.JESDS.

PF Keys

Programmable function keys, or PF keys, are represented by the uppercase letters PF followed by one or two digits:

- PF1
- PF12

Note: On most keyboards, PF keys are located either at the top or to the right of the main part of the keyboard. PF keys are usually marked PF or simply F followed by one or two digits, for example, PF1 or F12.

Mouse Buttons

The term mouse button or left mouse button refers to the primary button on your pointing device. Right mouse button refers to the secondary button. You can use the Windows Control Panel to reconfigure the buttons on your mouse as desired.

Chapter 2: Your First Application in 20 Minutes

In this tutorial, you create an application that stores information about projects, the tasks they contain, and the employees assigned to each task.

In this chapter, you define a project, and the information you want to store about each project. The pattern libraries in CA Plex are used to streamline the process. The Panel Designer is also used to make some changes to a dialog. Before the end of this chapter you will have the first part of the application up and running.

This section contains the following topics:

[What Is CA Plex?](#) (see page 13)
[Project Management Application](#) (see page 15)
[Opening the Tutorial Model](#) (see page 16)
[Objects and the Object Browser](#) (see page 16)
[Defining the Project Entity](#) (see page 19)
[Generating and Building the Project Entity](#) (see page 36)
[Using Your Generated Application](#) (see page 39)
[Introducing the Panel Designer](#) (see page 42)
[Chapter Review](#) (see page 53)

What Is CA Plex?

CA Plex is an architected RAD tool that uses patterns to accelerate the design, creation, and maintenance of software applications.

Reusable Business Patterns

A pattern describes a solution to a recurring problem in software systems. Patterns are abstract descriptions that you can reuse in many contexts. An example of a pattern is Structure. The Structure pattern provides the:

- Database schema
- User interface designs
- Programs

You can use to implement hierarchical data structures such as a bill of materials, an organization chart, or a chart of accounts. CA Plex includes libraries of such patterns, and additional libraries are available from third-party vendors.

Several of the patterns from the CA Plex pattern libraries are shown in this tutorial.

Workgroup Development

CA Plex provides a model-based workgroup environment in which teams of software developers can collaborate on the design and construction of applications. At the heart of this environment is a repository whose facilities include:

- Multiple developer support that enables developers to work offline and then update their changes to the central repository
- Built-in configuration management that enables a model to store a single application design in multiple versions, platforms, and national languages
- The integration of designs stored across multiple models

For simplicity, this tutorial focuses on a single-user environment. The single-user model, called a local model, has already been created for you.

Code Generation

Based on the designs held in the repository, CA Plex automatically generates 100% of the code to implement applications and database objects across a variety of platforms. Currently supported implementation options include:

- .NET Server applications generated in C# with Win32, Java or .NET clients.
- Multi-tiered e-business applications with Java or HTML clients and Java, Windows or System i (iSeries) servers
- Multi-tiered client/server applications with Win32 clients and Java, Windows or System i servers
- System i 5250 character based-terminals applications

CA constantly improves and expands the available generators to keep pace with customer demand and advances in technology. Generators insulate you from the underlying technology and its implementation details—CA Plex users can take advantage of new platforms simply by regenerating their existing designs.

This tutorial creates a Windows-ODBC application because this is the simplest platform to set up and configure. The development process is very similar regardless of the platform for which you are developing.

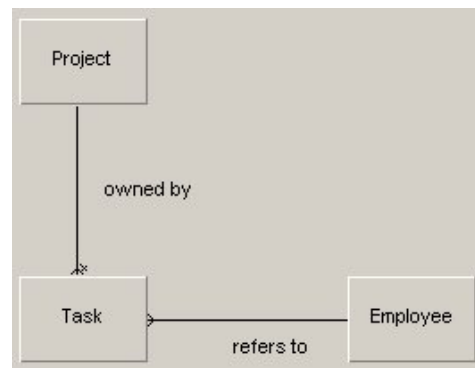
Project Management Application

In this tutorial, you create a simple project management application. For our purposes, a project consists of a group of tasks, and each task has one employee assigned to it. The model contains three entities:

- Project
- Task
- Employee

To save time, the Employee entity is already defined for you.

Here is an entity-relationship diagram of the application created in this tutorial. You will draw a diagram like this later.



If you have worked with entity-relationship diagrams before, you can see that:

- A project can have more than one task, but a task can only belong to one project.
- A task is owned by a project, which means that if you delete a project, you also want all of its tasks deleted.
- Employees are assigned to tasks—one employee can be assigned to more than one task, but each task can only have one employee assigned to it.
- An employee is not dependent on any particular task, which means that if you delete a task, you do not necessarily want to delete the employee record too.

You can see all of this without having to look at any code. This diagram shows useful information, which CA Plex uses to generate the application.

For the purpose of this tutorial, this model is very basic. If you have developed real project management systems before, you may find parts of the model that you would design differently.

Even though it is a simple example, the end product is far from simple. In this tutorial, you create both a wizard and a property sheet to work with the project data. If you have built these in other design environments, you know that you would not be able to reuse much code from the wizard when creating the property sheet. However, CA Plex enables you to create a property sheet using the information you added to create the wizard, plus a few new lines of code.

Opening the Tutorial Model

Start CA Plex and open the supplied tutorial model.

To open the tutorial model, follow these steps:

1. Click Start, Programs, CA, CA Plex r7.0, and CA Plex.
2. From the File menu, select Open.
3. In the Open File dialog, select TutorialWin.mdl, located in ...\\CA\\Plex\\7.0\\Tutorial.
4. Click Open.

By default, when the model opens, a window called the Object Browser appears and the name of the model appears in the title bar of the main application window.

Note: Save Your Work—If you need to stop working on this tutorial before reaching the end of a chapter, save your progress by clicking the Save toolbar button. CA Plex also prompts you to save your changes when you close certain editors after making changes in them.

Objects and the Object Browser

A local model is the file that stores the design of the application you are building. In your model, you create and define objects.

In this chapter, you create an entity called Project. The Project entity uses fields to store information, such as its start and end dates. You inherit functions to enable end users to create, modify, and delete projects. Entities, fields and functions are all types of CA Plex objects.

The Object Browser is the bucket that displays all of the objects in a model. In it, you can see the pattern library objects available, as well as the objects that you define in your model.

CA Plex Object Types

CA Plex uses many types of objects. The following list shows the abbreviation and the icon CA Plex uses to identify each object type that you encounter in this tutorial.

Object Type	Abbreviation	Symbol	Description
Entity	ENT		Entities are the tangible objects, such as customers and orders, and intangible objects, such as projects and job titles, about which you want to record information.
Field	FLD		Fields are the pieces of information that you want to record about an entity, such as employee names, item prices, and order statuses.
Function	FNC		Functions are processes that define the functionality of an application. A function is roughly equivalent to a program or method.
Panel	PNL		Panels are the windows and dialogs that make up the user interface of the application.
Table	TBL		Tables are the physical structures in which data is stored in applications that use relational databases.
View	VW		Views represent all or part of the data in a database table. A view itself does not contain any data—you can think of it as a window through which you look at the data in the table.
Diagram	DGM		Diagrams visually represent a subset of the objects in a model. They show objects and the relationships between the objects.
Message	MSG		Messages hold text. You can use the text in error messages, as the caption for windows and dialogs, and so on.

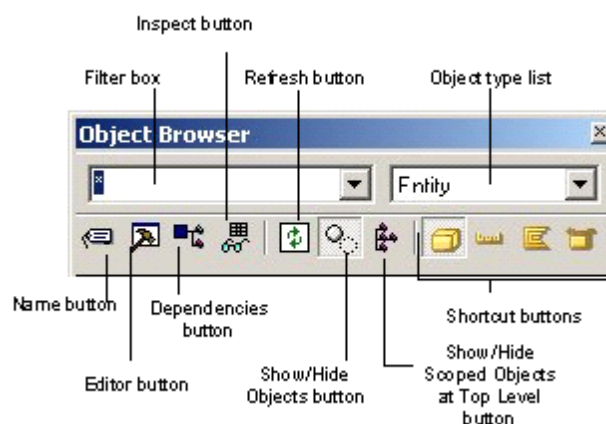
Viewing the Object Browser

To show or hide the Object Browser, do the following:

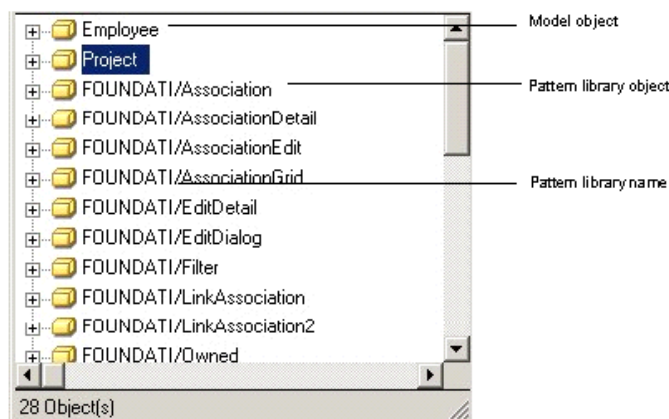
Click the Show/Hide Browsers toolbar button (located on right end of the toolbar) to show or hide the Object Browser.



You can keep the Object Browser open while you work, and use it as a palette from which you can drag the objects you need. When it overlaps other CA Plex windows, it always appears on top.



By default, the Object Browser shows you objects of one type at a time. You can see other objects if they are scoped by the main object type. For more information, see [More About Scope](#). Notice that the Object Browser has shortcut buttons for displaying entities, fields, and functions. Setting the Object Browser to display functions only displays unscoped functions. Functions that are scoped by another object (such as an entity or a view) display when the Object Browser is focused on that object type. You can also select an object type to view from the object list.



You can click the Show/Hide Library Objects toolbar button to show library objects. You can tell that an object is in a pattern library because, by default, it has the library name in front of the object name. For instance, the fourth entity in the preceding graphic is FOUNDATI/Association. This means that the entity Association comes from the FOUNDATION pattern library.



Defining the Project Entity

In CA Plex, you define objects and specify relationships between them with triples. As the name implies, a triple has three parts:

- A source object
- A verb
- A target object

For example, in this tutorial, you use the following triple to define a unique identifier for the Project entity:

Project known by FLD Project ID

In this example, Project (an entity) is the source object of the triple, known by (or known by FLD) is the verb and Project ID (a field) is the target object.

Note: In the CA Plex documentation, verbs are always printed in bold and often with their source and target object types (ENT **known by** FLD, for example).

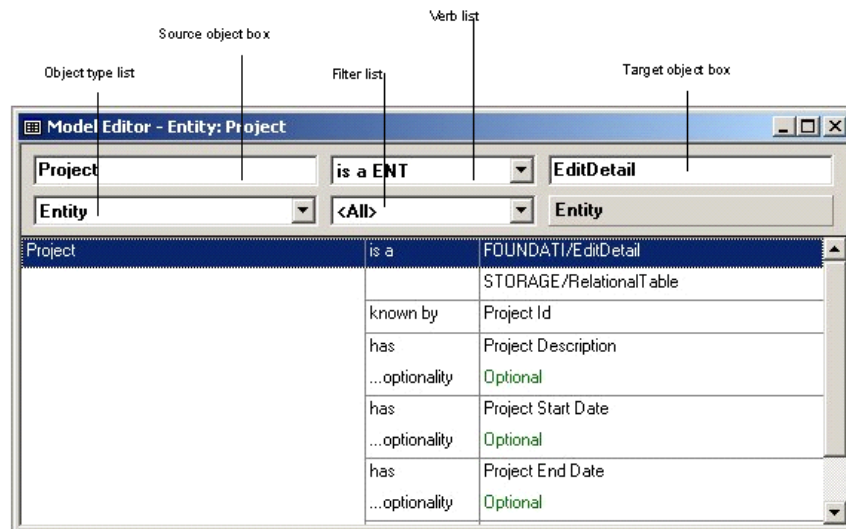
Triples provide a simple, easily understood language for entering information into CA Plex. Here are some examples of triples that you will enter:

- Project **has FLD** Project Description
- Project Description **is a FLD** LongDescription
- Project **described by DGM** Project Diagram

In this chapter, triples are entered to define fields for the Project entity: an identifier, a description, and start and end dates for a project.

You use the Model Editor to view, add, edit, and delete triples.

The following is an example of the Model Editor as it appears after you complete this chapter.



Specifying Attributes for the Project Entity

In the next series of steps you add the following triples to the model:

Project **known by** Project ID
Project **has** Project Description
Project **has** Project Start Date
Project **has** Project End Date

1. Open the Model Editor by choosing Model Editor from the Tools menu, or clicking the New Model Editor toolbar button.

You can see some triples already entered in the Model Editor.

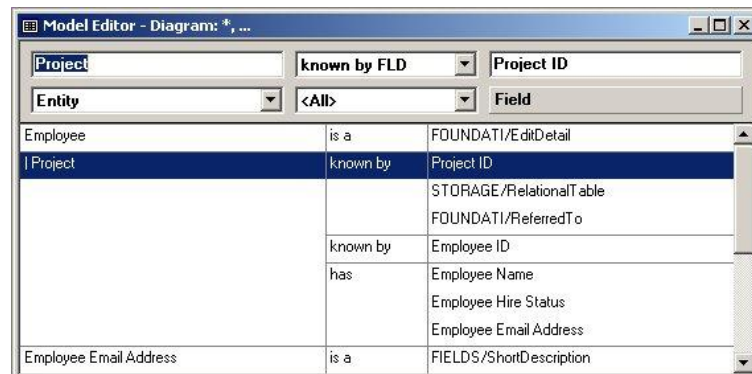


2. From the object type list, select **Entity**.
3. In the source object box, enter **Project**. Even though you have not specified anything about projects yet, you can type its name and CA Plex adds the entity to your model.
4. From the verb list, select **known by FLD**.

Note: Instead of scrolling in the list, you can click the verb list and then press the first letter of the verb you are looking for until the verb appears. To select the **known by FLD** verb, you press the k key once.

5. In the target object box, enter **Project ID**.
6. Press Enter.

The Model Editor should now look like this:



You just created the triple **Project known by Project ID**. This triple defines a primary key for the Project entity.

7. If the Object Browser is not open, click the Show/Hide Browsers toolbar button to open it.



8. Click the Fields toolbar button on the Object Browser to display field objects.



Notice the Project ID field you just added. All objects you define in a local model appear at the top of the Object Browser. Pattern library objects appear below the local objects.

Other fields are already defined: Employee Email Address, Employee Hire Status, and so on. The Employee entity and its fields are already defined in the model you are working with, to save time.



9. Project should still appear in the source object box in the Model Editor. Select **has** FLD from the verb list (you can click the verb list and then press the h key twice).
10. Enter **Project Description** in the target object box.
11. Press Enter.

You have created the triple **Project has Project Description**, which defines the field Project Description for the Project entity. You use this field to store a description of the project. This triple, ENT **has** FLD, creates a non-key attribute. The values in non-key attributes do not need to be unique to each entity. For instance, you can have more than one project with the same text in the description field.

12. Repeat Steps 9-11 to create the following triples:
Project **has** Project Start Date
Project **has** Project End Date

13. Click the Refresh toolbar button on the Object Browser. The Object Browser shows the new fields:



Defining Field Properties

In the next series of steps you use inheritance to define the properties of fields. Inheritance in CA Plex is defined in a simple, yet powerful way—using the **is a** verb. Here are the triples you enter in the next series of steps.

Project ID **is a** FIELDS/Identifier
 Project Description **is a** FIELDS/LongDescription
 Project Start Date **is a** DATE/CheckedDateISO
 Project End Date **is a** DATE/CheckedDateISO

The FIELDS and DATE prefixes indicate that the objects concerned belong to the FIELDS and DATE pattern libraries, respectively.

More About Field Inheritance

The fields you defined in the previous section have two different data types: character and date. These represent different kinds of fields:

- The Project ID field holds some sort of code that uniquely identifies a project
- The description holds text, and the start and end dates hold dates.

Currently, your model only indicates that those fields exist, and that they belong to the Project entity, but has no information about what type of data they store.

Inheritance is the mechanism that enables an object to adopt the properties of another more general (or abstract) object.

By inheriting from pattern library fields, you enable your application to:

- Validate data entered in the fields (ensuring, for example, that an end user does not accidentally enter February 31)
- Display data on the screen appropriately (such as displaying dates according to your Windows settings)
- Store data appropriately in the database (creating a text field in the database for the Project Description field, and date fields for the Project Start Date and Project End Date fields)

To define the properties of Project's fields:

1. Make sure that the Object Browser is focused on fields, and that library objects are showing (click the Show/Hide Library Objects button to display library objects and the Fields button to display field objects).
2. Select the Project ID field in the Object Browser by clicking on the name (not the icon to the left of the name) and drag the field from the Object Browser to the source object box of the Model Editor.

This changes the Model Editor source object type to Field, and changes the verb list so that only verbs appropriate for fields are contained in it.

Note: The cursor changes to a closed parcel icon when you drag an object. It changes to an open parcel icon when it is over a location where you can drop the object:

Closed Parcel Icon	Open Parcel Icon
	

3. From the verb list, select **is a** FLD.
4. From the Object Browser, drag the library object FIELDS/Identifier field to the target object box, and press Enter.

Note: You have to scroll down in the Object Browser to find the FIELDS/Identifier field. You can use the filter box at the top of the Object Browser to only show some of the library items. In this case, you could type ***Identifier*** to display only FIELDS/Identifier. Remember to set the filter back to ***** when you are done.

5. You just created the triple Project ID is a FIELDS/Identifier. Click the Refresh button on the Object Browser. Notice that the Project ID field has a plus sign to the left, indicating that it now has scoped objects.



6. Click the plus sign to expand the field:



Values are another type of CA Plex object. You can see that Project ID now has the value *Blank, but you cannot tell much else about what it inherited from FIELDS/Identifier.

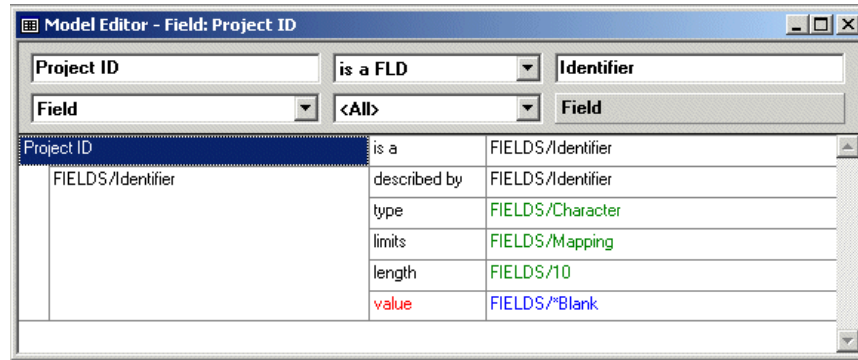
7. Drag the Project ID field from the Object Browser to the body of the Model Editor. The body is the bottom part of the editor, where the full triples are displayed.

When you drag one or more objects to the body of the Model Editor, the display changes to show you only the triples that define those objects. This is called *focusing* the Model Editor. When you drag the Project ID field to the Model Editor, it focuses on this field, showing the triple Project ID is a FIELDS/Identifier. This still does not tell you much.

8. To see more about what an object inherits from its ancestor objects, click the Two Levels toolbar button.



The Model Editor shows another level of detail.



Now you can see that Project ID has inherited a data type of character and length of 10, along with the value *Blank (which you saw in the Object Browser in Step 6).

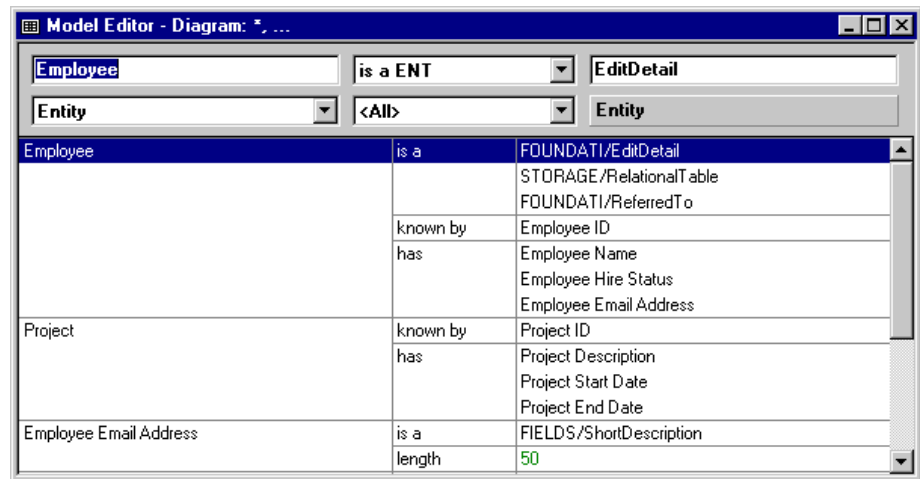
Note: In Step 4, you dragged the library object FIELDS/Identifier from the Object Browser to the target object box in the Model Editor. You can actually enter the name of the object into the target object box (without the library name) to accomplish the same thing. In step 4, you would have entered Identifier.

A word of caution: if you entered a wrong object name, you could create a new object with the wrong name. If you do this, find the erroneous object in the Object Browser and delete it.

9. Click the One Level toolbar button to set the Model Editor to show a single level of information.



10. Reset the Model Editor display by clicking the Clear Focus button. This displays all of the triples in the model, again:



If your Model Editor displays a lot more triples than shown in the previous graphic, you have your model set to display library objects. If this is the case, click the Show/Hide Library Objects toolbar button.

11. Drag the Project Description field from the body of the Model Editor to the source object box. This field is in the third column of the Project **has** Project Description triple.

12. Enter LongDescription in the target object box, and press Enter.

Notice that the Model Editor displays the triple as:

Project Description is a FIELDS/LongDescription

This indicates that you correctly spelled the name of the pattern library field.

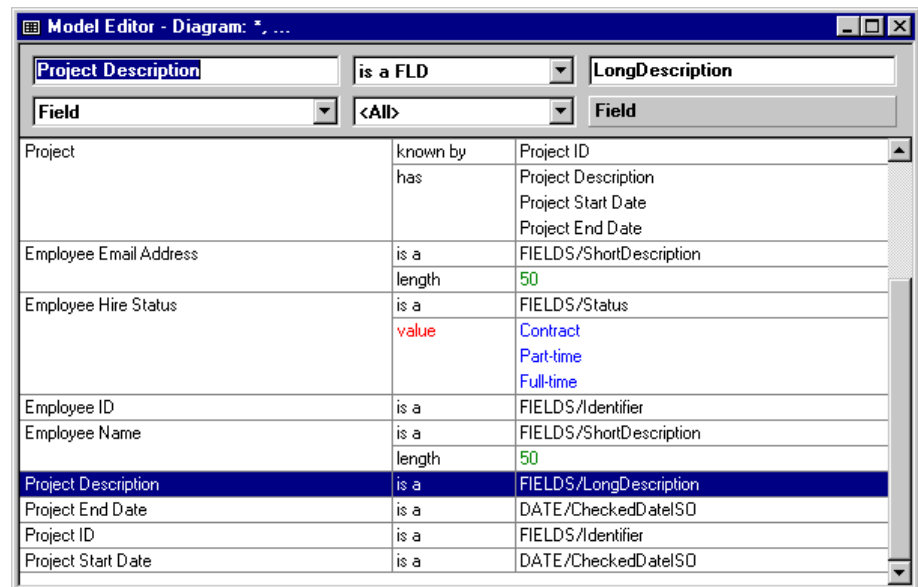
Note: If you create a new object for a model, and it happens to share the name of a library object, you must rename your object or delete it (if you did not intend to create it).

13. Repeat Steps 2-4 or Steps 11-12 to create the following triples:

Project Start Date **is a** DATE/CheckedDateISO

Project End Date **is a** DATE/CheckedDateISO

14. Click the Refresh toolbar button on the main toolbar (not on the Object Browser). Your Model Editor should look like this:



Inheriting from DATE/CheckedDateISO gives the fields functionality to ensure that end users enter valid dates.

15. Use the process explained in Steps 7 and 8 to look at the characteristics that these fields inherit from the Pattern Library fields.

Using Continuation Triples to Make Fields Optional

Hopefully you are getting comfortable with understanding and entering triples. We now introduce an extension to the triple syntax—the continuation triple. A continuation triple is like other triples except that the source object is itself a triple. In the next series of steps, you enter the following continuation triples to specify that certain fields are optional:

Project **has** Project Description
...optionality Optional

Project **has** Project Start Date
...optionality Optional

Project **has** Project End Date
...optionality Optional

You enter other continuation triples later in this tutorial.

More About Optionality

When your end users enter data, they typically enter data in every field. You can specify that some fields are mandatory, while others are optional. If you do not specify optionality, the default is that they are mandatory.

Primary key fields must always be mandatory, so you should leave the Project ID field as it is. But, your end users can use a Project ID that is descriptive enough that they do not need to enter any data in the Project Description field. When end users create projects, they may not have a start or end date yet.

If end users leave a mandatory field blank and then try to close the dialogs, a message dialog prompts them to enter data for the blank mandatory field, and does not let them close the dialog until they do. Since this processing is defined as part of the pattern library and not hard-coded into CA Plex, you can adapt it as required (but this concept is not explained in this tutorial).

To make fields optional:

1. Click the Entities toolbar button to set the Object Browser to display entities.



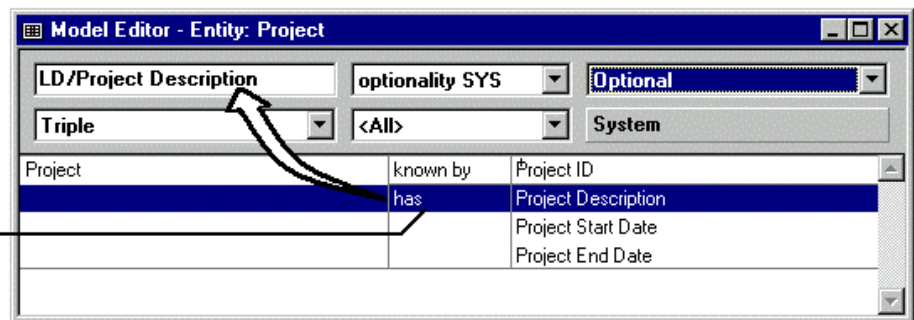
2. Select the Project entity in the Object Browser and click the Inspect toolbar button.



This focuses the Model Editor on the Project entity, showing only the triples that define that entity.

3. Click in the center of the triple **Project** **has** Project Description to select it.
4. Drag it to the source object (top left) box in the Model Editor:

To enter a confirmation triple,
drag from middle column

A screenshot of the 'Model Editor - Entity: Project' window. The window has a title bar and several controls. At the top, there's a text field containing 'LD/Project Description', a dropdown menu for 'optionality' set to 'SYS', and another dropdown menu set to 'Optional'. Below these is a 'Triple' dropdown set to '<All>' and a 'System' dropdown set to 'System'. The main area is a table with three columns. The first column contains 'Project'. The second column contains 'known by' and 'has'. The third column contains 'Project ID', 'Project Description', 'Project Start Date', and 'Project End Date'. The row with 'Project' and 'has' is highlighted. An arrow points from the text 'To enter a confirmation triple, drag from middle column' to the 'has' text in the second column. Another arrow points from the 'LD/Project Description' text field to the 'Project Description' text in the third column.

5. Select **optionality** SYS from the verb (top middle) list, and Optional from the target object (top right) list.

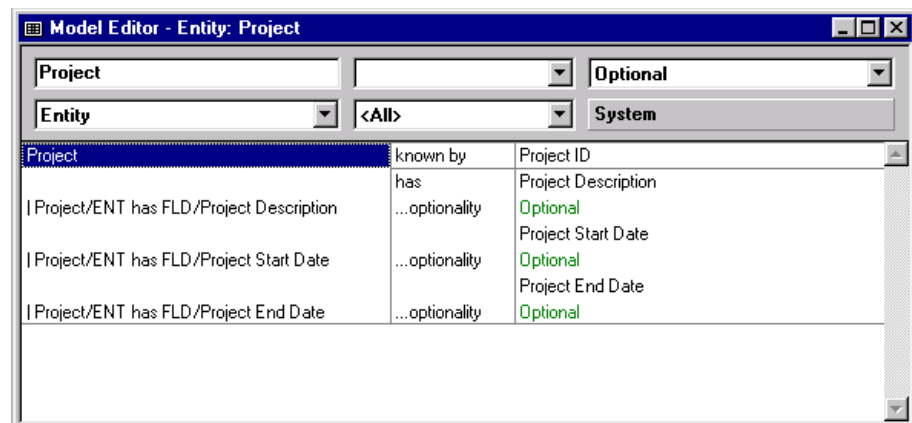
6. Press Enter.

You have entered the continuation triple:

Project **has** Project Description
...optionality triples Optional

Note: If you do not see this view, click the One Level toolbar button to set the Model Editor to show a single level of information.

7. In the same manner, make the Project Start Date and Project End Date fields optional.
8. Refresh the Model Editor:



Using Inheritance to Define the Project Entity

You have now defined the fields in which the Project entity stores data, and specified the Pattern Library fields from which those fields inherit. In the next step, you give the Project entity a user interface and functionality to interact with a database.

You again use inheritance to add this functionality. In fact, you do it with only two triples:

Project **is a** FOUNDATI/EditDetail
 Project **is a** STORAGE/RelationalTable

The first inheritance triple gives your entity the ability to display and process a user interface. The second inheritance triple provides the functionality to read Project records from, and write Project records to, a relational database.

The step-by-step instructions that follow help you better understand the consequences of entering these two triples.

CA Plex Feature: Patterns and Inheritance

Inheritance is the primary means of reuse when you design an application with CA Plex. Inheriting from pre-built and pre-tested pattern libraries provides tremendous productivity and quality benefits.

Patterns are groups of CA Plex objects that are designed to be reused and customized. In this tutorial, you inherit from the included pattern libraries. Additional pattern libraries are available from third-party vendors.

Patterns Versus Templates

The templates or frameworks supplied by other tools provide some of the benefits of patterns. Typically, a template provides a means of copying a predefined solution.

Note: Inheritance is not the same as copying.

When an object inherits from an CA Plex pattern, the relationship is dynamic—changing a pattern automatically changes all the objects that inherit from the pattern.

Also note that patterns are created with CA Plex rather than being hard-coded into the tool. You can customize a Pattern Library by creating your own patterns that inherit from the supplied patterns. Or you can create your own patterns from scratch.

Patterns and Components

Patterns and components (such as COM and EJB components) are complementary technologies. CA Plex patterns can be used to combine and implement groups of components. Components are usually likened to the building blocks of a house. Extending this metaphor, you can think of patterns as the blueprint or plan for the house.

Like patterns, components are intended to maximize software reuse. Components are black-boxes, whereas patterns can be regarded as gray-boxes whose internal structure can be modified in certain places. Another distinction is that patterns are purely a design construct whereas components form a part of the running application.

Adding Functionality to the Project Entity

To add functionality to the Project entity:

1. If the Object Browser is not displaying entities, click the Entities toolbar button.



Notice that there is no plus sign to the left of the Project entity. This tells you that there are no objects scoped to it.

To add these triples, you must first set the source object type in the Model Editor to Entity. You could change the object type directly. But, when you drag an object from the Object Browser, it sets the object type and shows all of the triples for that object.

2. Drag Project from the Object Browser to the body of the Model Editor.

This is similar to using the Inspect toolbar button. It changes the Model Editor so that it only shows triples related to the Project entity, changes the object type (assuming it was not already set to Entity), and puts Project in the source object box. The body of the Model Editor is the area of the Model Editor below the Entry Bar that displays existing triples.

3. From the verb list, select is a ENT.

Note: The is a verb you use in this step is different from the one that you used to specify inheritance for Project's fields. There are several verbs that have the same name, but which have a different source and target object. CA Plex only lets you select the verb that matches the target object (in this step, the is a ENT verb). For more information about the types of is a verbs, search for is in the online help index.

4. Enter EditDetail in the target object box, and press Enter.



You just created the triple Project is a FOUNDATI/EditDetail. This indicates that Project inherits the structure and functionality of the EditDetail pattern in the FOUNDATION Pattern Library. You can find the FOUNDATION/EditDetail pattern in the Object Browser by making sure that library objects are displayed (by clicking the Show/Hide Library Objects toolbar button) and scrolling down.

For more information about the EditDetail pattern, select it in the Object Browser and press SHIFT+F1.

5. In the Object Browser, click the Refresh toolbar button. By inheriting from EditDetail, the Project entity now has some scoped objects (it has a plus sign next to it).



For more information about scoped objects, see [More About Scope](#).

6. Click the plus sign to the left of the entity icon to expand Project, and then click the plus sign next to the Edit function to expand it:



Project inherited one function, Edit, with a scoped panel and a caption, and two views, Fetch and Update. These objects give Project a user interface, and enable it to store data to and retrieve data from a database.

Specifically:

- The Edit function displays the panel scoped by it.
- The Caption and Panel objects, which are scoped by the Edit function, store the layout of the Edit Projects panel. To see what this panel looks like, see the generated application illustration in [Running the Project.Edit Function](#).
- The Fetch and Update views scope functions that read and write database records.

Next, you indicate how the Project entity stores information. Your application uses a relational database, so you need processing that creates and maintains database tables. You set Project to inherit from STORAGE/RelationalTable for that functionality.

7. The Model Editor should still have Project in the source object box, and is a ENT showing in the verb list. Drag STORAGE/RelationalTable from the Object Browser to the target object box and press Enter to create the following triple:

Project is a STORAGE/RelationalTable

8. Click the Refresh button to see all of the triples that you defined for the Project entity in the Model Editor.
9. Click the Refresh button on the Object Browser:



Notice that the Project entity inherited an object called Physical Table from STORAGE/RelationalTable. This defines the table that is created in the database when you generate the application. The fields you defined for the Project entity are stored in this table.

More About Scope

A *scoped object* is an object that belongs to another object. A scoped object cannot exist independently of the object by which it is scoped. For instance, a panel that is scoped by a function is deleted when that function is deleted.

Some types of objects are unscoped. This means that they exist independently of all other objects in the model. For example, entities are typically unscoped. Conversely, a table is always scoped to an entity. To create a table object, you must specify the entity to which it belongs.

When you refer to a scoped object, you often need to use its full name to avoid ambiguities. For example, there is an entity called Project that scopes a function called Edit. The function's full name is Project.Edit, which distinguishes it from other Edit functions such as Employee.Edit that also exist in this model.

The previous steps showed you how scoped objects are usually created when you inherit from a pattern. You can also create them manually.

Generating and Building the Project Entity

You have now defined fields for the Project entity, and specified the properties of those fields. You have also defined functionality for the Project entity, providing a basic user interface and the ability to write to and read from a database. You are now ready to generate and build the Project entity. This is the process in which CA Plex turns your model into source code (generating), and then turns your source code into compiled objects (building).

After you have generated and built the objects in your model, you will be able to run the program to see what you have created.

To generate and build the Project entity:

1. From the Tools menu, choose Generate and Build, or click the New Gen and Build toolbar button.

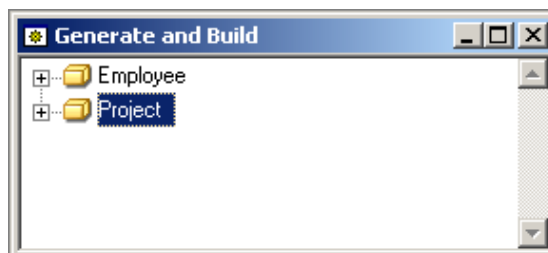


The Generate and Build window appears. The Message Log pops up when you open the window. From the Options menu, choose Quiet Mode, and then minimize the window. This keeps the Message Log from popping up every time it has a new message.

2. If the Generate and Build window shows library objects, click the Show/Hide Library Objects toolbar button to hide them.



The Generate and Build window now shows the Project and Employee entities (Employee was already added to your model before you started):



3. Select the Project entity.
4. Click the Generate and Build toolbar button.



CA Plex expands Project and highlights all of the generatable objects under it. Not all of the scoped objects are selected.

5. A Confirm Generate dialog appears, indicating the number of objects that are generated. Click Yes.

CA Plex generates those objects, and then summarizes the generation process.

Note: Three warnings appear in the Message Log during the generation and build processes. This is expected and not a problem. You can expect *one* warning message (BLD9083) to occur during the generation stage. For more information, see Checking for Errors.

6. When the generation process is complete, the Cancel button becomes an OK button. Click OK to close the Generation Status dialog.
7. CA Plex prompts you to compile and build the objects. Click Yes.

CA Plex starts Microsoft Visual Studio to compile the generated C++ source files. Depending on your settings, Visual Studio may start minimized.

At the same time, the database table is sent to the ODBC data source being used for this tutorial. To make set up easier, CA Plex created this data source automatically during installation together with the underlying Microsoft Jet database.

Note: You can expect to see two of the three warning messages during the build process. For more information, see Checking for Errors.

You can tell that your C++ build is done when the label on the Visual Studio taskbar button changes from the name of the model to Microsoft Visual C++. Or, if you have opened the Visual Studio window so that you could watch the build, you can tell that it is finished when the cursor returns to the top of the build summary.

CA Plex Feature: Native Platform Implementation

CA Plex provides a complete environment for generating and compiling across all supported platforms. For Windows implementations, you can see how it seamlessly integrates with Microsoft Visual Studio.

CA Plex generates native C++ that is based on Microsoft Foundation Classes (MFC), a robust, industry standard C++ class library. Each function in the model is implemented as a Windows DLL.

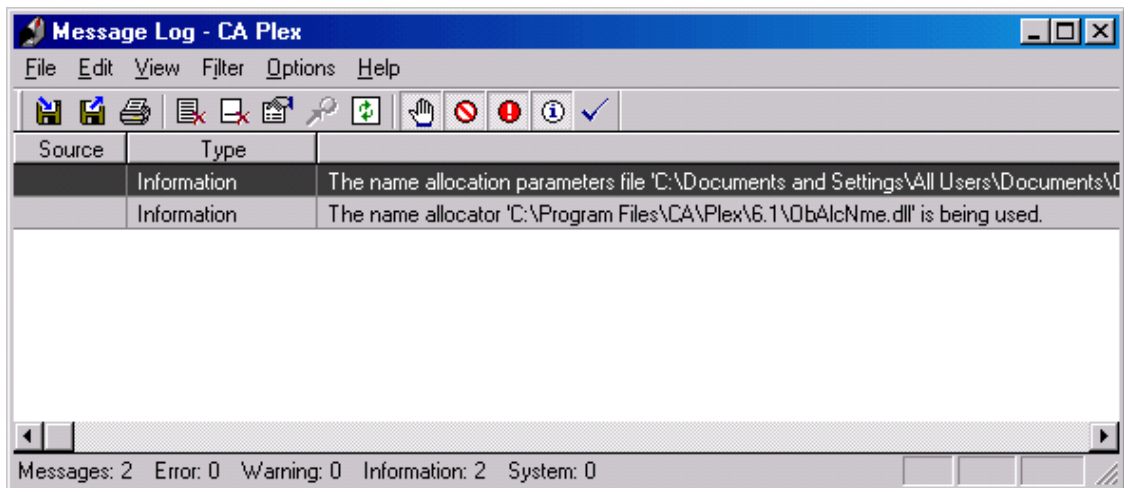
If we chose to implement our application in Java, CA Plex would generate 100% pure Java code based on Sun's Java 2 platform. Similarly, we could also implement the application on the IBM System i by generating native RPG and DDS code.

Checking for Errors

Three warnings appear in the Message Log during the generation and build processes. This is expected and not a problem. If you have additional warnings or error messages this may indicate a problem. The most likely cause is that you have not entered all the necessary triples into the model—go back and check against the instructions.

Errors are also likely if CA Plex did not install properly.

- One warning message (BLD9083) indicates that the Project Description field has a length greater than 255. This is not a problem for the database we are using.
- Two warning messages (BLD9081) indicate that no source for various views was submitted to the ODBC data source. This is expected because we set up CA Plex not to implement views in the Jet database being used for this tutorial.



Checking for C++ Build Errors

1. Click the WinC Build Summary toolbar button to view the build summary, to make sure there were not any errors.



Note: The WinC Build Summary toolbar button may not work if the path to your generation directory contains embedded spaces.

You are not prompted to review the build summary any more in this tutorial, but if your application does not run correctly, you should review to see if there were errors.

2. Click the Save toolbar button to save your model.



Using Your Generated Application

By entering a few triples, you created a fully functional application—an instance of the EditDetail pattern. You can use your generated application to create, edit, and delete projects. You can create a description for each project, and indicate a start and end date.

Running the Project.Edit Function

1. In the Generate and Build window, select the Edit function under the Project entity:
2. Click the Run toolbar button.



3. When you are prompted to choose the data source for your application, double-click Plex r7.0 Tutorial.dsn.

Your generated application starts. It should look like this:

 A screenshot of a software application window titled "Edit". The window has a standard Windows-style title bar with a close button. Inside, there's a toolbar with a green plus icon and a yellow icon. Below the toolbar is a large empty rectangular area on the left, labeled "Project ID" at the top. On the right side of the window, there are four text input fields: "Project ID", "Project Description", "Project Start Date" (with a default value of "00/00/0000"), and "Project End Date" (also with "00/00/0000"). At the bottom right, there's a checkbox labeled "Continue new?" which is unchecked. Below the checkbox are four buttons: "Apply", "New", "Delete", and "Refresh".

4. Leave this dialog open. The next steps show you how to add projects to your database.

CA Plex Feature: Default Panel Layouts

Based on the design details you entered earlier, CA Plex automatically added the correct fields to the panel and provided a default layout. You will adjust the default layout later. The general appearance and functionality of this panel, with the grid on the left and the fields on the right, is determined by the Pattern Library from which it is inherited. You can override almost any aspect of this appearance and behavior.

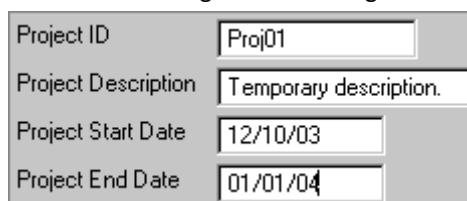
Adding a New Project

To add a new project:

1. You started your application in the previous procedure. Now, enter the following values. Note that the format required for the dates vary depending on your regional settings in the Windows Control Panel.

Value	Enter
Project ID	Proj01
Project Description	Temporary description.
Project Start Date	12/10/2003
Project End Date	01/01/2004

The area on the right of the dialog should look like this after you enter these values:

A screenshot of a dialog box with a light gray background. It contains four labeled text input fields arranged vertically. The labels are 'Project ID', 'Project Description', 'Project Start Date', and 'Project End Date'. The corresponding values entered in the fields are 'Proj01', 'Temporary description.', '12/10/03', and '01/01/04'. The text 'Temporary description.' is truncated on the right side of the field.

Project ID	Proj01
Project Description	Temporary description.
Project Start Date	12/10/03
Project End Date	01/01/04

For now, you only enter **Temporary description.** in the Project Description field because the field extends past the right edge of the dialog, and if you type anything longer, you will not be able to see what you are typing. Since Project Description was defined as an optional attribute, you can leave it blank if you prefer.

After you finish entering data, you modify the panel layout so that you can see more of the description so that it does not run off the right edge.

2. Click the Apply button. The values are posted to the database, and the grid region on the left of the window shows the new project.
3. Click the Copy toolbar button to create a new project populated with Proj01's data. Add the following two projects to the database:



Value	Enter
Project ID	Proj02
Project Description	Temporary description.
Project Start Date	03/01/2001

Value	Enter
<i>Project End Date</i>	04/04/2001

Value	Enter
Project ID	Proj03
Project Description	Temporary description.
Project Start Date	05/05/2004
Project End Date	06/05/2004

- Click the close window button at the upper right corner of the dialog to exit the application.

Note: None of the panels inherited from the Pattern Libraries have Cancel buttons on them. If you click the close window button, any pending actions are discarded. If you click Apply, those changes are completed before the window closes.

CA Plex Feature: 100% Code Generation

You have entered 13 triples into the model. From those triples, CA Plex generated over 9,000 lines of C++ and SQL code to create a fully-functional application. One hundred percent code generation!

Feel free to experiment. Check out referential integrity checking inherited from the pattern library. For example, you cannot enter the same Project ID for multiple records.

Preserving Data

By default, each time you build your application, CA Plex rebuilds all of the objects you select in the Generate and Build window, including the tables in your database. Because rebuilding a database table erases all data in the table, if you leave your local model set up as it is, you will lose all of the data you just entered the next time you build.

You can prevent losing this data by entering a TBL **implement** SYS No triple for the table. This keeps the table from being rebuilt the next time you build the entity to which it is scoped (in this case, Project). Only set this triple **after** you have built the table at least once.

To keep the Project entity table from being regenerated:

1. In the Object Browser, select Project.Physical Table, and drag it to the source object box in the Model Editor.
2. From the verb list, select **implement** SYS.
3. From the target object list, select the value No.
4. Press Enter.
5. Save your model.

Keep in mind that if you make any changes to an entity that affect its table, such as adding fields to it, you must set the **implement** SYS value back to Yes, and regenerate the table. Any data in it is lost. If you want to preserve data entered after rebuilding, make sure you reset the **implement** SYS value to No.

Introducing the Panel Designer

You may have noticed when you ran the Project.Edit function that the dialog did not look quite right. The Project Description field ran off the right edge and there was not enough room to enter more than a couple of words of description.

The Panel Designer changes the panel displayed by Project.Edit. You change the size of the Project Description field and give it a multi-line edit control (so that you can enter multiple lines of text). Some spin controls are added to the date fields as well.

CA Plex Feature: Visual Development

You have seen that, unlike visual development tools, the starting point for an CA Plex application design is a design model, not screen layouts. This is because design and modeling are the keys to building large-scale applications successfully. But, just like a visual programming tool, CA Plex provides an easy-to-use editor for designing graphical user interfaces (GUIs). It includes a rich set of native GUI controls plus the ability to use third-party components (ActiveX controls in Windows and JavaBeans in Java). There is another mode of editor that enables the character-based screens of an System i 5250 application to be designed in the same way.

Opening the Panel Designer

To open the Panel Designer:

1. If the Object Browser is not displaying entities, click the Entities toolbar button.

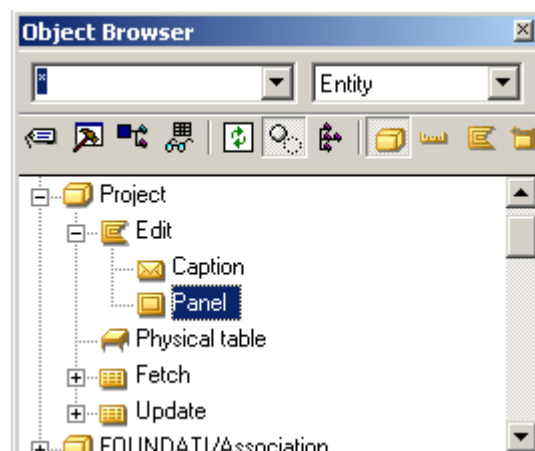


Remember that this button is not on the toolbar, but rather is on the Object Browser. Some buttons are on both the Object Browser and the toolbar.

2. If you do not see the Project entity, click the Refresh toolbar button.



3. Expand the Project entity, expand the Edit function, and select Panel:



4. Click the Editor toolbar button.



The Panel Designer appears. You can click the Editor button when you select any object. CA Plex always opens the appropriate editor.

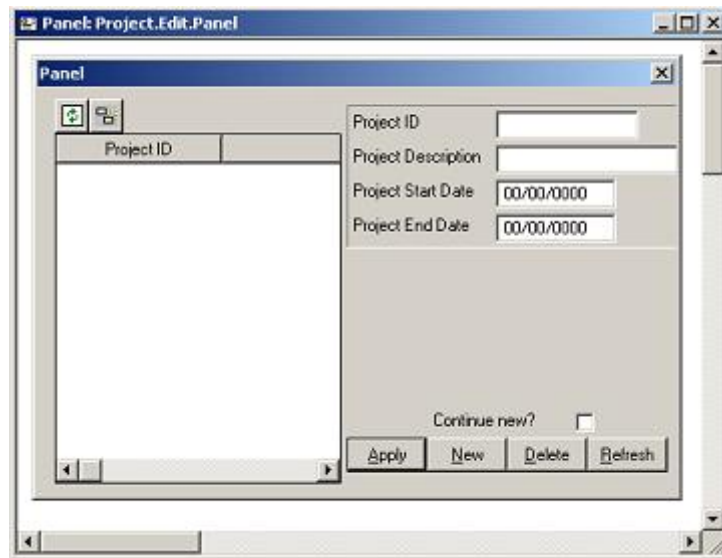
The Panel Designer is made up of three windows:

- Design Window
- Panel Palette
- Panel Property Sheet

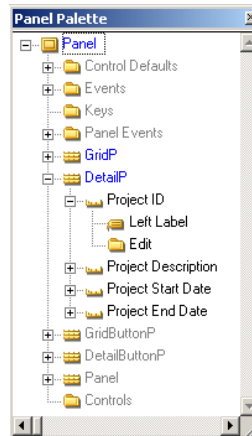
Design Window

The Design Window is the main window in the Panel Designer, and shows you what your panel looks like. When you change a panel's properties, you see how the changes affect what the panel looks like here. You can select, move, and resize buttons, fields, and other user interface elements using this window. When you make visual changes to the panel and its elements using the other windows, the changes appear here.

A panel's elements are grouped into regions (each region has a name). The following graphic shows the grid and detail regions in the panel displayed by Project.Edit:



Panel Palette

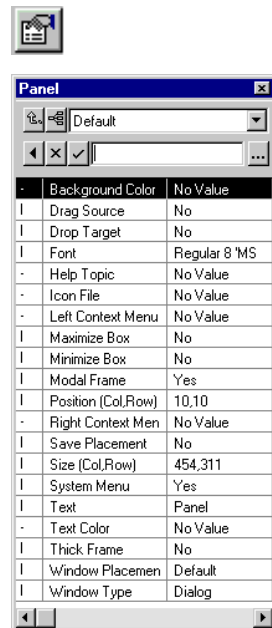


The Panel Palette (left) shows all the elements of the panel, including fields, labels, and buttons, which are grouped under folders (📁) and regions (📊). Feel free to expand the folders and regions to see what they contain. Most of the visible elements are contained in regions. Notice that you can see the five regions of the panel (GridP, DetailP, GridButtonP, DetailButtonP, and Panel) in the Panel Palette (left).

When a region is expanded, as the DetailP region is in this example, you can see the elements contained in that region. In the case of the DetailP region, this includes the Project ID, Project Description, Project Start Date, and Project End Date fields.

When represented on panels, fields typically contain more than one part, usually including at least one label and a control. Notice on the preceding graphic that the Project ID field shows both a Left Label and an Edit control. The label indicates whether it is a Left Label, a Top Label, or a Right Label. The type of label indicates if it appears to the left or right of the control (left/right label), or as a column heading (top label). The control is the part that end users interact with. The settings for the control indicate if it is displayed as an edit box, a list box, a combo box, and so on.

Property Sheet



Every element on a panel has a set of properties associated with it, such as its size, color, or position. The Property Sheet shows you the properties of the currently selected element. If no element is selected, the Property Sheet shows you the properties of the panel as a whole (left).

To see what you can change about an element, select the element in the Design Window or the Panel Palette, and check its properties on the Property Sheet. The properties displayed depend on the type of element selected.

For example, you can specify if a button should be included in the tab sequence by setting its Tab Stop property. However, a field label would not ever be included in a tab sequence, so if you select a label, the Property Sheet does not display this property.

Defining a Multi-line Edit Control

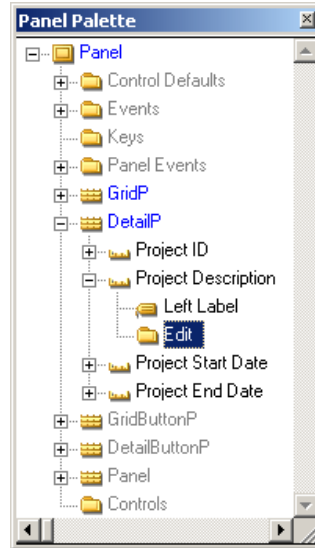
In this set of steps, you modify the panel displayed by Project.Edit so that it displays more than one line of text, and so that it does not run off the edge of the dialog.

To modify Project.Edit.Panel:

1. Arrange the Panel Designer windows so that they do not overlap, and you can work with the contents of each.

You must select an element on a panel to make a change to it. To select only the element, and not the region it is in, you use the Panel Palette.

2. If it is not already expanded, expand Panel to display the regions on the Edit panel.
3. Expand the DetailP region, and then expand the Project Description field.



4. Select the Edit control.

The values in the Property Sheet indicate the properties you can change for the edit box. Also, notice in the Design Window that only the edit box next to the label Project Description is selected.

5. In the Design Window, drag the edit control below the label, so that its left edge lines up more or less with the j in the word Project. Do not be concerned that it covers up the Project Start Date field—this is fixed later.

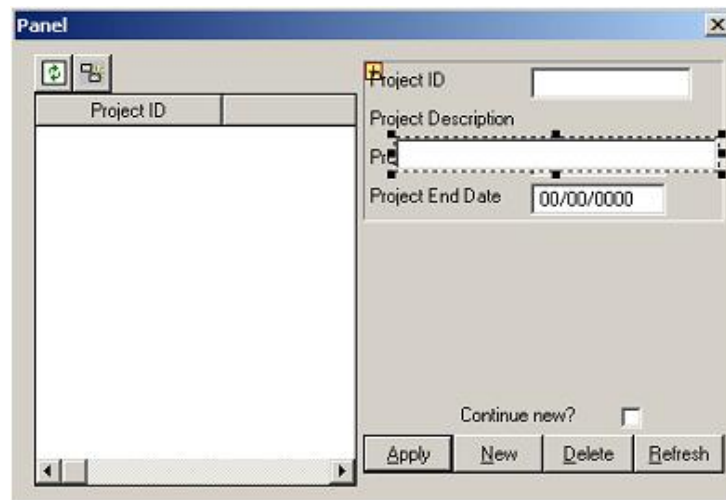
In the Property Sheet, the Position (Col, Row) property changes as you move the element. If you want to make fine tuning changes, you can specify the exact pixel location in the Property Sheet.

The field still extends past the right edge of the panel.

6. In the Property Sheet, click the Size (Col, Row) property.
7. In the edit area at the top of the Property Sheet, change the value to 200, 20, and then click the check mark button.

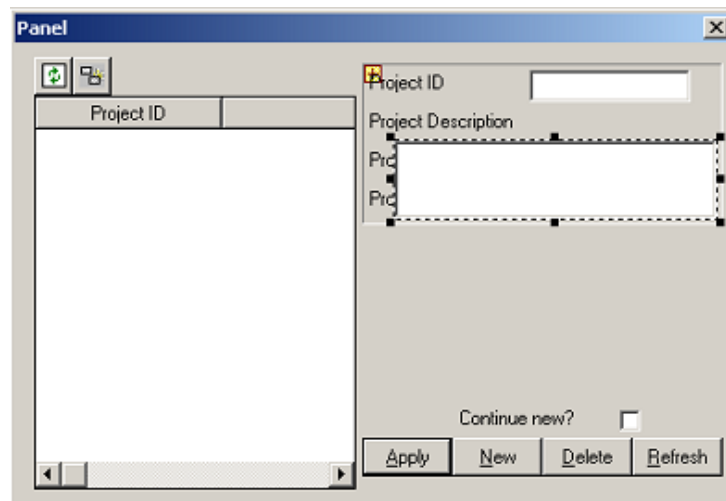


The edit control resizes so that it fits within the panel, and the beveled box that surrounds the fields resizes so that it is just slightly wider than the fields.



8. Now, drag the handle on the bottom center of the edit box to make the box taller. Make it about three times as tall.

The panel should look something like this:



You have repositioned and resized the Project Description edit box. Next, you set two properties that enable end users to enter more than one line of text in the edit box.

9. In the Property Sheet, double-click the Multiline property to change its value from No to Yes.

10. Select the Scrolling property in the Property Sheet.

Note: If the Scrolling property is not available, make sure that the value of the Multiline property changed from No to Yes.

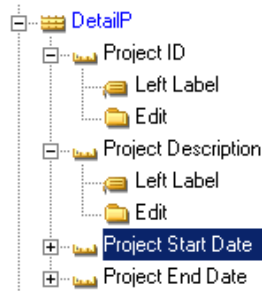
Notice that the input area at the top of the Property Sheet changed so that you can select a value from a list, but you cannot type in a value.

11. Select Vertical from the list.

Setting the Scrolling property to Vertical causes the text in the multi-line edit box to wrap to the next line when the cursor reaches the right edge. If you did not set this option, text would continue on the first line off the edge of the field, rather than wrapping to the next.

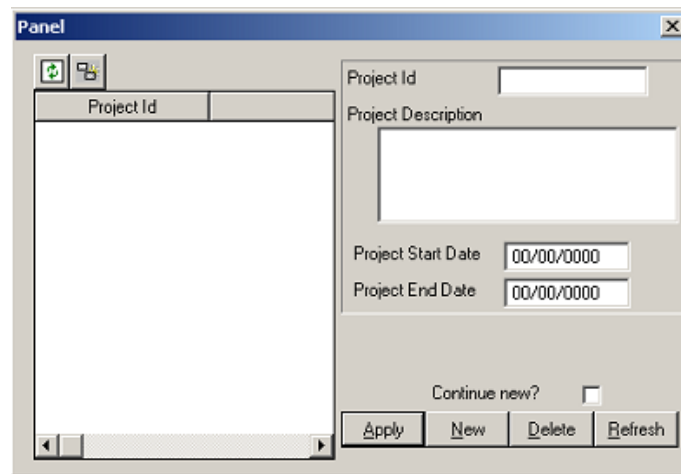
When you changed the size and shape of the Project Description edit box, it covered both the Project Start Date and the Project End date fields. Next, you move those two fields down.

12. In the Panel Palette, select the Project Start Date field in the DetailP region (without expanding it).



13. Holding the CTRL key down, select the Project End Date field. This selects both of the fields (you can see this in the Design Window).
14. In the Design Window, drag the fields so that they are below the bottom of the Project Description multi-line edit box.

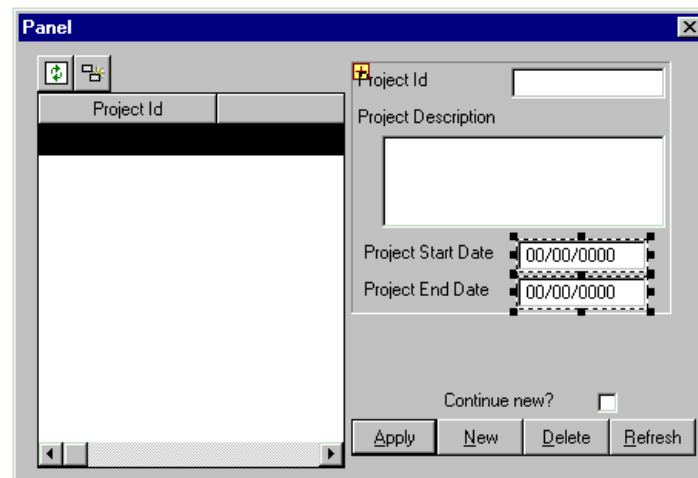
In the Design Window, you can see that the beveled box around the fields resizes to fit around all of the fields:



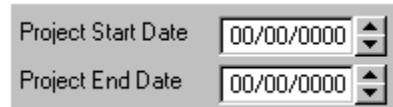
Adding Spin Controls

In the following steps, you modify the two date fields so that they include spin controls.

1. On the Design window, hold down CTRL and click one of the edit controls of the date fields. Holding down CTRL causes the control to be selected instead of the region in which it is contained.
2. Hold down CTRL and SHIFT at the same time and click the other date control. Both date controls should now be selected and the Design window should look like this:



3. In the Property Sheet, locate the Spin Control property and set it to Yes.



Notice how you can select multiple elements on a panel and change properties for all of them.

4. From the File menu, choose Save.
5. Close the Design Window. This closes the whole Panel Designer.

Regenerating the Function

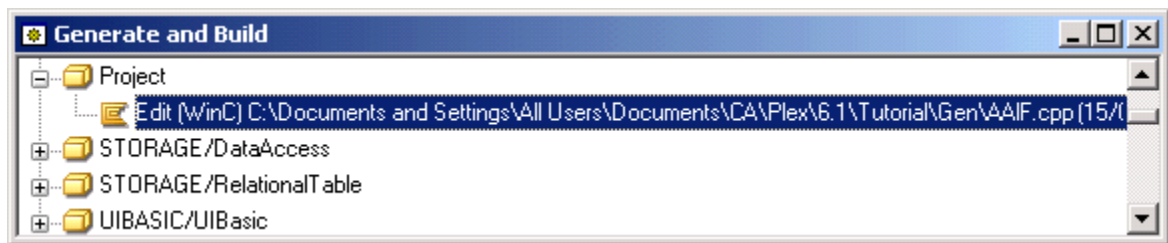
Now that you have modified the panel, you must regenerate and rebuild the function that displays it before you can see the changes.

To regenerate and rebuild Project.Edit:

1. If you still have the Generate and Build window open, switch to it. Otherwise, click the New Gen & Build window toolbar button.



2. Select Project.Edit:



To rebuild a panel, you regenerate and rebuild the function that scopes it. Because you only modified the Project.Edit.Panel object, you only need to regenerate and rebuild Project.Edit. If you had made changes that affected the scoped table or other objects, you would need to regenerate and rebuild those other objects as well.

3. Click the Generate and Build toolbar button.



4. Click Yes when prompted to generate, and again when prompted to build.
5. When the generate and build process is complete, choose Project.Edit in the Generate and Build window and click the Run toolbar button.



6. When prompted to choose a data source, choose Plex r7.0 Tutorial.dsn.

The modified dialog appears, showing the data you entered:

The screenshot shows a Windows-style dialog box titled "Edit". It contains a table with two columns: "Project ID" and "Temporary Description". The table has four rows: Proj01, Proj02, Proj03, and Proj04. To the right of the table are detail fields for the selected row (Proj01): "Project ID" (text box with "Proj01"), "Project Description" (text area with "Temporary Description"), "Project Start Date" (calendar picker showing "12/10/03"), and "Project End Date" (calendar picker showing "01/01/04"). At the bottom right, there is a checkbox labeled "Continue new?" and four buttons: "Apply", "New", "Delete", and "Refresh".

Project ID	Temporary Description
Proj01	Temporary Description
Proj02	Temporary Description
Proj03	Temporary Description
Proj04	Temporary Description

Project ID: Proj01
Project Description: Temporary Description
Project Start Date: 12/10/03
Project End Date: 01/01/04
Continue new? ☐
Apply New Delete Refresh

7. Select the text in the Project Description field for Proj01, and replace it with a longer description. For example:
Create a database application for the McCreedy account. Contact is Jim Hauser, 415-555-3146.
8. Click Apply.
Notice that the description has changed in both the grid and detail regions.
9. Select Proj02 and repeat this process to change its description to:
Create an email client for internal support staff.
and change the Proj03 description to:
Convert the ODBC version of the McCreedy application to a System i version.
10. Close the application.

Chapter Review

In this chapter, you:

- Learned how to use the Model Editor to add triples
- Inherited from pattern library objects to define fields, and data access and user interface functions
- Generated and built the Project entity's database table and the inherited functions, and added some sample data
- Used the Panel Designer to modify the inherited panel layout
- Regenerated and rebuilt the affected objects to see your changes

This chapter introduced the following patterns (and their scoped objects):

Pattern	Description
 FIELDS/Identifier	A 10-character data field.
 FIELDS/LongDescription	A variable-length character field.
 DATE/CheckedDateISO	A date field, where the date is stored in the ISO standard for dates. Verifies that dates entered are valid.
 FOUNDATION/EditDetail	Lists database records and lets you edit, add, change, and delete records in a single dialog.

This chapter introduced the following triples:

Triple	Description
ENT known by FLD	Defines the primary key of an entity
ENT has FLD ... optionality SYS	Defines a data attribute of an entity and whether it is mandatory or optional to provide a value for a field
ENT is a ENT	The source entity inherits the properties of the target entity.
FLD is a FLD	The source field inherits the properties of the target field
TBL implement SYS	Specifies whether to generate and build a database table

Chapter 3: Modeling Entity Relationships

In this chapter, you use the Diagrammer to create an entity-relations (E-R) diagram of your data model, and specify the relationships between the entities.

You also define the Task entity and generate and build the Employee entity.

This section contains the following topics:

[Introducing the Diagrammer](#) (see page 55)

[Creating a Diagram](#) (see page 56)

[Defining the Task Entity](#) (see page 61)

[Generating and Building the Employee Entity](#) (see page 63)

[Adding Employee Records](#) (see page 64)

[Chapter Review](#) (see page 65)

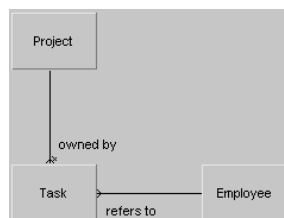
Introducing the Diagrammer

Typically, the first step in creating a business application is creating a data model. In the data model, you specify how real-world entities are represented in a database.

Diagrams enable you to visually communicate the specifics of an application's data model to other developers or business analysts, and in some cases, to your end users.

With the project management application you are designing, the real-world entities map quite easily into database objects. The application tracks projects, the tasks that make up those projects, and the employees that are assigned to those tasks. Projects, tasks, and entities are each represented by one entity.

When you have identified the data structures you need, you can represent this information visually using design diagrams. The CA Plex Diagrammer enables you to create these diagrams, and define objects and their relationships. When you create entities and relationships using the Diagrammer, the corresponding triples are automatically added to your model.



In this chapter, you use the Diagrammer to create a diagram of the data model behind the sample application (left). In the process, CA Plex creates the Task entity for you, along with triples representing relationships between the entities.

In CA Plex diagrams, objects are represented by boxes (or other shapes) called nodes. A relationship between two objects is represented in one of two ways:

- As a line (called a link) which joins the two nodes
- As a node within a node (called a sub-node)

In the diagram that you create, the nodes represent entities.

Together, two nodes and the link between them, or a node and its sub-node, represent a triple in an CA Plex model. For example, the diagram of the project application has an owned by link that relates the Task entity to the Project entity. This corresponds to the following triple:

Task **owned by** Project

When you define and change objects using the Diagrammer, your changes are reflected everywhere in the model. Similarly, if you delete a triple in the Model Editor, any corresponding links on diagrams are also removed.

Creating a Diagram

You defined Project in the previous chapter, and Employee was already defined before you started. In the following steps, you create an entity-relations diagram. On the diagram, you create the Task entity, and specify the relationships between the different entities.

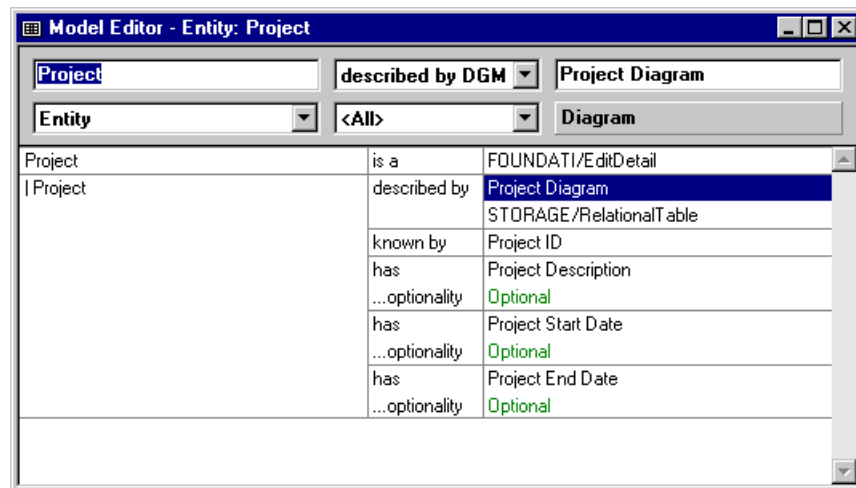
To create and open a diagram:

1. If the Model Editor is not open, open it.
2. Make sure the object type field is set to Entity, then add the following triple:

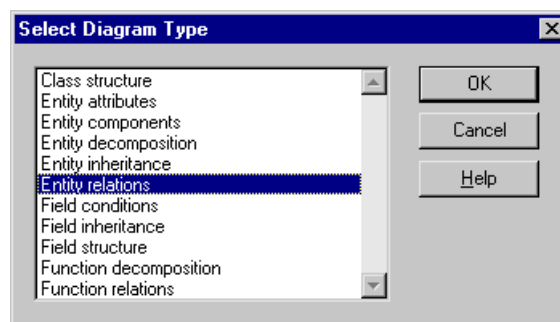
Project **described by** Project Diagram

You can now work in the diagram you just created. A diagram is like a canvas, onto which you place nodes, sub-nodes, and links between nodes, to represent your model.

3. Select the Project Diagram object in the Model Editor, and click the Editor toolbar button:



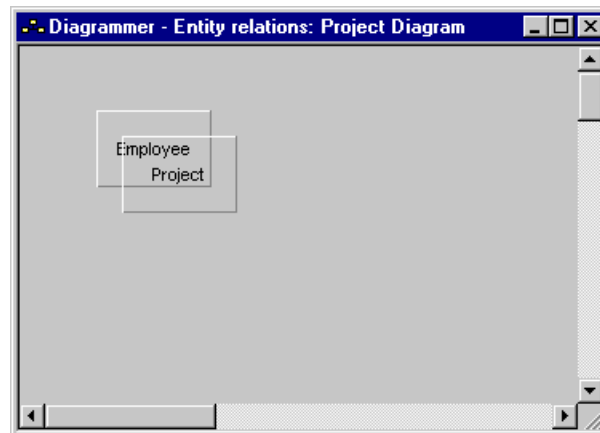
The Diagrammer opens, with the Select Diagram Type dialog over it:



4. Select Entity Relations and click OK.
5. If necessary, move the Diagrammer window so that the Object Browser does not overlap it.
6. If the Object Browser is not displaying entities, click the Entities toolbar button.

7. Drag the Project and Employee entities from the Object Browser to the Diagrammer window.

The Diagrammer window should look something like this:



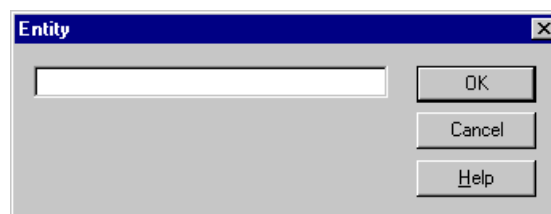
8. Drag the Employee node down and to the right, so that it does not overlap the Project node.

Next, you add the Task entity to the diagram.

9. Right-click below the Project entity.
10. From the pop-up menu, choose Create Node, and then Entity.

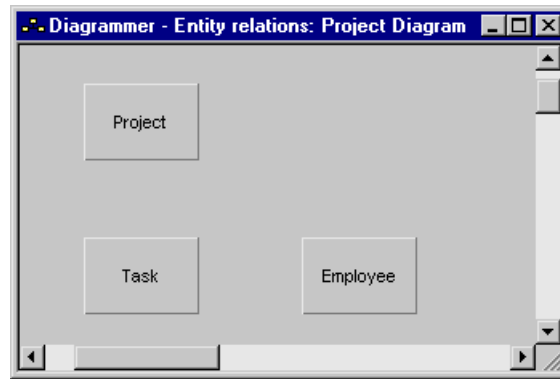
CA Plex adds the new node at the point where you clicked the mouse.

The New Node dialog appears:



Note that the title bar on the dialog says Entity. The title bar shows you the type of node you are defining.

11. Enter **Task**, and click OK.
12. Position the nodes (by dragging) so that the diagram looks something like this:



Defining Relationships with the Diagrammer

In this section, you define two relationships between entities: **an owned by**, and a **refers to** relationship.

In your application, a task is part of a project. Therefore, if you delete a project, you want all of its tasks to be deleted. Because of this, you define Project as the owner of Task, so that each task is directly related to the project it is part of. This is known as an owned by relationship, which is sometimes referred to as a parent-child relationship.

When you define a task that is part of a project, this project management application requires that you specify an employee who is responsible for completing the task. Because an employee can be assigned to more than one task, in more than one project, you do not want to delete the employee record if you delete a task. So, you define a refers to relationship between the Task and Employee entities.

When you add the triple Task refers to Employee, the Task entity stores the identifier of the Employee entity, and uses it as a pointer to that entity. The stored value is called a foreign key.

An employee can be assigned to more than one task, even to more than one task in more than one project, but a task can have only one employee assigned to it.

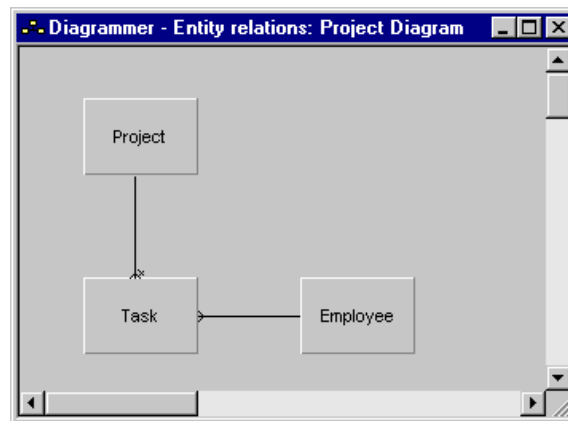
To define relationships in a diagram:

1. Select the Task node and then hold down the CTRL key.
2. With the CTRL key still held down, right-click the Project node.
3. From the pop-up menu, choose Create Link, and then Owned By.

This creates an owned by relationship between the two entities. Next, you create a refers to relationship between Task and Employee.

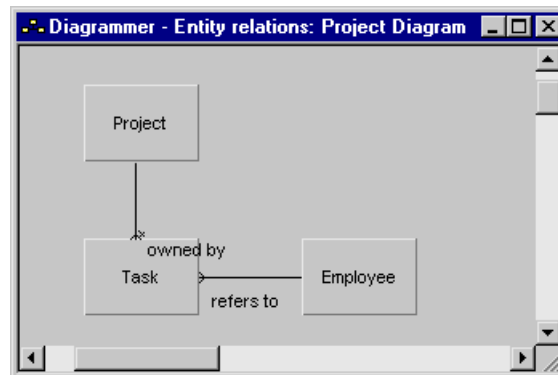
4. Select the Task node.
5. Hold down the CTRL key and right-click the Employee node.
6. From the pop-up menu, choose Create Link, and then Refers To.

The diagram should look something like this:



You can see the relationships between the entities, but the links look the same, with no indication of what kinds of relationships they represent. Next, you change the display so that it shows more information.

- From the View menu, choose Display Link Labels, and then Link Names. The Diagrammer window should look something like this:



- If the link label appears on top of one of the nodes, drag it to a more appropriate location. When you select a label, the associated link turns red to show you which link the label goes with.
- Save your model and close the Diagrammer.

CA Plex Feature: Dynamic Diagrams

You have seen how CA Plex lets you create diagrams showing the objects in the model. These are not just pictures—when you created the link between Task and Employee, you did not simply add a line to a picture, you defined a relationship that was recorded as a triple in the model.

The diagram is automatically synchronized with the model. For example, if you deleted the Task refers to Employee triple in the Model Editor, the corresponding link would automatically be removed from the diagram.

Defining the Task Entity

You have created the Task entity but you have not specified anything else about it. In the next series of steps, you add triples to define the Task entity. After you have defined the Task entity, you learn more about the Panel Designer, and you use it to modify the inherited panels.

To define fields for the Task entity:

- Add the following triples:
 - Task **known by** Task ID
 - Task **has** Task Description

2. Add the following triples to define the properties of these fields:

Task ID **is a** FIELDS/Identifier

Task Description **is a** FIELDS/ShortDescription

These are similar to the triples you used to define the corresponding fields on the Project entity in the chapter "Your First CA Plex Application in 20 Minutes," except for the FLD **length** NBR triple. The FLD **length** NBR triple enables you to override the field length that you inherited from FIELDS/ShortDescription. This is a simple example of one of the ways you can customize the effects of a pattern library object.

3. Refresh your Model Editor.

Now that you have defined the fields for the Task entity, you need to specify which patterns you want the Task entity to inherit from.

To specify what pattern library objects the Task entity inherits from:

1. Add the following triples:

Task **is a** FOUNDATI/EditDetail

Task **is a** STORAGE/RelationalTable

Task **is a** FOUNDATI/OwnedCascade

You have already learned about the EditDetail and RelationalTable patterns, but the OwnedCascade entity is new.

OwnedCascade connects Task with a parent entity so that each record in the Task entity belongs to a record of the parent entity. When the parent entity that owns the Task records is deleted, the Task records are deleted as well. This is known as cascade deletion.

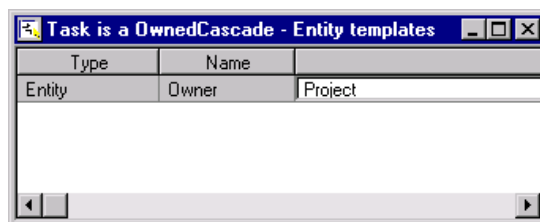
In the diagram, you added the relationship Task **owned by** Project, so Project is Task's parent. The OwnedCascade pattern contains an abstract (placeholder) parent, called Owner, which you need to replace with a real owner entity.

Next, you replace the placeholder in the FOUNDATI/OwnedCascade pattern with the Project entity in your model.

2. In the Model Editor, select the Task is a FOUNDATI/OwnedCascade triple (make sure the whole line is highlighted) and click the Editor toolbar button.



The Template Editor appears:



3. In the blank column, enter **Project**.
4. Close the Template Editor, saving the changes when prompted.
5. If you refresh the Model Editor, you should see that the Template Editor has added the following triples:

Task **replaces** FOUNDATI/Owner
...**by** Project

6. Refresh the Object Browser. If the Task entity is not expanded, expand it.



Notice that Task has another view called SuperKeys in addition to the Fetch and Update views inherited from EditDetail. The SuperKeys view stores the key fields of the owner of the entity—in this case, Project. Unlike Fetch and Update, SuperKeys does not have any scoped functions. Views are defined in CA Plex Object Types in the chapter “Your First CA Plex Application in 20 Minutes.” You learn more about them later in this chapter.

Now you have defined the Task entity. You have specified what pattern library entities it should inherit from, defined its fields, and the properties of the fields.

Generating and Building the Employee Entity

Since you last built the project management application, you have created the Task entity and specified relationships between it and the other two entities. None of the changes you have made affect the Project entity directly, so you do not need to regenerate and rebuild that entity. You have not yet generated and built the Employee entity, so that is the next step. For more information on the process of generating and building, see Generating and Building in the chapter “Your First CA Plex Application in 20 Minutes.”

The Employee entity also inherits its user interface from FOUNDATION/EditDetail, and its database access functions from STORAGE/RelationalTable. It has five fields: Employee ID, Employee Name, Employee Hire Status, and Employee Email Address. Employee Hire Status is a status field, which displays as a drop-down list on the Employee panel.

To generate and build Employee:

1. Open the Generate and Build window, and select Employee. (If Employee is not in the window, click the Refresh button.)
2. Generate and build the entity and its scoped objects.

You do not build the Task entity until the following chapter, “Creating a Wizard.”

Adding Employee Records

In this section, you add employees by running the function scoped by the Employee entity.

To add employees to the database:

1. In the Generate and Build window, select Employee.Edit.
2. Click the Run toolbar button.



3. Choose Plex r7.0 Tutorial.dsn when prompted to select a data source.
4. Select the Continue New option.

When this option is selected, you do not have to click New when entering a series of records.

5. Add some employees, clicking Apply to add each. Here is some example data you can use.

Employee ID	Name	Hire Status	Email Address
Emp01	Heather	Contract	Heather@No.Where
Emp02	David	Full-time	David@No.Where
Emp03	Martha	Part-time	Martha@No.Where

6. You may notice that the grid on the left does not show the new employees you have just added. This is the default behavior when adding a series of records using the Continue New option.

Click the Refresh toolbar button above the grid to show the records.



7. Close the dialog.
8. Add the following triple to your model to keep the data you have entered from being lost the next time you generate and build the Employee entity:

Employee.Physical Table **implement SYS No**

For more information, see Preserving Data in the chapter "Your First CA Plex Application in 20 Minutes."

9. Save your model.

Chapter Review

In this chapter you:

- Used the Diagrammer to create a diagram showing the entity relationships in your application
- Added new nodes to the diagram for the Task and Employee entities
- Specified the attributes of the Task entity, and inherited from pattern library fields to define their properties
- Inherited from the pattern library entities EditDetail and RelationalTable to provide database access and user interface functions for Task and Employee
- Inherited from OwnedCascade for the Task entity to specify that each task record belongs to a Project record in your database
- Generated and built the Employee entity's database table and the inherited functions, and added some sample data

This chapter introduced the following patterns:

Pattern	Description
 FIELDS/Status	A data field with a group of predefined values.
 FOUNDATION/ReferredTo	Provides a function that lets you select from a list of existing records to fill in a foreign key field in a dialog.
 FOUNDATION/OwnedCascade	An abstract entity, used for the child part of a parent/child relationship, which provides cascade deletion.

This chapter introduced the following triples:

Triple	Description
ENT described by DGM	Associates a diagram object with the source entity.
ENT owned by ENT	Defines a parent/child relationship between two entities.
ENT refers to ENT	Specifies that one entity references another.
ENT replaces ENT ...by ENT	Replaces an abstract entity with a real entity in your application.

Chapter 4: Creating a Wizard

In this chapter, you define a wizard that enables end users to add a Project. The wizard has two parts:

- One where you specify information about the new project
- One where you add the tasks that are part of the project

By providing a wizard, you give your end users an easy way to define a new project and all of the information that goes with it.

Some designers might argue that this is not the best use of a wizard. This tutorial includes it simply as an example to show how quick and easy it is to create sophisticated user interfaces by inheriting from patterns.

This chapter is your first opportunity to work with the Action Diagrammer, which you use to modify inherited code.

This section contains the following topics:

[Defining the Wizard](#) (see page 68)

[Modifying the Wizard Parent Panel](#) (see page 73)

[Action Diagrammer](#) (see page 74)

[Modifying the First Part of the Wizard](#) (see page 77)

[Modifying the Second Part of the Wizard](#) (see page 87)

[Adding Commit and Rollback Processing](#) (see page 91)

[Generating and Building](#) (see page 93)

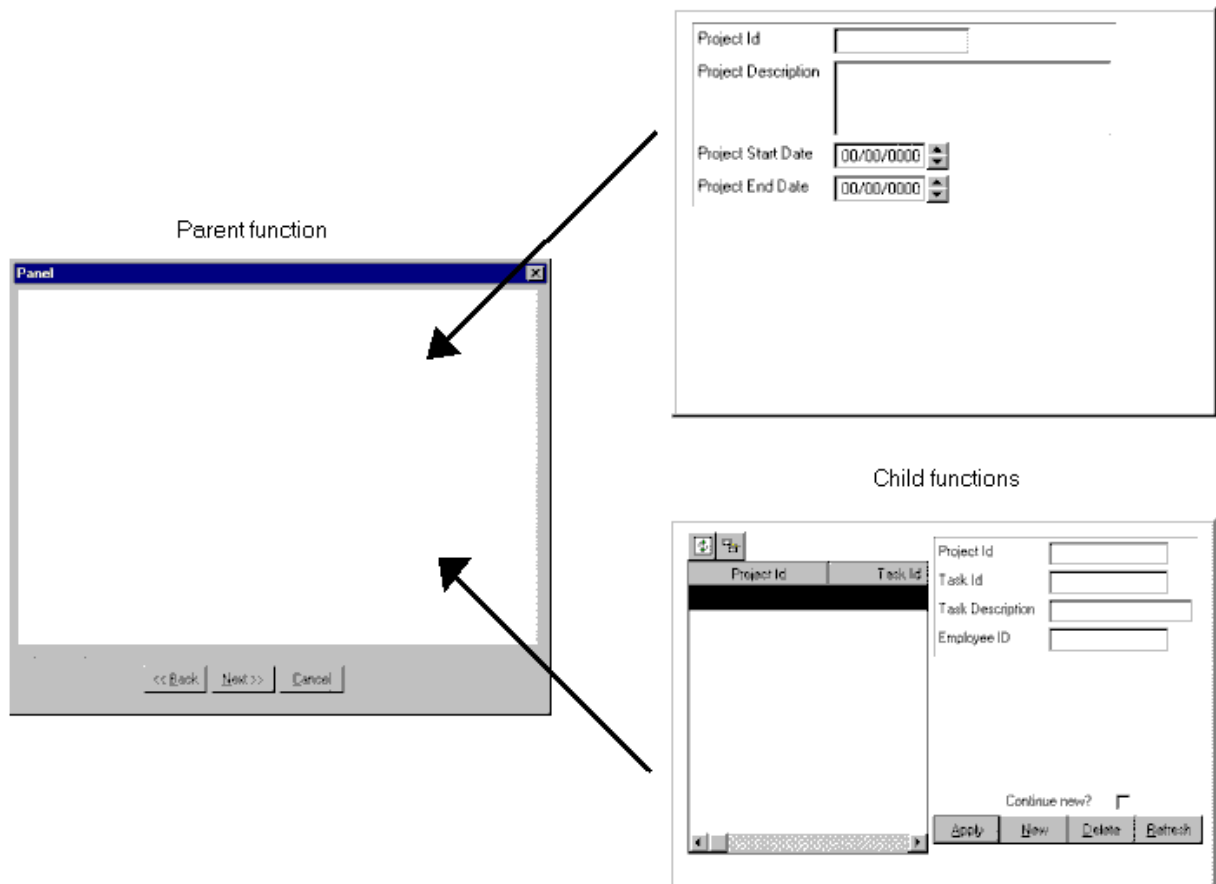
[Testing Your Wizard](#) (see page 93)

[Chapter Review](#) (see page 96)

Defining the Wizard

To create a wizard, you inherit from two patterns in the UISTYLE pattern library. You have not used this library directly, yet.

You create the wizard using a parent function that contains a child site, and two child functions that display panels inside the child site.



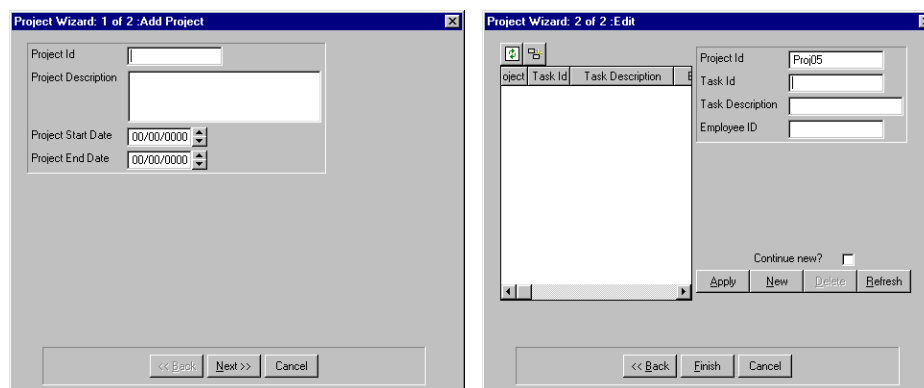
The parent function you create is an unscoped function. This means that it does not appear under either the Project or Task entities, as Project.Edit and

Task.Edit do. To view unscoped functions in the Object Browser, you click the Functions toolbar button.



Wizards are linear. In other words, you run each function in a specific order. You should design the sequence of a wizard before creating its parts. After you determine the order of the parts, you use FNC **comprises** FNC triples to set up the order. This wizard uses the Edit functions you already have for the Project and Task entities, and uses them as the parts of the wizard.

After you define the wizard functions, the user interface looks like this:



To define the wizard functions:

1. Drag the Project entity from the Object Browser to the Model Editor.

This changes the source object type of the Model Editor to Entity, and shows all of the triples currently defined for the Project entity.

2. To define the first function in the wizard sequence, add the following triple:

Project **function** Add Project

Note: Double-clicking a target object in the Model Editor focuses on the object concerned. To reverse the double-click, press the Esc key.

3. To define the details of this function, first double-click Add Project in the body of the Model Editor (in blue). Now add the following triples:

Project.Add Project **is a** FNC UISTYLE/FrameChild

Project.Add Project **is a** FNC UISTYLE/EditInsert

Project.Add Project **replaces** VW UISTYLE/UIStyle.Update

...by VW Project.Update

Note: Remember to select Function in the Object Browser if you are dragging and dropping.

The Project.Update view can be found by expanding the Project entity in the Object Browser.

When you inherit from FrameChild, the panel scoped to your function becomes a child panel, which you can display in a child site on the parent panel of the wizard.

The UISTYLE/EditInsert pattern provides a dialog that lets you enter a record, which is then validated before insertion in the database.

The previous **replaces VW** triple specifies which view is used by the Add Project function to display and update records. Note that there are a number of **replaces** verbs available for functions—make sure you select **replaces VW**.

4. To define the second function in the wizard sequence, add the following triple:
Task.Edit **is a** FNC UISTYLE/FrameChild

Note: To get the Task.Edit function into the source object box, drag it from the Object Browser.

You created the Task.Edit function in the previous chapter, "Modeling Entity Relationships." Adding the **is a** triple modifies the function so that it can run as part of wizard.

5. In the Model Editor, set the object type to Function.

6. Add the following triples:

Project Wizard **is a** FNC UISTYLE/FrameWizard

Project Wizard **comprises** Project.Add Project

Project Wizard comprises Task.Edit

You just created an unscoped function called Project Wizard.

When you inherit from FrameWizard, you get a panel with Back, Next, and Cancel buttons; a caption; and some code that performs a wizard's generic functions, such as moving back and forth through the parts.

The **comprises** triples specify which functions are called by the FrameWizard and appear as the actual parts of wizard.

Note: The sequence of these two triples is important since it determines the sequence in which the parts appear at run time. If necessary, you can change the sequence of triples by dragging and dropping them in the Model Editor.

7. Save your model.

Running the Basic Wizard

The series of triples you entered in the previous section provide the basic definition for the wizard. It is not yet complete—there are some user interface refinements and additional processing we need to add. However, it is useful to generate, build and run the wizard now. This lets you see how the basic wizard works and it helps you understand the additional changes you will make in the rest of this chapter.

Since you last built the model, you defined two new functions, Project Wizard and Project.AddUpdate. You also modified the function Task.Edit to inherit from UISTYLE/FrameChild. You have not yet generated and built the Task entity.

To generate and build the Task Entity:

1. If you do not have a Generate and Build window open, click the New Gen and Build toolbar button.

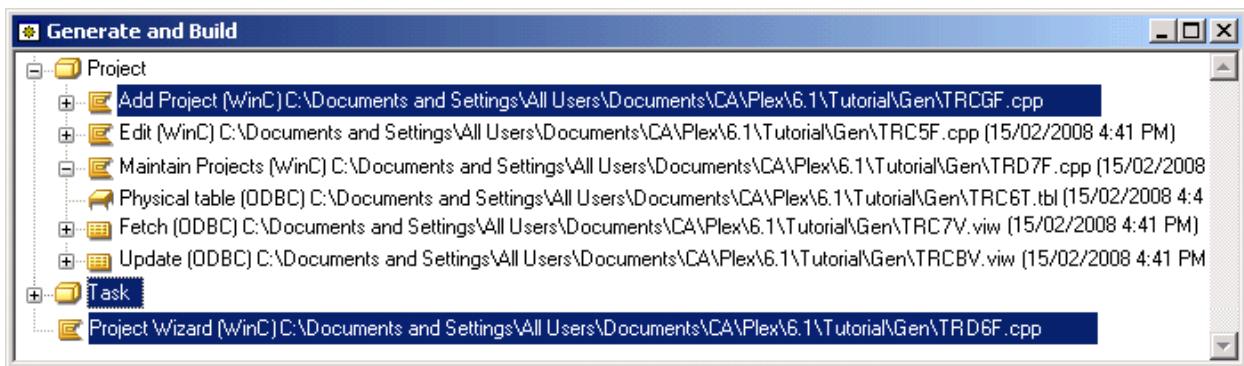


2. If you do have one open, reload it.



3. Select the following objects:

- The Project.Add Project function
- The Project Wizard function
- The Task entity (make sure it is collapsed so that all of its scoped objects are also generated and built)



4. Generate and build these objects. For more information on generating and building, see Generating and Building in the chapter "Your First CA Plex Application in 20 Minutes."



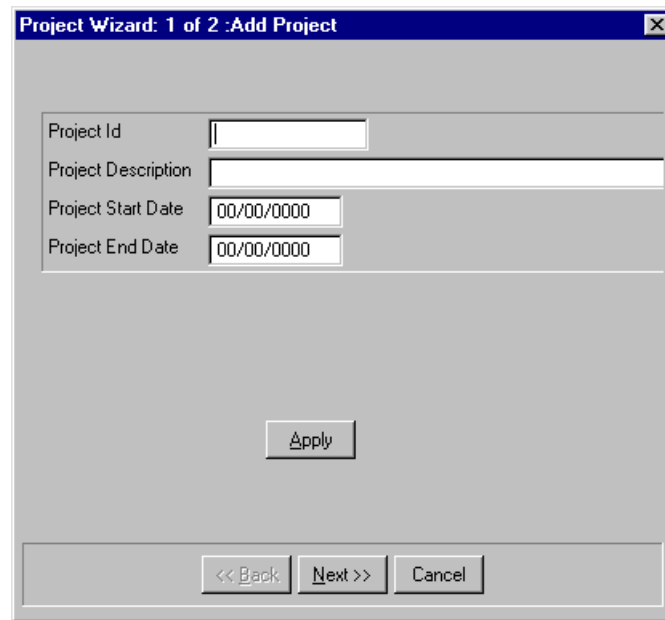
To test the wizard:

1. In the Generate and Build window, select Project Wizard.
2. Click the Run toolbar button.



3. When prompted for a data source, select Plex r7.0 Tutorial.dsn and click OK.

The first part of the wizard appears:

A screenshot of a Windows-style dialog box titled "Project Wizard: 1 of 2 :Add Project". The dialog has a grey background and a blue title bar. It contains four input fields: "Project Id" (a single-line text box), "Project Description" (a multi-line text box), "Project Start Date" (a date picker showing "00/00/0000"), and "Project End Date" (a date picker showing "00/00/0000"). Below these fields is an "Apply" button. At the bottom of the dialog are three buttons: "<< Back", "Next >>", and "Cancel".

Try out the wizard and note following:

- To create a new Project record, enter the required values and click Apply. It would be better if clicking Next caused the record to be created. If that were so, the Apply button would not be needed at all.
- The Project Description is not a multi-line edit control and the date fields do not have spin controls.
- After you click Next, note that the controls on the second part of the wizard do not fit properly on the dialog.
- Finally, after you create a Project record and click Next, the value you entered in the Project ID field does not appear in the Project ID field on the second part of the Wizard.

These limitations are addressed in the remainder of this chapter.

4. Close the wizard.

Modifying the Wizard Parent Panel

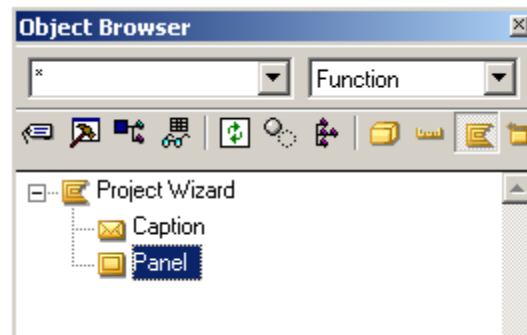
When you inherit from the pattern `UISTYLE/FrameWizard`, the panel scoped by it has a child site in which the child panels of the wizard are displayed.

You determine the size that the child site in a wizard needs to be by opening each of the panels that are displayed in the site, and noting their size. You record the largest height and width of the child panels, and set the child site to those dimensions.

The child site defined on the panel is too small to hold the panel scoped by `Task.Edit`, which measures 454 by 311 pixels (you find a panel's size by opening the panel and looking in the Property Sheet). Also, the child site extends to the bottom of the panel, which means that the child panels overlap the buttons unless you move the button region.

To change the wizard parent panel:

1. Using the Object Browser, open `Project Wizard.Panel` (double-click `Panel`):



2. In the Panel Palette, select `Panel` (the top-level element):



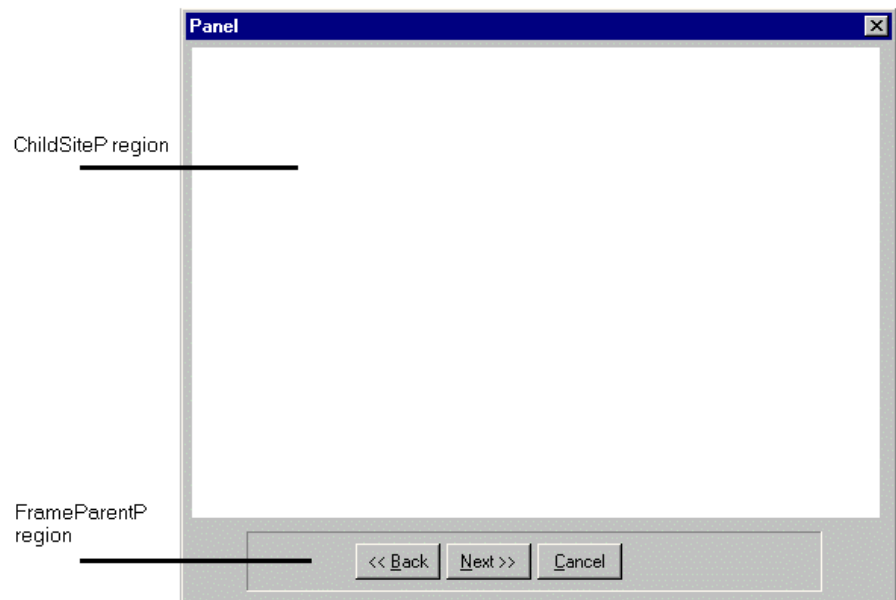
3. In the Property Sheet, change the `Size` property to 475,400 to make the whole panel bigger.
4. In the Panel Palette, expand `Panel` and select the `ChildSiteP` region.
This is the child site region, where the child panels appear.
5. In the Property Sheet, change the `Size` property to 454,311.

6. In the Panel Palette, select the FrameParentP region.

This region is the beveled box that holds the Back, Next, and Cancel buttons.

7. In the Design Window, drag the box below the child site. To drag the entire region, first position the mouse pointer over the small yellow box that appears in the top-left corner of the region when it is selected.

The panel should look something like this:



8. Save your model and close the Panel Designer.

Action Diagrammer

CA Plex functions are objects that perform the activities, processes, and procedures that constitute the functionality of an application. Functions can:

- Display panels and control how end users interact with the user interface
- Create, read, update, and delete records in the database
- Perform calculations

Associated with each function is an action diagram, which contains the code that specifies what the function does. The Action Diagrammer is used to edit a function's action diagram. This editor consists of three windows:

- Action Diagram window, where you enter instructions
- Action Diagram Palette, where you select Action Diagram syntax
- Variable Palette, which displays the variables for the function and their contents

Action Diagram Window

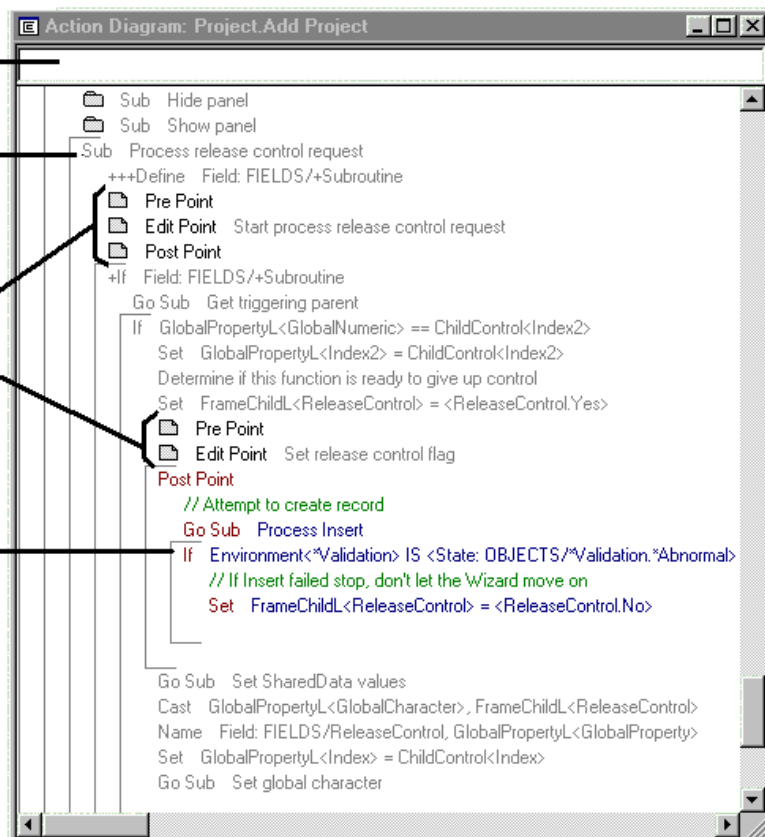
The Action Diagram window displays function logic. The following example shows a portion of what the action diagram for the Project.Add Project function looks like when you have completed this chapter:

You enter new lines of code in the input line. Each new instruction is added above the currently selected line in the body of the action diagram.

Inherited code appears in gray and cannot be modified.

You can make local modifications in the Pre, Edit, and Post Points.

This is an If construct. Constructs display as verticle brackets that enclose other lines of code.

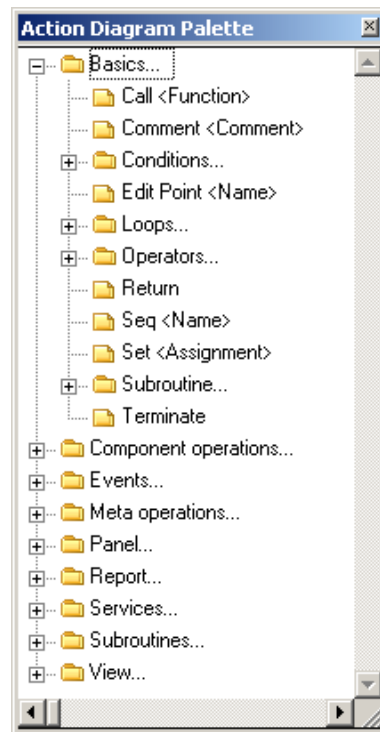


Action Diagram Palette

Use the Action Diagram Palette toolbar button to hide or show the Action Diagram Palette.



The Action Diagram Palette displays the instructions you use to create function logic. It is organized in folders containing statements that are grouped based on the types of actions they perform. You can enter new instructions by typing them in the input line or dragging them to the input line from the Action Diagram Palette.

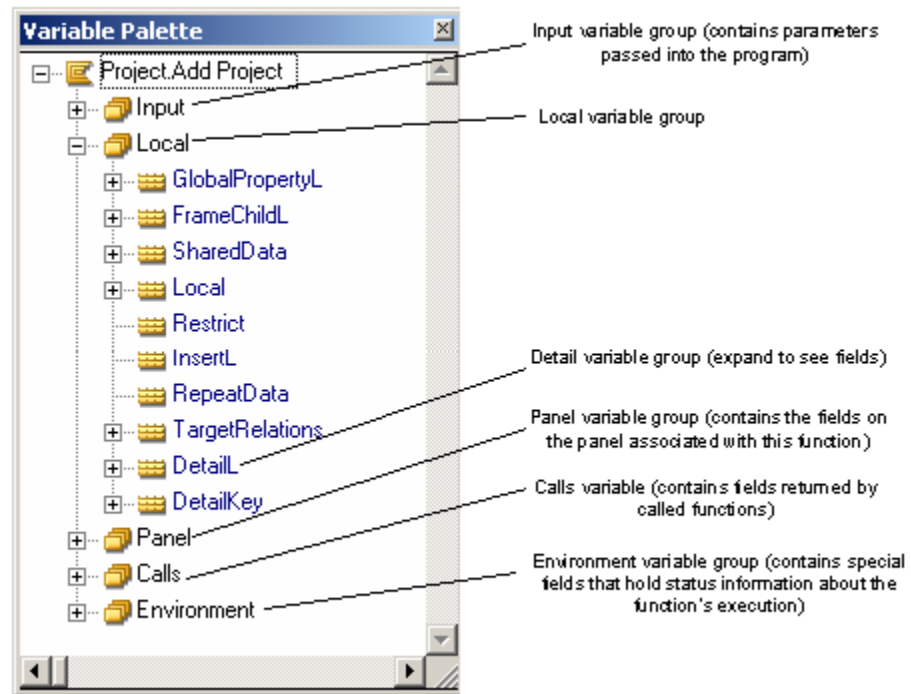


Variable Palette



The Variable Palette toolbar button is used to hide or show the Variable Palette. The Variable Palette lets you view and edit the variables available to a function. Variables in CA Plex are different from variables in other programming languages. A variable in CA Plex is a container for one or more fields. A field must be added to a variable before you can reference it in action diagram code. This is equivalent to declaring variables or fields in many programming languages.

The Variable Palette displays variables in folders, known as variable groups. Each group contains one or more variables. Notice that the icon for a variable looks like three field objects on top of each other.



Modifying the First Part of the Wizard

Modifying the first part of the wizard results in changes that:

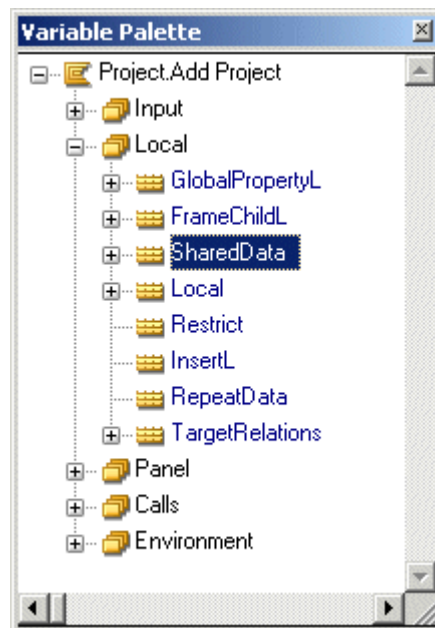
- Permit the value entered for the Project ID in the first part of the wizard to be available in other functions that comprise the wizard
- Cause the Next button to request creation of a Project record and, if successful, to move to the next part of the wizard
- Hide the Apply button on the first part of the wizard and makes some other minor enhancements

Adding a Field to a Variable

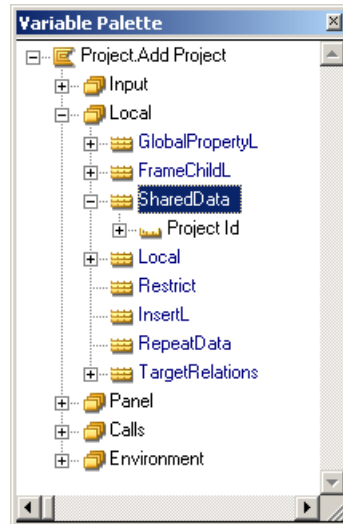
Adding fields to variables is equivalent to declaring variables in many programming languages. You cannot reference a field in an action diagram until you add it to one or more variables.

To add the Project ID field to the SharedData variable:

1. Open the function Project.Add Project.
2. Click Cancel on the Select Edit Point dialog.
3. In the Variable Palette, expand the Local variable group, and then select the SharedData variable:



4. Set the Object Browser to display fields.
5. Drag the Project ID field from the Object Browser onto the SharedData variable in the Variable Palette.
6. Expand the SharedData variable:



Refresh and notice that the variable now contains the Project ID field. This adds the following triples to the model:

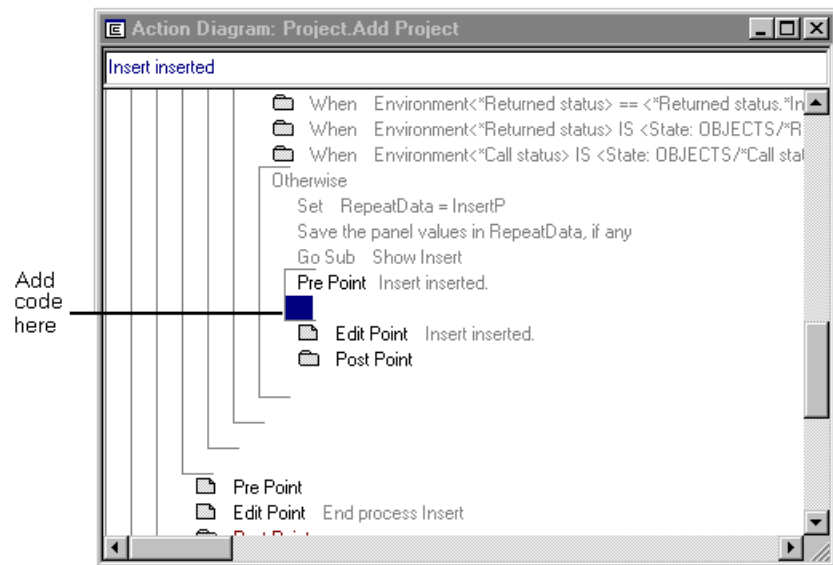
```
Project.Add Project local Project ID
...for UIBASIC/SharedData
```

The Project ID could have been added to the SharedData variable by entering these triples in the Model Editor using drag and drop in the Variable Palette. The result is the same whichever method you use.

7. Keep the Action Diagrammer open for the next section.

Updating the Shared Data

1. Enter **Insert inserted** in the input line of the Action Diagrammer, and press F4.
The Action Diagrammer moves to the Insert Inserted Pre Point. (Pre Points are explained in more detail later.)
2. Select the Pre Point and press the space bar on the keyboard to expand it.
3. Click the line below the Pre Point so that it has focus. This is where you add a line of code. The body of the action diagram should now look like this:



The Insert Inserted Pre Point corresponds to the point at which a new Project record was successfully inserted into the database.

4. Copy the data from the Project ID field in the InsertP region (located in Panel, Project.AddProject in the Variable Palette) into the SharedData variable so that the next part of the wizard can use it.
5. In the input line in the Action Diagrammer, enter the following statement and then press Enter.

Set SharedData<Project ID> = InsertP<Project ID>

This statement copies the data from the Project ID field in the InsertP region into the SharedData variable, so that the next wizard part can use it. Instead of typing the entire statement you could have constructed it using drag and drop techniques, as explained in the next step.

More About Variables and Fields

A variable is a list of fields. Each function has its own set of variables that are used to:

- Pass values between the function and panels, reports, views and other functions
- Temporarily store interim values during processing

When you perform operations on fields in an action diagram you always specify the variable to which the field belongs. The same field can (and typically does) appear in more than one variable. Consider the statement you just entered. The Project ID field appears in both the InsertP and SharedData variables. Each instance of the field can store a different value.

To specify the variable to which a field belongs you use the syntax:

variable <field>

For example, this refers to the instance of Project ID in the InsertP variable:

InsertP<Project ID>

Global Data Storage and the SharedData Variable

As a general rule, the values stored in action diagram fields are local to the function concerned. Updating the value of a field in one function does not change the value in another function, unless the field concerned is explicitly passed as an input or output parameter on a function call.

Graphical user interfaces such as Windows enable multiple functions to run asynchronously. In such situations, it is useful to share data across multiple functions. Our Project Wizard is a case in point, where we need the Project ID to be available to both parts of the wizard.

This type of global data storage is provided by the SharedData variable. The Pattern Library provides additional processing (in the Get SharedData Values and Set SharedData Values subroutines) that maintains the values of the fields concerned in an area of memory that is available to all the functions in a single Windows client application.

Making the Next Button Add a Record

In this section, you add some processing that causes the Next button on the Wizard to trigger the creation of a new Project record. Bear in mind that:

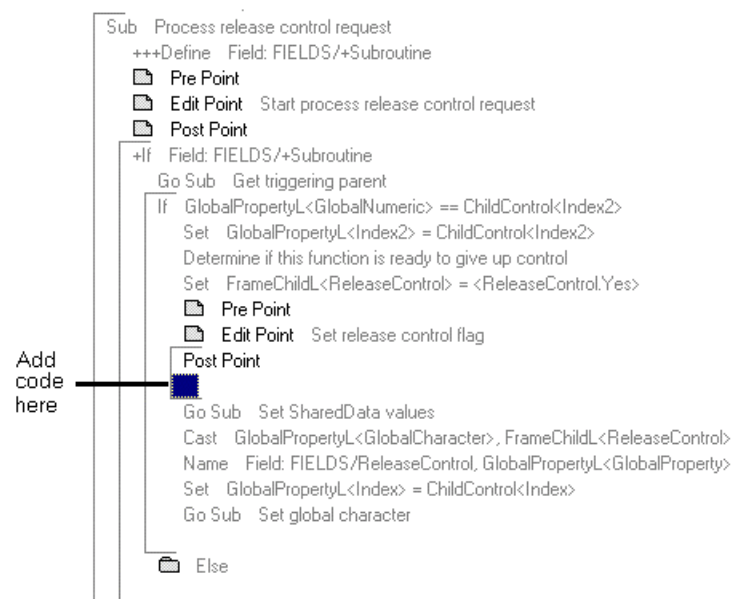
- The Next button is part of the Project Wizard function (inherited from UISTYLE/FrameWizard)
- The processing for adding a record is in the Add Project function (inherited from UISTYLE/EditInsert)

The Next button is needed to trigger an event in a different function. Fortunately, the wizard patterns provide the entire infrastructure required—only a few lines of code need to be added.

When the Next button is clicked, it triggers an event in the Add Project function causing the Add Project panel to be hidden (and then the Task.Edit panel to be shown in its place). The Add Project function has edit/collection points (inherited from UISTYLE/FrameChild) that enable us to intervene in this processing:

1. Enter **Set release control flag** in the input line in the Action Diagrammer, and press F4.
2. The Action Diagrammer moves to the Pre Point before the Set Release Control Flag Edit Point.

Select the Post Point following this Edit Point and expand it by pressing the space bar.



This Post Point corresponds to when end users click the Next or Back buttons to move to a different part of the wizard. At that stage, this you want to create a Project record using the values entered by the user. All the processing required to validate the entered values and then create the record is provided in a subroutine called Process Insert. You simply need to call this subroutine using a Go Sub statement.

3. Enter the following statement into the input line and press Enter:

Go Sub Process Insert

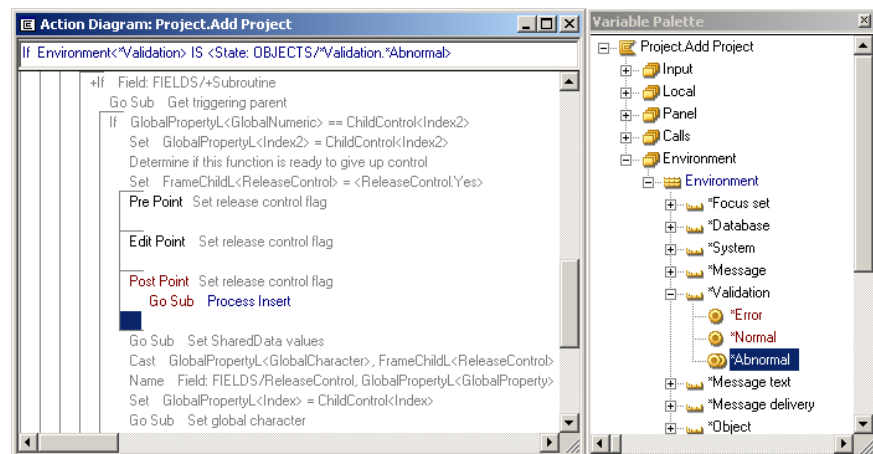
There is one further refinement required. The creation of the record may fail for a variety of reasons. For example, the user may not have entered all the required values or they might try to create a record using the key of an existing record. In these circumstances, you need to prevent the wizard moving to the next screen. There is a simple flag you can set to control this—the ReleaseControl field.

There are a set of fields, Environment fields that indicate whether the creation of a record was successful. The following If instruction uses a compound condition separated by OR operators to test these fields.

Because this instruction is quite long, it is shown on three lines, but you need to enter the entire instruction on one line. You can type the entire line or drag or drop the required objects from the Variable Palette (see the following example).

4. Enter the following instruction immediately after the previous Go Sub statement:

```
If Environment<*Validation> IS <*Validation.*Abnormal>
OR Environment<*Returned status> IS <*Returned status.*Abnormal>
OR Environment<*Call status> IS <*Call status.*Abnormal>
```



Note: It is often easier to enter long action diagram statements by dragging the necessary parts from the palettes rather than typing them. In this example, the state *Validation.*Abnormal was dragged from the Variable Palette to the input line.

5. Enter the following statement inside the If construct you created in the previous step:

Set FrameChildL<ReleaseControl> = <ReleaseControl.No>

This statement prevents the wizard from moving to the second part if the creation of the Project record fails.

Note: You can find the FrameChildL variable in the Local variable group in the Variable Palette.

Checking Your Modifications

You have now added all the necessary code to the Add Project function. As a final check:

1. From the View menu, choose Show Local Modifications CTRL+SHIFT+L.

This restricts the display to show only the code you added. Unless you made a mistake your action diagram should look like the one below. This picture includes some additional comments in the code.

Note: Show Local Modifications is only available from the View menu when the Action Diagram Panel is open.



2. From the View menu, choose Show Full Logic CTRL+SHIFT+F. This resets the display of the Action Diagrammer.
3. Keep the Add Project action diagram open for a later section.

More About Pre, Edit, and Post Points

In this section, you modified the behavior of inherited action diagrams by entering code into its Pre, Edit, or Post Points. These three types of construct are known collectively in the CA Plex documentation as edit/collection points. Edit/collection points are placed in Pattern Library action diagrams at strategic points where the developer of the pattern expects that you will need to make modifications. In team-based development, this approach gives pattern developers control over how their patterns are used. Conversely, it guides developers who use a pattern to the best places to make modifications.

Why Are There Three Types of Edit/Collection Points?

Edit/collection points always appear as a group of three constructs; an Edit Point with a Pre and Post Point before and after.

```

Pre Point  Start initialize
  This comment is inside a Pre Point

Edit Point  Start initialize
  Don't add code here. This is an Edit Point

Post Point  Start initialize
  This comment is inside a Post Point
  ---
  
```

Always enter your code into either a Pre Point or a Post Point. Edit Points are provided for backwards compatibility with earlier versions of CA Plex and should be avoided (see the online help for a full explanation).

Always use a Post Point to enter changes unless you have a particular reason to use the corresponding Pre Point. In Updating the shared data we told you to use the Pre Point. This was because there was code in the Post Point that needed to be executed after the code we added.

Which Pre or Post Point Should I Use?

In this tutorial, we are making things easier for you by specifying exactly which Pre or Post Point to use. In practice, it takes a little time to become familiar with the flow of each Pattern Library function. After you learn the action diagram syntax, you can examine the inherited code to understand how it works. The shipped CA Plex Pattern Libraries are also supplied with extensive online help that includes function flow diagrams for many of the major functions.

Note: To get help for a Pattern Library object, select it in the Object Browser and press SHIFT+F1.

Hiding the Apply Button

The Apply button is no longer required on the Add Project function's panel. In this section you hide it. At the same time, some other improvements are made to this panel.

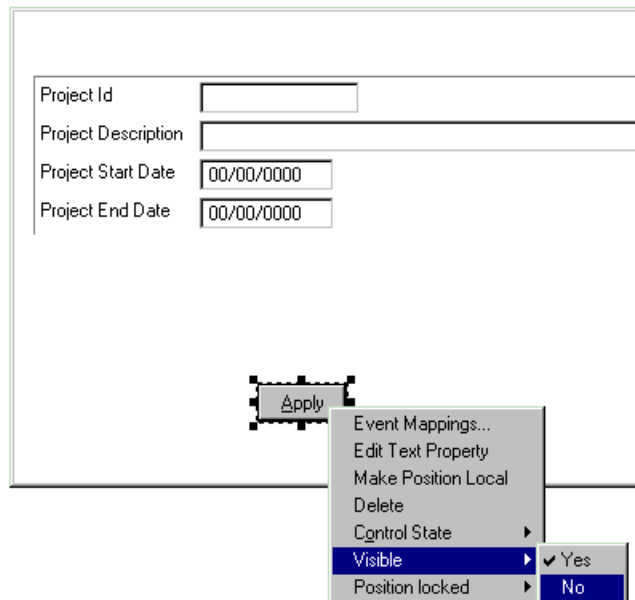
You should have the Add Project action diagram open. Here is a quick way to switch to the panel associated with an action diagram:

1. From the View menu, choose Edit Panel (F11).



The Add Project panel design appears as a white box without the usual title bar and border. This is because this panel is a Child Panel that can only be run within a site on another panel.

2. On the Design window select the Apply button. Click the right-mouse button and set the Visible property to No using the pop-up menu. This is an alternative to setting the property on the Property Sheet.



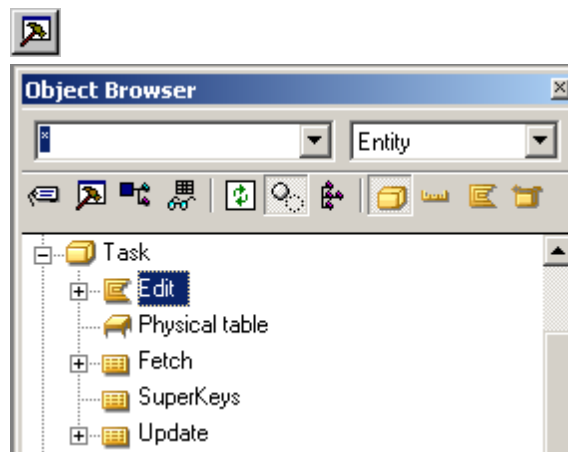
3. Change the Project Description edit control into a multi-line edit control and add a spin control to the date fields. You made these same changes to the Project.Edit panel previously. For more information, see Defining a Multi-Line Edit Control and Adding Spin Controls in the chapter "Your First CA Plex Application in 20 Minutes."
4. That completes the changes to the Add Project function and its panel. Close the Action Diagrammer and the Panel Designer and save your changes.

Modifying the Second Part of the Wizard

You now modify the Task.Edit function to improve the way it works in the wizard. If you recall from running the basic wizard, we need to restrict the display of tasks to show only the tasks for the project currently being added. The ID of the current project is available in global memory (because you added it to the SharedData variable of the AddProject function).

Adding Project ID to the SharedData and Restrict Variables

1. Locate the Task.Edit function in the Object Browser and click the Editor toolbar button.



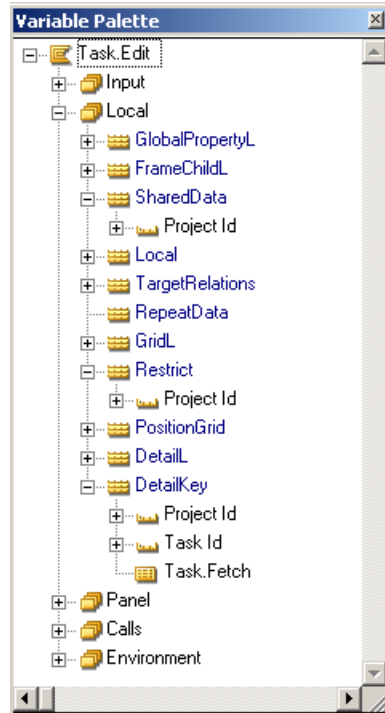
2. Cancel the Select Edit Point dialog.

3. Open the Variable Palette.



4. Expand the Local variable group and then expand the DetailKey variable. Note that the Project ID field is available in this variable.

5. Drag and drop the Project ID field from DetailKey to the Restrict variable. Then drag and drop it again to the SharedData:



You have now added the Project ID field to the SharedData and Restrict variables. This adds the following triples to the model:

```
Task.Edit local Project ID
...for STORAGE/Restrict
Task.Edit local Project ID
...for UIBASIC/SharedData
```

Note: For this exercise, Project ID is dragged from the DetailKey variable. This was just for convenience—it could have also been dragged from the Object Browser and the result would have been the same.

Defining Restrictor Processing

When end users use the wizard, they add a project in the first part, and then add tasks for that project in the second part. The second part uses the Edit function scoped to Task, which contains a list of Task records. This list should not include all of the tasks for all of the projects. Instead, only the tasks you are adding for the new project entered in the first part should appear.

The functionality you set up to accomplish this is called restrictor processing because it restricts database access functions so they only display the records associated with a particular key value—in this case, the primary key of the project.

Adding restrictor processing involves three steps:

1. Informing Task.Edit that you only want it to display the tasks owned by a specific project.
2. Specifying which project's tasks you want it to display.
3. Calling the database access function that returns the list of tasks with the right Project ID.

To use a restricted BlockFetch function:

1. If you do not have a Model Editor open, open one.
2. Drag Task.Edit from the Object Browser to the body of the Model Editor.
3. Add the following triple:

```
Task.Edit replaces FNC Task.Fetch.BlockFetch  
...by Task.Fetch.BlockFetchSet
```

(Make sure you select the **replaces FNC** verb.)

BlockFetch is a function that selects all the rows defined by the Task Fetch view. The triple you just entered tells CA Plex to use a different version of the BlockFetch function. BlockFetchSet only reads a specific set of rows.

Restricting the Display of Tasks

You have now added the Project ID field to the Restrict variable in the Task.Edit function. Remember that Task.Edit is the function that the wizard uses to display its second part.

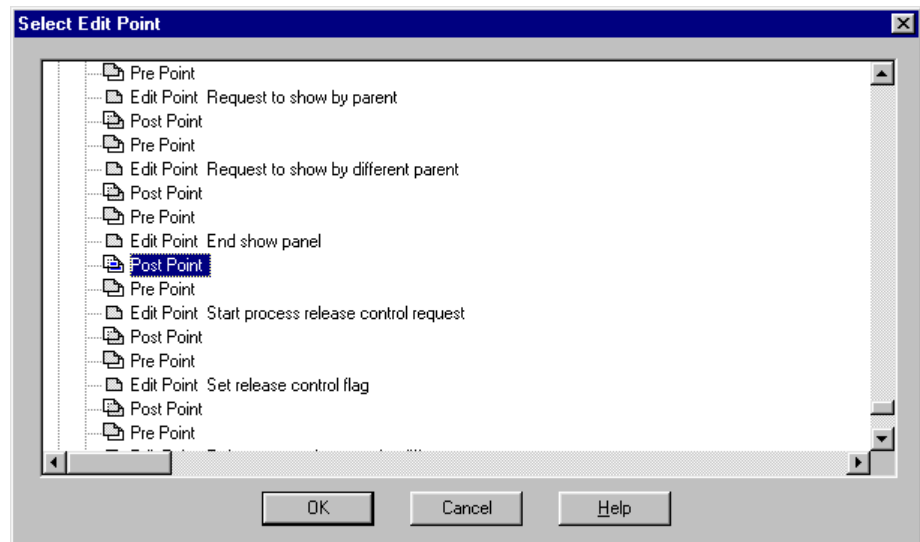
Next, you add action diagram statements to restrict the display of tasks on the second part of the wizard. The first step is to put the Project ID field into the Restrict variable.

To set the Project ID field value:

1. Click the Action Diagrammer to bring the Task.Edit action diagram into focus.
2. On the View menu, ensure that Show Full Logic is selected.
3. From the Block menu, choose Edit Points (CTRL+SHIFT+E).



4. Scroll down and select the End Show Panel Post Point, then click OK. This is Post Point that immediately follows the End Show Panel Post Point:



The Select Edit Point dialog provides a conventional way to locate and focus on a particular Pre or Post Point.

5. Inside the End Show Panel Post Point, enter the following lines of code:

```
Set Restrict<Project ID> = SharedData<Project ID>  
Go Sub ReLoad Grid
```

The Reload Grid subroutine reads the database to retrieve rows to display on the grid. Because you use the Restrict variable to read only the tasks for a single project, the grid only shows those rows. Adding the Project ID field to the Restrict variable, and then setting its value, is like setting up a display filter.

6. Save your model and close the Action Diagrammer.

Adding Commit and Rollback Processing

When end users use a wizard, you give them the option to go back and change things, or to cancel the whole process before they click Finish. If they change their minds and cancel out of the wizard before clicking Finish, you do not want anything added to the database.

You can define your wizard so that it remembers what end users enter, but does not write anything until they complete the process. In database terms, you commit the changes when you are ready to write the records (the end users click Finish). The opposite of commit is rollback, which lets you cancel any changes in memory without making any changes to the database. You can use this when end users click Cancel.

To add commit and rollback processing:

1. Set the Object Browser to display functions.



2. Drag the Project Wizard function from the Object Browser to the body of the Model Editor, and then add the following triple:

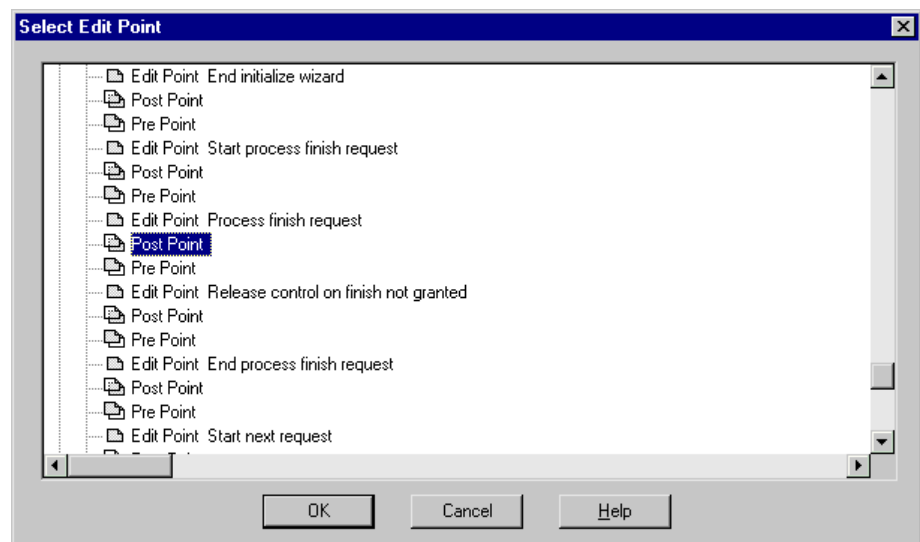
Project Wizard **commit** SYS Parent

This enables the Project Wizard function to commit and rollback the database.

3. In the Object Browser, select Project Wizard and click the Editor button.



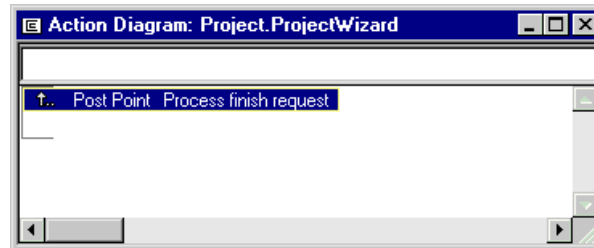
The Action Diagrammer appears, with the Select Edit Point dialog over the Action Diagram window:



4. Select the Post Point that follows the Process Finish Request Edit Point, and click OK.

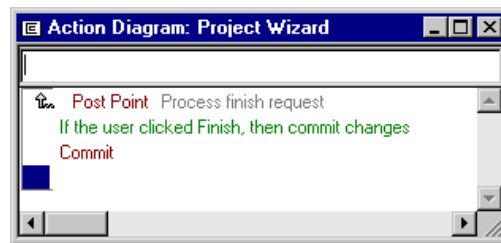
The Process Finish Request Edit Point corresponds with the end user clicking the Finish button.

The Action Diagrammer appears with the Post Point highlighted:



5. In the input line of the window, enter **Commit** and press Enter.

The statement is added to the body of the window:



In Step 2 you added the triple Project Wizard **commit** Parent. This indicated that you want to do commit/rollback processing. Adding the Commit statement indicates *when* you want the transaction committed.

6. Double-click the up arrow (↑) next to the Post Point name in the action diagram.
7. Enter **Start cancel request** in the input line, and press F4.

The Action Diagrammer moves to the Pre Point before the Start Cancel Request Edit Point. This Edit Point corresponds to the end user clicking Cancel on the wizard.

8. Double-click the Post Point after the Edit Point.
9. Add the statement Rollback.
10. Save your model and close the Action Diagrammer.

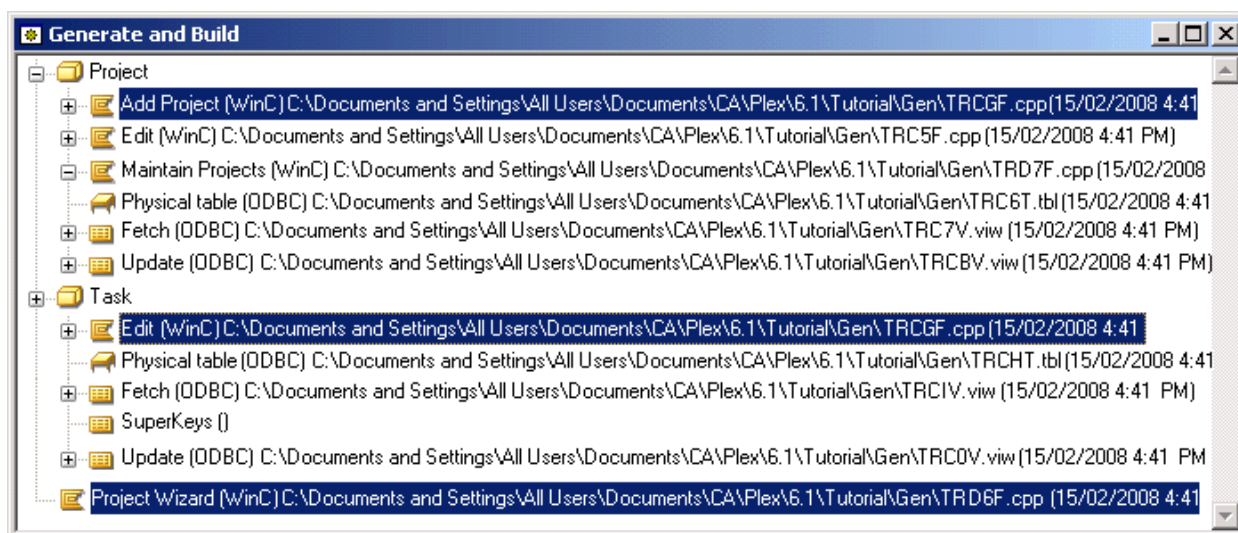
Now, when end users cancel out of the wizard, no records are written to the database, and when the end users click Finish, the records are written to the database. In the next section, you generate, build, and test the wizard.

Generating and Building

Since you last generated, you have modified Project.Add Project and Task.Edit. You also modified the panel scoped by the Project Wizard function.

Note: Remember that when you change a panel, you must generate and build the function that scopes it before you can see the change. You do not generate or build panels directly.

1. On the Generate and Build window, select the functions Project.Add Project, Task.Edit, and Project Wizard.



2. From the Build menu, choose Generate and Build.

For more information on generating and building, see Generating and Building in the chapter "Your First CA Plex Application in 20 Minutes."

Testing Your Wizard

Now you can test the wizard to see how the changes and additions you made work.

For best results, make sure you have entered some Employee records into the database before you test the wizard. For more information, see Adding Employee Records in the Chapter "Modeling Entity Relationships."

To test the wizard:

1. In the Generate and Build window, select Project Wizard.
2. Click the Run toolbar button.



3. When you are prompted to select a data source, select Plex r7.0 Tutorial.dsn and click OK.

The wizard appears:

The screenshot shows a dialog box titled "Project Wizard: 1 of 2 :Add Project". It contains the following fields and controls:

- Project Id:** A text input field.
- Project Description:** A larger text input area.
- Project Start Date:** A date picker showing "00/00/0000".
- Project End Date:** A date picker showing "00/00/0000".
- Navigation buttons:** "<< Back", "Next >>", and "Cancel".

In the next step, you create a new project by entering a Project ID and a description.

Note: You made all of the fields except Project ID (the key field) optional, so you do not have to enter a project start and end date.

4. Enter **Proj04** in the Project ID field.
5. Enter the following text in the Project Description field:
Port client to Java, to enable staff to view email from a browser.
6. Click Next.

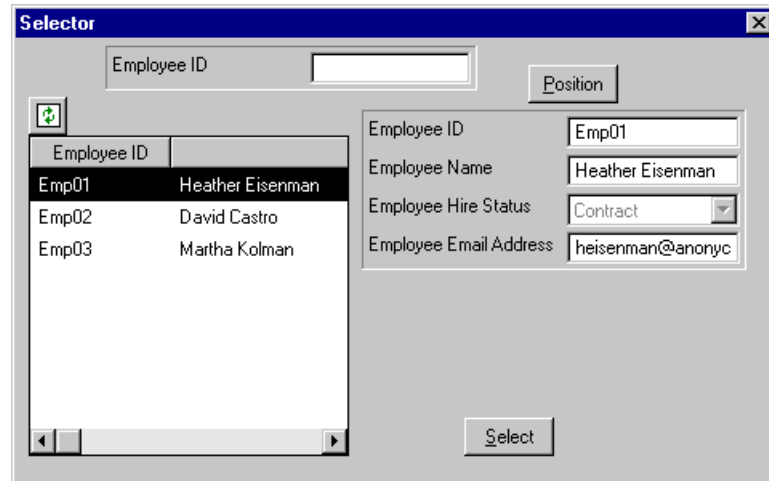
The second part of the wizard appears. Notice that the Project ID field is already filled in for you.

The screenshot shows a dialog box titled "Project Wizard: 2 of 2 :Edit". It contains the following fields and controls:

- Project Id:** A text input field containing "Proj05".
- Task Id:** A text input field.
- Task Description:** A text input field.
- Employee ID:** A text input field.
- Continue new?:** A checkbox.
- Buttons:** "Apply", "New", "Delete", "Refresh", "<< Back", "Finish", and "Cancel".

7. Enter **Task01** in the Task ID field.
8. Enter the following in the Task Description field:
Setup client
9. Double-click the Employee ID field.

The Selector dialog appears:



10. Select an employee from the grid on the left, and click Select.
11. On the wizard, click Apply.
At this stage, you can enter more tasks if you want.
12. Click Finish.

CA Plex Feature: Automatic Selector Functions

You have seen that when entering Task records you can double-click to display a list of possible values for the Employee ID field. This functionality is provided automatically by the CA Plex Pattern Libraries on every panel where you need to select an Employee ID. You do not need to add a single line of code to make it happen.

This is an excellent example of the power that comes from combining model-based development, patterns and code generation. Because the model records the relationship between Task and Employee (Task refers to Employee), CA Plex is able to automatically generate the implied referential integrity and selection processing based on definitions inherited from patterns.

Chapter Review

In this chapter, you:

- Created a two-part wizard for adding a new project and defining the tasks that are part of that project
- Used the Action Diagrammer to modify the logic of an inherited function
- Learned that Pre and Post Points are the places in an action diagram where you can make changes
- Enabled commit and rollback to control when records are written to the database
- Enabled the wizard functions to share data so that you can supply the key of the new Project record when you define Task records

This chapter introduced the following patterns:

Pattern	Description
 UISTYLE/FrameWizard	The parent function for a wizard sequence.
 UISTYLE/FrameChild	A child function that can be part of a wizard or a property sheet.
 UISTYLE/EditInsert	Enables you to add a database record.
 UIBASIC/UIBasic	An abstract entity that scopes template views for primitive user interface functions.

This chapter introduced the following triples:

Triple	Description
FNC is a FNC	The source function inherits the properties of the target function.
ENT function FNC	Creates a function scoped by the source entity.
FNC comprises FNC	Connects the target function to the source function, without scoping one to the other.
FNC replaces VW ... by VW	Replaces an abstract template view with a view in your application.
FNC replaces FNC ... by FNC	Replaces references to one function with references to a different one.

FNC commit SYS	Enables commitment control for database changes.
FNC local FLD ... for VAR	Places a field in a local variable in a function's action diagram.

Chapter 5: Creating a Property Sheet

In the chapter, "Creating a Wizard," you:

- Defined a wizard for adding a new project and the tasks that belong to it
- Added action diagram statements so that the wizard functions could share information with each other
- Added commit and rollback processing so that end users can cancel the wizard and undo any records added

In this chapter, you create a property sheet. Like the wizard, the property sheet uses panel frames to host child functions within a parent function. It enables end users to maintain employees, projects, and tasks.

This section contains the following topics:

[What Is a Property Sheet?](#) (see page 99)

[Defining the Property Sheet](#) (see page 100)

[Creating a Function to Work with Projects and Tasks](#) (see page 101)

[Modifying the Parent Panel](#) (see page 112)

[Setting the Tab Text](#) (see page 115)

[Setting the Caption](#) (see page 116)

[Generating and Building](#) (see page 117)

[Testing Your Property Sheet](#) (see page 117)

[Chapter Review](#) (see page 120)

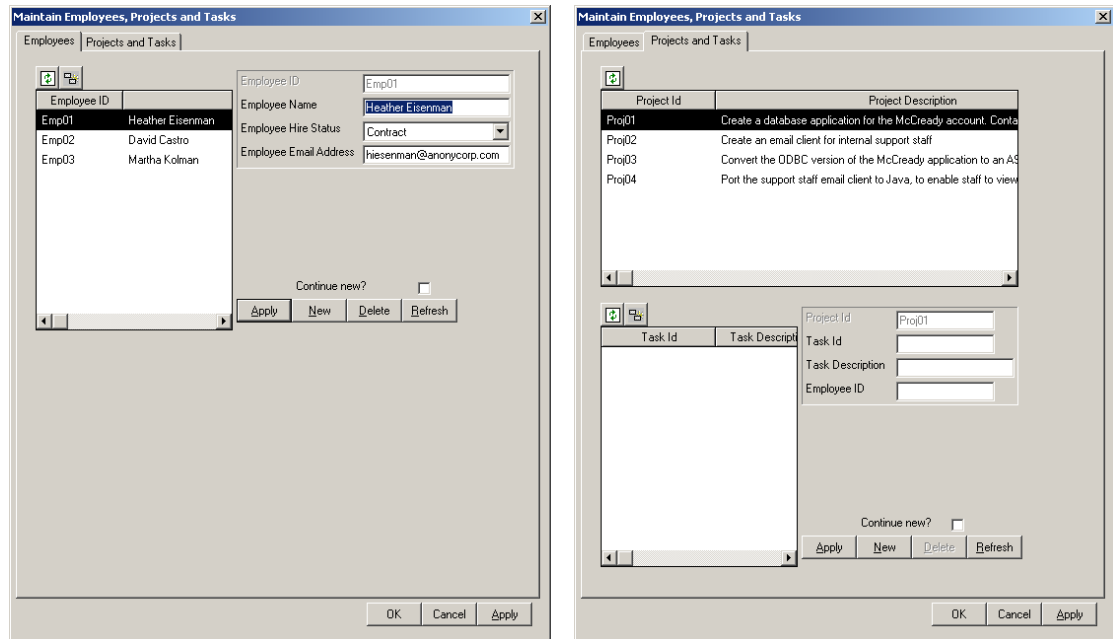
What Is a Property Sheet?

Property sheets are dialogs that contain two or more tabs. When you click a tab, the dialog displays information based on the tab text. Property sheets are also known as tabbed dialogs. CA Plex uses FrameChild functions to display the information for each tab of the property sheet, as it does for wizards.

The tabs that you see on property sheets are provided by an ActiveX control that has been integrated with CA Plex patterns.

In this chapter, you create a property sheet that enables end users to maintain employees and the projects that they defined using the wizard.

When you are done, the two tabs of your property sheet will look like this:



Defining the Property Sheet

Defining a property sheet is similar to defining a wizard. A function is created that inherits from `UISTYLE/FrameProperty`, which gives it the property sheet user interface, and then you specify the functions to use for the tabs on the property sheet. You inherit from `UISTYLE/FrameChild` to define the functions for each tab.

To define the property sheet:

1. Set the Model Editor object type to Function, and then add the following triple:
Project Property Sheet **is a** FNC `UISTYLE/FrameProperty`

As with the function Project Wizard, you create the function Project Property Sheet as an unscoped function.

2. Add the following triple:
Employee.Edit **is a** FNC `UISTYLE/FrameChild`

This triple enables you to use the edit function scoped to Employee on a tab of the property sheet. It turns the scoped panel into a child panel.

3. Add the following triple:

Project Property Sheet **comprises** FNC Employee.Edit

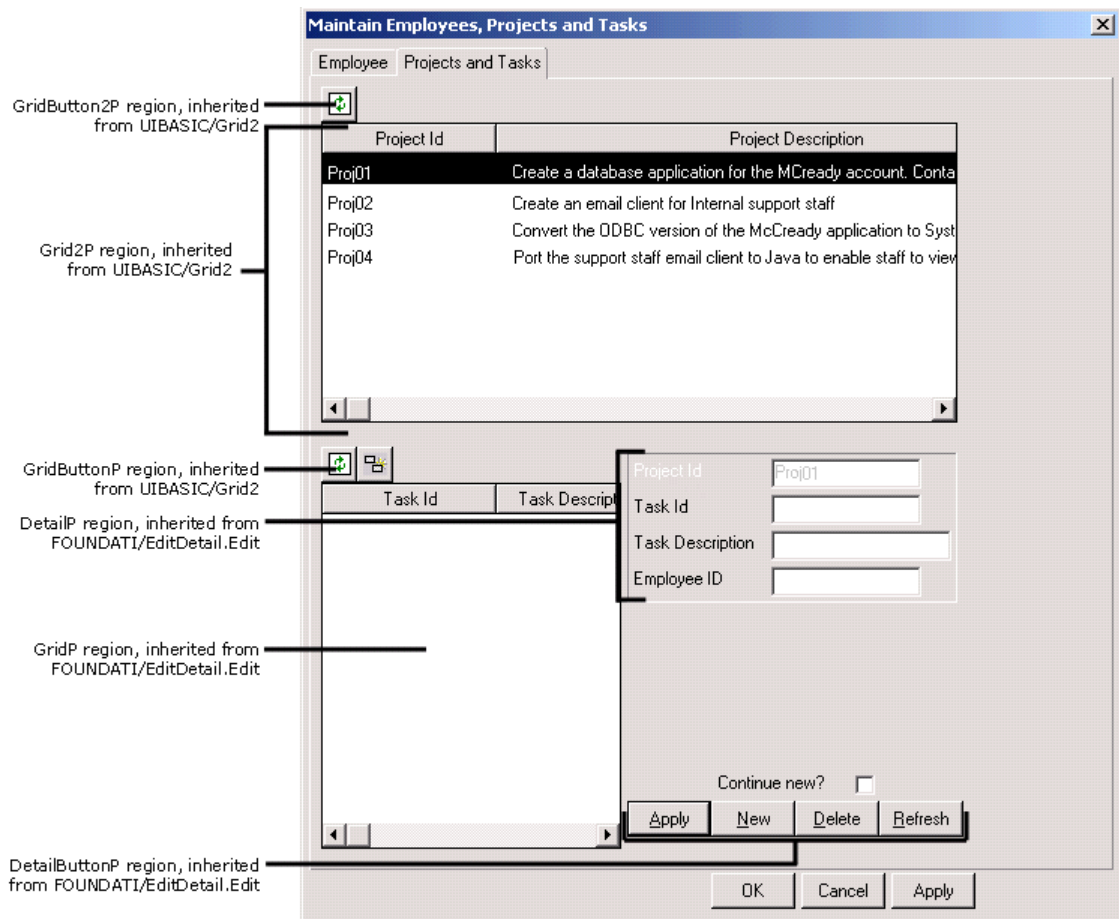
This triple associates Employee.Edit with the first tab on the property sheet. Later in this chapter, you will associate a function that displays both projects and tasks with the second tab.

Creating a Function to Work with Projects and Tasks

You have added the ability to add, delete, and update employee records from the property sheet. Next, you need a way to maintain projects and tasks. You could make both Task.Edit and Project.Edit FrameChild functions, and then put each on its own tab on the property sheet. But you only want to work with tasks in conjunction with the project they belong to, so it makes more sense to see them both on the same tab.

To make this work, you create a new function that shows both project and task data. The panel scoped to the function has a grid on the top that displays projects, and a section at the bottom that displays tasks. When end users select a project at the top, only the tasks that belong to that project are displayed in the bottom.

This function comprises the second tab of the property sheet. When you have added this function to the property sheet, the property sheet looks like this:



To create the function:

1. Drag the Project entity from the Object Browser to the Model Editor, and then add the following triple:
Project **function FNC** Maintain Projects
This defines a new function scoped to the Project entity. You customize this function to work with projects and tasks.
2. Double-click Project.Maintain Projects in the body of the Model Editor to focus on the triples for that object. (The body of the Model Editor actually clears, as there are no triples defined for Maintain Projects function, yet.)
3. Add the following triple:

Project.Maintain Projects **is a** FNC FOUNDATION/EditDetail.Edit

The EditDetail.Edit function is in the Object Browser under the FOUNDATION/EditDetail entity. Inheriting from EditDetail.Edit provides the parts the function needs to work with tasks. The Edit function calls database access functions that are scoped to the Fetch and Update views scoped to EditDetail.

Later, you add replacement triples so that instead of calling the functions and views in the original pattern, your function calls the database access functions and views for your Task entity.

You might wonder why you are not inheriting from Task.Edit instead of inheriting from EditDetail.Edit. You already customized Task.Edit for the wizard, and some of the changes you made will not work quite right for the property sheet. Because your new function works with both projects and tasks together, you define restrictor processing in a slightly different way. Therefore, you inherit directly from the original pattern to define your function.

4. Add the following triples:

Project.Maintain Projects **is a** FNC UIBASIC/Grid2

Project.Maintain Projects **is a** FNC UISTYLE/FrameChild

You inherit from Grid2 because EditDetail.Edit already has a region called GridP, and you want two grid regions in the new function. You must inherit from Grid2 to get a separate Grid2P region; otherwise, with two functions both having GridP regions, they would become a single one in the new function.

To call the correct database access functions, you need to replace the pattern views that the original function expects with the Fetch and Update views scoped to your Task entity.

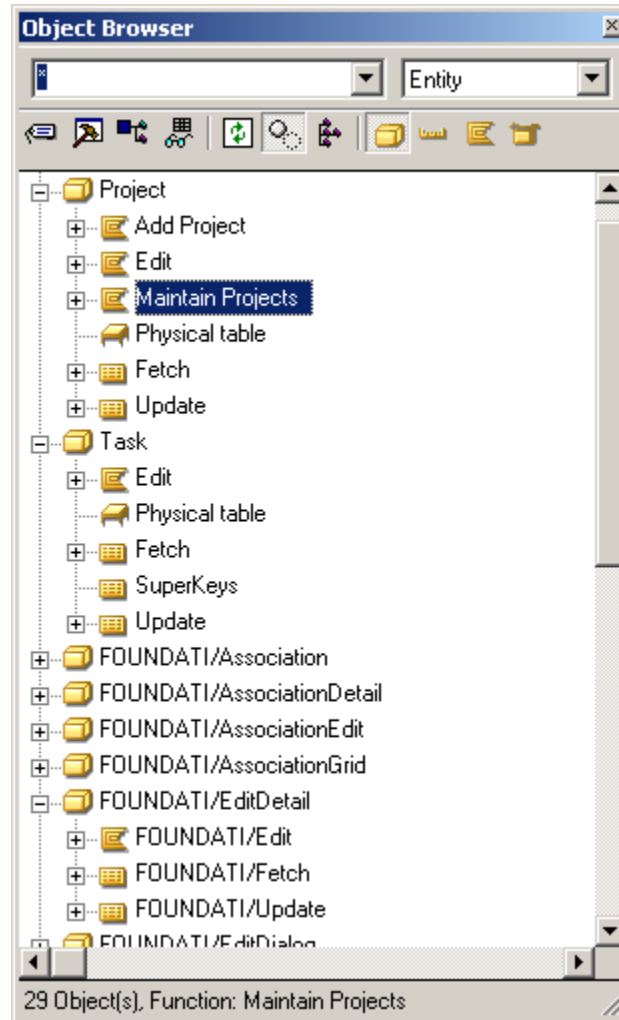
Because you are replacing views, use the verb **replaces** VW in the next step.

5. Add the following triples:

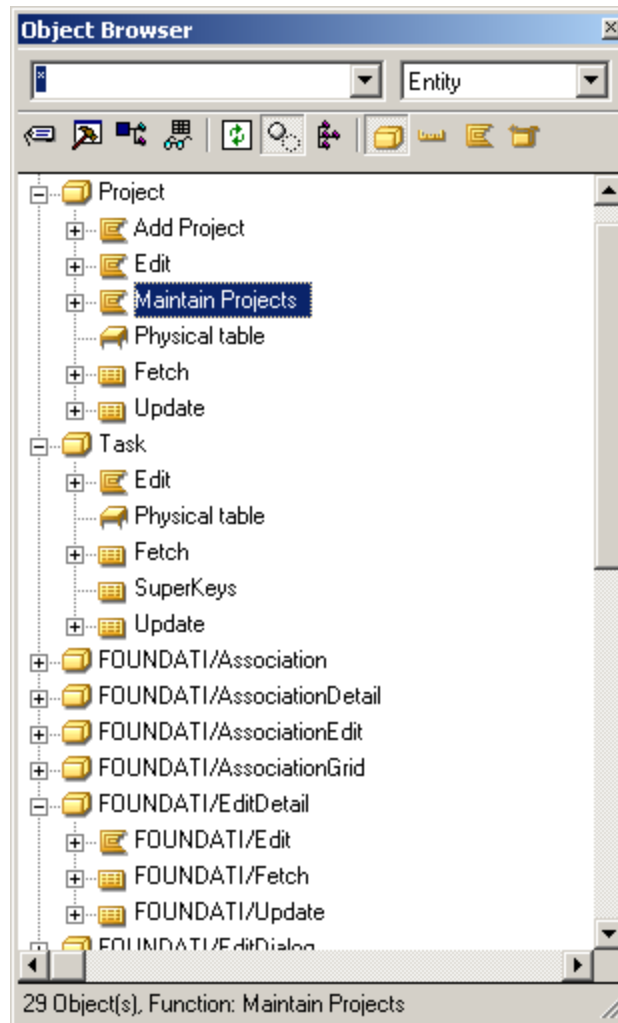
Project.Maintain Projects **replaces** VW FOUNDATION/EditDetail.Fetch
...by VW Task.Fetch

Project.Maintain Projects **replaces** VW FOUNDATION/EditDetail.Update
...by VW Task.Update

Before you add the replacement triples, Maintain Projects calls the abstract functions scoped to EditDetail.Fetch and EditDetail.Update.



After you add the replacement triples, Maintain Projects calls the functions scoped to Task.Fetch and Task.Update.



Inheriting from Grid2 provides the parts needed to work with projects: a grid to display them on the panel, and calls to template database access functions to read the records to display.

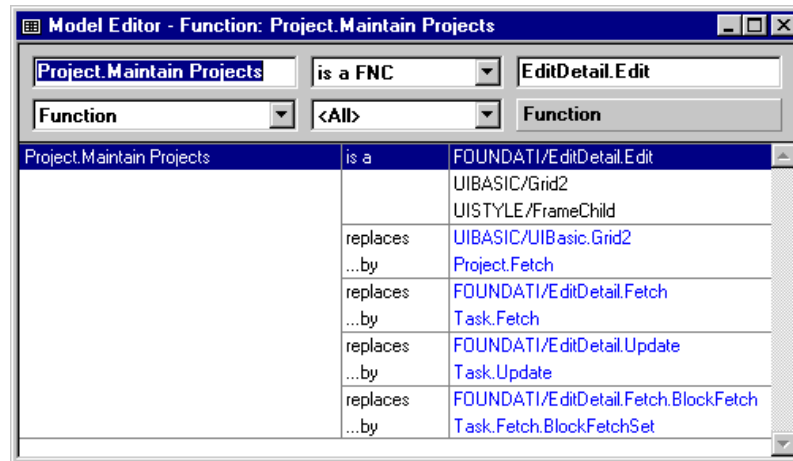
Next, you replace the abstract view Grid2 with the Fetch view from Project, so that the BlockFetch function reads the right records to display on the Project grid.

Note: In the next step, after dragging Project.Maintain Projects to the source object box of the Model Editor, you must change the Object Browser to show entities to find UIBasic.Grid2.

- Refresh your Model Editor, and then add the following view replacement triple:
 Project.Maintain Projects **replaces** VW UIBasic.Grid2
 ...by VW Project.Fetch

In the chapter "Creating a Wizard", you modified Task.Edit so that it would use BlockFetchSet instead of BlockFetch to retrieve only the tasks for the project you specify. You do that again for Project.Maintain Projects, so that the task grid only loads the records that match the selected project.

7. Add the following function replacement triple:
Project.Maintain Projects **replaces FNC** EditDetail.Fetch.BlockFetch
...**by FNC** Task.Fetch.BlockFetchSet
8. Refresh the Model Editor to see the triples you defined for Project.Maintain Projects:



Adding Action Diagram Code

Now that you have created the function, you need to add a bit of code so that the project parts can interact with the task parts. You will be adding code to:

- Load the Project grid when the Maintain Projects tab is displayed
- Restrict the display of tasks so that only the tasks that belong to the selected project are listed
- Load the Project ID field of the Task detail region when you add a new task

To modify Project.Maintain Projects:

1. Open the action diagram for Project.Maintain Projects.

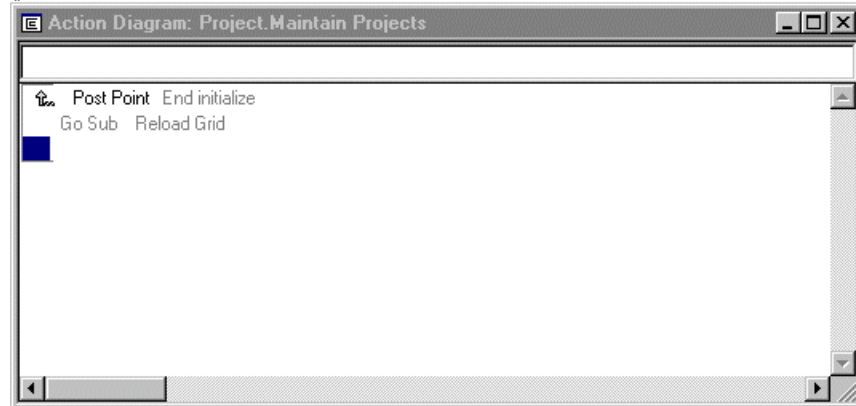
The patterns in UIBASIC are more primitive than the patterns in FOUNDATION. They do not make any assumptions about how you want to use them. The patterns in FOUNDATION are intended to be ready to use with just a generate and build. So, when you use patterns from the UIBASIC Pattern Library, you have to add some initialization and setup code to the part of the function that inherits from Grid2. If you do not do this, projects will not appear on the list until you click the Refresh button.

2. Click Cancel in the Select Edit Point dialog.
3. Enter **End Initialize** in the input line, and press F4.

The following processing is added in the End Initialize Post Point because the grid should be loaded after the rest of the panel is initialized, but before it is displayed.

4. Double-click the Post Point below the End Initialize Edit Point.

This is the first Post Point that already has content:



You can only add code at the end of a Pre or Post Point, below any statements that are already there.

5. Click just below the Go Sub Reload Grid line, and enter the following statement:
Go Sub Reload Grid2

This statement loads the project grid when the panel is initialized.

6. In the Variable Palette, expand the Local variable group.
7. Drag the Project ID field from the Object Browser to Position Grid2.
8. Navigate to the End Initialize Grid2 Post Point.
9. Inside the End Initialize Grid2 Post Point, enter the following statement:
Set PositionGrid2<Project ID> = <Project ID.*Blank>

Note: You can find the PositionGrid2 variable in the Local variable group.

This statement tells the function to read the database starting with the first project. If you did not add this statement, the function that reloads the grid would start wherever it stopped the last time it read projects.

10. In the Variable Palette, expand the Local variable group.
11. Drag the Project ID field from the Object Browser to the Restrict variable.
12. Navigate to the Post Point below the Process Selected Grid2 Row Edit Point.
13. Enter the following statements:

Set Restrict = Grid2P
Go Sub Reload Grid

This sets the values of the fields in Restrict variable equal to the corresponding fields in the currently selected row in the project grid. This means that you only get a list of the tasks for that project from the database, and then it reloads the project grid. Remember, when the function loads the task grid, it uses the BlockFetchSet function, which checks the Restrict variable to determine which tasks to load.

14. Navigate to the Post Point below the End Reload Grid2 Edit Point.

15. Enter the following statement:

For Each Selected Grid2P

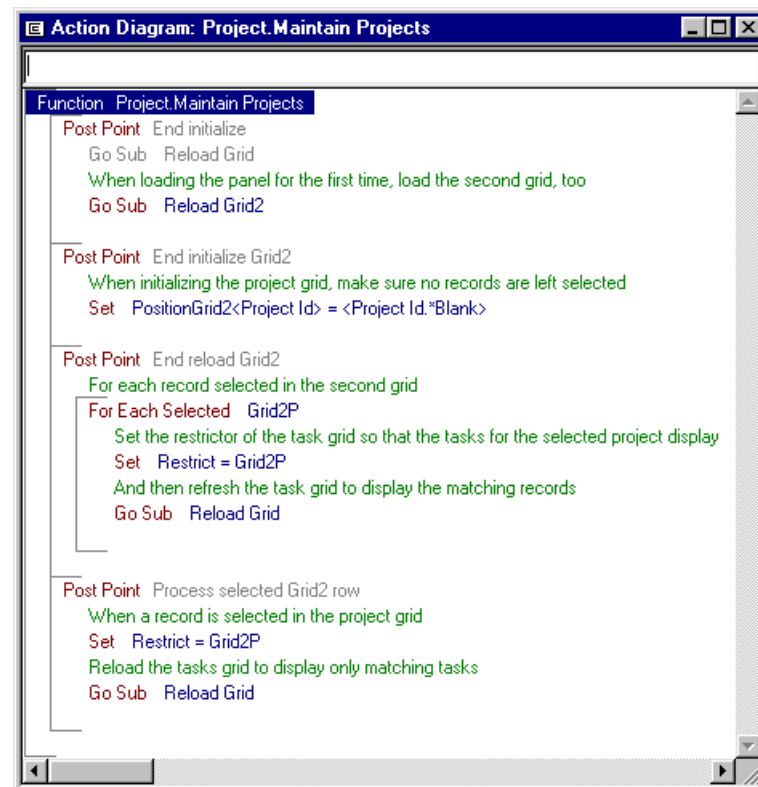
16. Enter the following statements inside the new construct:

Set Restrict = Grid2P
Go Sub Reload Grid

These statements display the tasks associated with any selected projects in the Grid2P region.

17. To check that you entered all the required code, choose Show Local Modifications from the View menu. Use the space bar to expand each construct.

Your action diagram should look like the following example. This example includes extra comments in the code:



18. Reset the action diagram display by choosing Show Full Logic from the View menu.
19. Save your model and close the action diagram.

Modifying the Panel

The panel scoped by Maintain Projects inherited panel elements from the panel scoped to Grid2 and Task.Edit. Next, you will reposition the elements on the panel.

To modify the panel scoped to Maintain Projects:

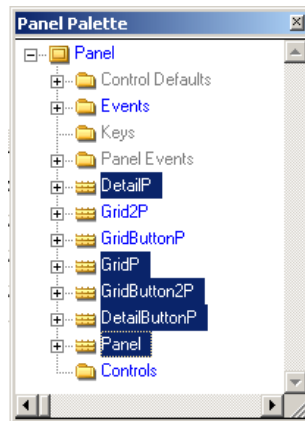
1. Open Project.Maintain Projects.Panel. You may need to refresh the Object Browser to display the panel.

It currently looks like this:

The screenshot shows a dialog box titled "Project Id" and "Task Id". It contains a large empty text area on the left. On the right, there are four input fields labeled "Project Id", "Task Id", "Task Description", and "Employee ID". Below these fields is a checkbox labeled "Continue new?". At the bottom right, there are four buttons: "Apply", "New", "Delete", and "Refresh".

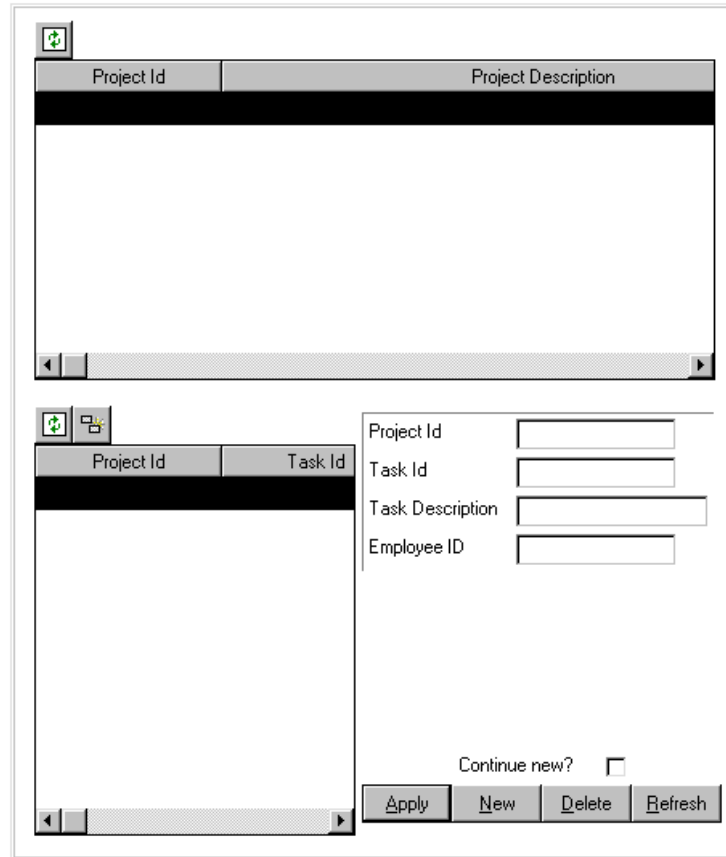
Because you made it a child panel, you have no title bar or dialog frame. You can see the panel elements that it inherited from Task.Edit.Panel, but you cannot yet see the elements inherited from Grid2.Panel.

2. In the Design Window, drag the bottom edge of the dialog to make it about twice as tall.
3. In the Panel Palette, select DetailP, GridP, GridButtonP, DetailButtonP, and Panel:



4. In the Design Window, drag the selected elements to the bottom of the panel.

Note: There is a second grid region—it was behind the one you just moved down. Your panel should look like this.



5. In the Panel Palette, expand Grid2P, and then select the Project Description field.
6. Open the Properties icon. Change the Width property of this field to 400.



7. Widen the entire dialog so that there is some space between the detail region at the bottom and the right edge.
8. Widen the grid at the top of the dialog so that it is about as wide as the bottom section.
9. In the Panel Palette, expand the GridP region and select the Project Id field. Right-click and use the pop-up menu to set the Visible property to No. It is not necessary to display the Project Id in the grid since we are restricting the grid to the tasks for a single project.
10. In the Panel Palette, select Panel at the top.

11. Note the size of the panel (it should be roughly 460, 540). You will need this information later.
12. Save your model and close the panel.

Adding the Function to the Property Sheet

Now that you have defined the function that end users use to work with projects and tasks, you must add it to the property sheet.

To add the function to the property sheet:

1. Drag the Project Property Sheet function from the Object Browser to the body of the Model Editor, and add the following triple:
Project Property Sheet **comprises** Project.Maintain Projects

This adds the Maintain Projects function to the property sheet. The order of the **comprises** triples is important. The first **comprises** triple sets the function that displays the first tab, and the second one sets the function for the second tab.

If the Project.Maintain Projects function is the target of the first triple, then do the following steps.

2. Click the middle column of the Project Property Sheet **comprises** Project.Maintain Projects triple to select the entire triple.
3. Drag the triple onto the Project Property Sheet **comprises** Employee.Edit triple.

This reorders the triples.

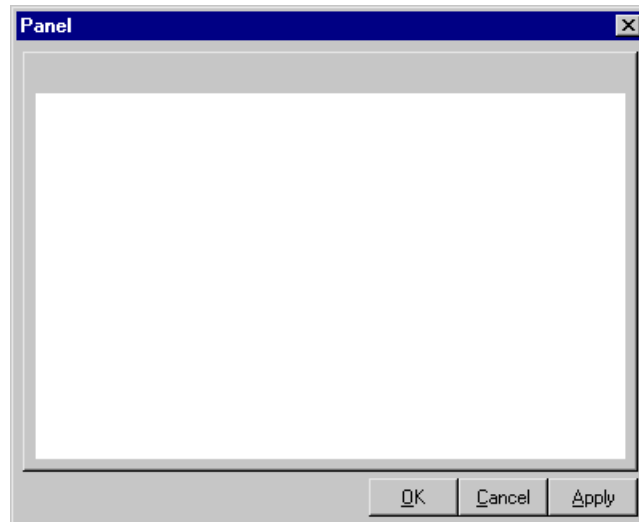
Next, you modify the parent panel so that the child site is large enough to display both the employee tab and the project/task tab.

Modifying the Parent Panel

In the chapter "Creating a Wizard," you modified the wizard panel so that its child site was large enough to display the child panels. In the following steps, you change the panel scoped to the Project Property Sheet function so that it has enough room to display the panels scoped to Employee.Edit and Project.Maintain Projects.

To modify the property sheet panel:

1. Open Project Property Sheet.Panel:



2. In the Panel Palette, expand Panel, and then select ChildSiteP.

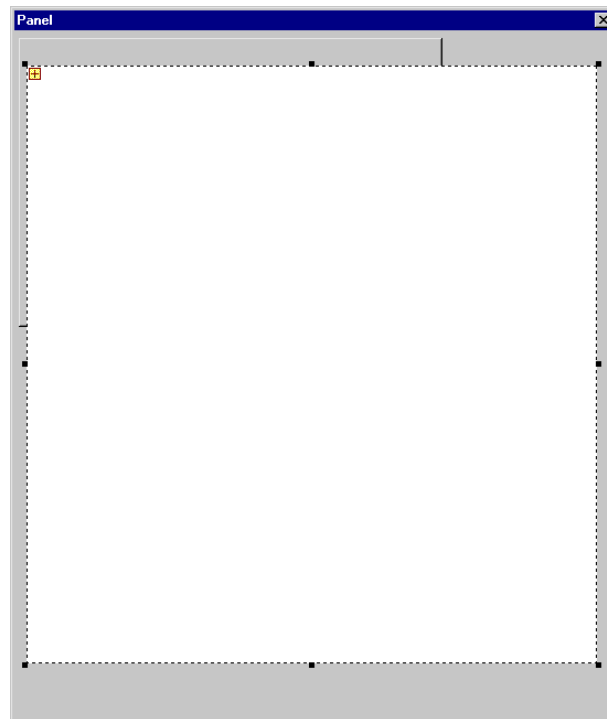
Note that its size is 400, 300 (width, height), which is much shorter than the height of the panel scoped to Project.Maintain Projects, and much narrower than the width of the panel scoped to Employee.Edit.

3. Change the size of ChildSiteP so that it is 515 pixels wide (the width of the panel scoped to Employee.Edit), by 540 pixels tall (the height of the panel scoped to Project.Maintain Projects).

The child site now extends past the right and bottom edges of the dialog.

4. In the Design Window, drag the right edge of the dialog so that it is a little wider than the child site. Then drag the bottom edge so that it extends enough below the bottom edge of the child site to display the OK, Cancel, and Apply buttons.

The buttons are currently hidden behind the child site. The panel should look like this:



5. In the Panel Palette, expand Controls, and select TabStrip1.
This is the part of the panel that contains the actual tabs. FrameProperty inherited this from a function in the ActiveX Pattern Library called ACTIVE/TabStrip.
6. In the Design Window, drag the tab strip's right and bottom edges down so that it is slightly larger than the child site. You can move both edges by grabbing the bottom right handle that appears in the middle of the child region when you select the tab strip.
7. In the Panel Palette, expand FrameParentP and select OK, Cancel, and Apply.
8. In the Design Window, drag the buttons to the bottom right corner of the panel.
9. Save your model and close the Panel Designer.

Setting the Tab Text

By default, the property sheet uses the unscoped name of each child function for the labels on the tabs. Remember that you used `Employee.Edit` for the first tab and `Project.Maintain Projects` for the second tab. The property sheet you created has the text `Edit` on the employee tab and `Maintain Projects` on the project and task tab:



To override this and specify tab labels yourself, you can add FNC **name** NME triples to specify the text to display on that function's tab in the property sheet.

To specify text for the tabs on the property sheet:

1. In the Model Editor, set the object type to Function, and add the following triples:

```
Employee.Edit name NME Employees  
Project.Maintain Projects name NME Projects and Tasks
```

2. Save your model.

When you generate and build the property sheet function, the first tab will read `Employees` and the second tab will read `Projects and Tasks`:



Setting the Caption

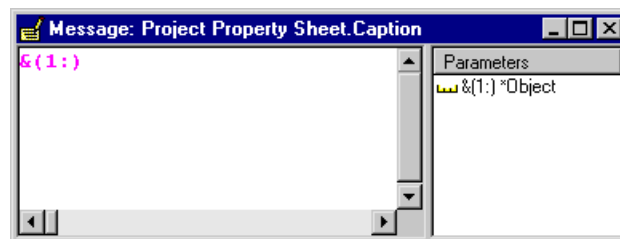
The last step in creating the property sheet is specifying a meaningful caption for the title bar (you inherited a caption when you inherited from `FrameProperty`). You can change the contents of the message object, which then changes the caption on the dialog.

To set the property sheet caption:

1. Set the Object Browser to display functions.
2. Expand Project Property Sheet.
3. Select Caption and click Edit.



The Message Editor appears:

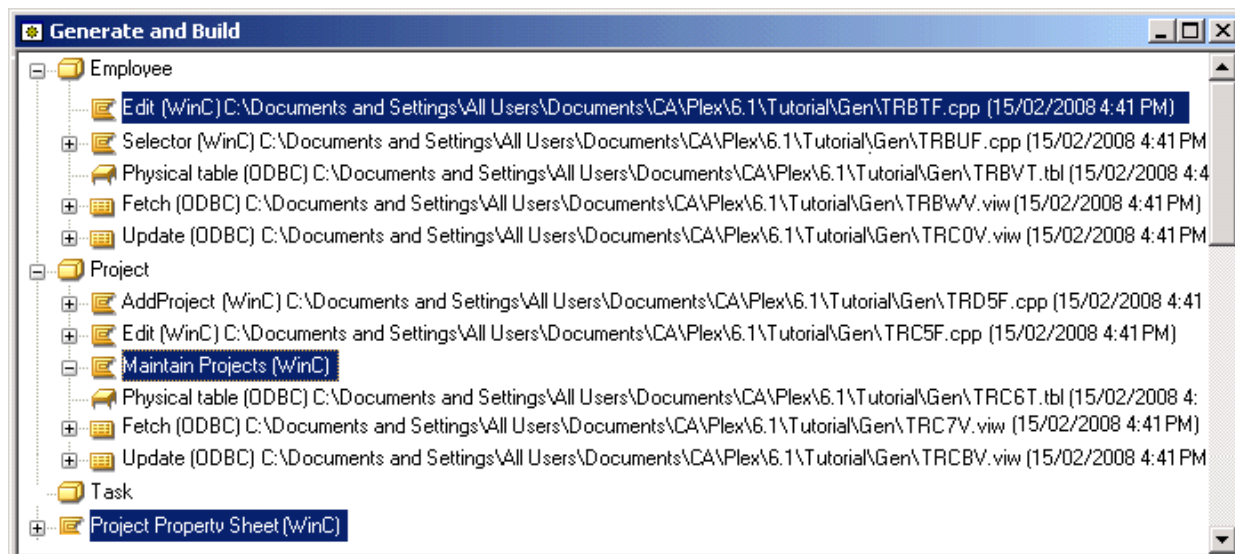


4. Select the `&(1:)` text, and replace it with:
Maintain Employees, Projects, and Tasks.
5. Save your model and close the Message Editor.

Generating and Building

You have defined the Project Property Sheet and Project.Maintain Projects functions, and have modified the Employee.Edit function.

Generate and build these three functions:



For more information on generating and building, see the section Generating and Building in the chapter "Your First CA Plex Application in 20 Minutes."

Testing Your Property Sheet

You now have a fully functional property sheet. You:

- Added restrictor processing so that when you select a project from the Projects and Tasks tab, only the tasks that belong to the selected project appear
- Changed the tab text and the dialog caption

To test your property sheet:

1. In the Generate and Build window, select the Project Property Sheet function and click the Run button.



2. Select Plex r7.0 Tutorial.dsn from the Select Data Source dialog.

The Maintain Employees, Projects, and Tasks property sheet appears.

There are two Apply buttons:

- One on the Employee tab—Detail region applies a new or changed employee record
- One at the bottom of the property sheet—Processes all of the action diagram code that clicking the OK button processes, but does not close the property sheet

The screenshot shows a Windows-style dialog box titled "Maintain Employees, Projects, and Tasks." with a close button (X) in the top right corner. It has two tabs: "Employees" (selected) and "Projects and Tasks".

On the "Employees" tab, there is a list box on the left containing three entries:

Employee ID	Employee Name
Emp01	Heather Eisenman
Emp02	David Castro
Emp03	Martha Kolman

The "Emp03 Martha Kolman" entry is selected and highlighted. Below the list box is a horizontal scrollbar.

To the right of the list box is a detail region with four labeled text boxes:

- Employee ID: Emp03
- Employee Name: Martha Kolman
- Employee Hire Status: Part-time (dropdown menu)
- Employee Email Address: kolman@No.Where

Below these text boxes is a checkbox labeled "Continue new?" which is currently unchecked. At the bottom of the detail region are four buttons: "Apply", "New", "Delete", and "Refresh".

At the very bottom of the dialog box are three buttons: "OK", "Cancel", and "Apply".

1. Select the Projects and Tasks tab:

The screenshot shows a software interface titled "Maintain Employees, Projects, and Tasks." with two tabs: "Employees" and "Projects and Tasks". The "Projects and Tasks" tab is active. It features a table with two columns: "Project ID" and "Project Description". Below this table is another table with two columns: "Task ID" and "Task Description". To the right of the task table are four input fields labeled "Project ID", "Task ID", "Task Description", and "Employee ID". At the bottom right, there are buttons for "Apply", "New", "Delete", and "Refresh", along with a "Continue new?" checkbox. At the very bottom are "OK", "Cancel", and "Apply" buttons.

Project ID	Project Description
Proj01	Create a database application for the McCready account. Conta
Proj02	Create an email client for internal support staff.
Proj03	Convert the ODBC version of the McCready application to AS/4
Proj04	Port client to Java, to enable staff to view email from browser.

Task ID	Task Description
---------	------------------

Project ID: Proj01
Task ID:
Task Description:
Employee ID:
Continue new? ☐
Apply New Delete Refresh
OK Cancel Apply

2. Select Proj04 from the grid at the top.
3. Notice that the task you created for Proj04 appears in the bottom portion of the property sheet.
4. Select Proj02 from the grid at the top.
5. Note that the Project ID field in the task detail region fills in.
6. Add the following task information:

Value	Enter
Task ID	Task01
Task Description	Design the server functionality
Employee ID	Emp03

7. Click the Apply button in the Detail region.
8. Click OK to close the dialog.

Chapter Review

In this chapter you:

- Created a property sheet using the FrameProperty and FrameChild patterns
- Changed the size of the panels of all of the child functions to fit on the parent function's panel
- Defined a new function by combining two pattern functions to create a region with a grid listing projects and a region with both a grid listing tasks and an editing region
- Replaced inherited template views with actual views in your model
- Modified inherited action diagrams so that they only list the tasks that belong to the selected project
- Specified the text to display on each tab of the property sheet
- Generated and built your property sheet and tested it

This chapter introduced the following patterns:

Pattern	Description
 UISTYLE/FrameProperty	The parent function for a property sheet.
 UIBASIC/Grid2	Adds a second grid region to a panel.

This chapter introduced the following triple:

Triple	Description
FNC name NME	Specifies a name to use for a function on panels; in this case, as the text for a property sheet tab.

Chapter 6: Putting It All Together

In the chapter "Creating a Property Sheet," you created a property sheet with two tabs, one for editing employee data, and one for maintaining the tasks associated with a project. It also taught you how to replace abstract views from inherited patterns with actual views for database access.

The property sheet was created by defining a new function with combined patterns from the UIBASIC, UISTYLE, and FOUNDATION Pattern Libraries. Creating a property sheet is much like creating a wizard.

In this chapter, you will define a top-level interface from which you can create a new project, using the wizard, and maintain employees and projects using the property sheet.

This section contains the following topics:

[Creating a Function to Tie the Application Together](#) (see page 123)

[Modifying the Project Manager's Panel](#) (see page 124)

[Creating and Mapping Logical Events](#) (see page 126)

[Event Handling in the Action Diagram](#) (see page 128)

[Generating and Building](#) (see page 134)

[Testing the Application](#) (see page 136)

[Chapter Review](#) (see page 137)

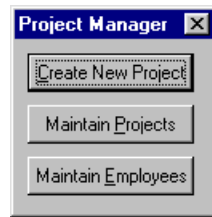
Creating a Function to Tie the Application Together

There are three things that you can do with the application:

- Add new projects, using the wizard
- Maintain data about projects and tasks, using the second tab of the property sheet
- Maintain employee data, using the first tab of the property sheet

In this chapter, you create a simple function that calls the wizard and property sheet. You start with UIBASIC/UIBasicShell, which has a panel and a caption, but no associated processing.

Next, you modify the panel for your function by adding three buttons to it, which let users open the property sheet and the wizard:



You also specify a recognizable file name for the application, so that when you create an executable file (.EXE), its name makes sense.

To create a top-level interface:

1. Set the Model Editor object type to Function, and add the following triple:
Project Manager **is a** UIBASIC/UIBasicShell

UIBASIC/UIBasicShell is one of the simplest functions in the CA Plex Pattern Libraries. It has a panel with a caption, and a single event to close the panel. The action diagram has an Events Handler, where you specify how to respond to logical events you define on the panel, but not much processing beyond that.

When you deploy your application, you want your executable file to have a meaningful name. The Project Manager function becomes the executable program. If you do not specify a file name for that function, CA Plex creates an executable with an automatically generated file name, such as AAc4F.exe.

2. Add the following triples:
Project Manager **file name** Projects
Project Manager **impl name** Projects

Now, when you build your application, CA Plex gives it the name Projects.exe.

Modifying the Project Manager's Panel

In this section, you modify the Project Manager panel by adding three buttons:

- The first button enables end users to add a project by starting your wizard.
- The second button enables end users to maintain project data by opening your property sheet with the Projects and Tasks tab showing.
- The third button enables end users to maintain employee data by opening your property sheet with the Employees tab showing.

To modify the Project Manager's panel:

1. Open Project Manager.Panel. Notice that, unlike the other panels you have worked with, this one does not come with any regions, buttons, or grids.



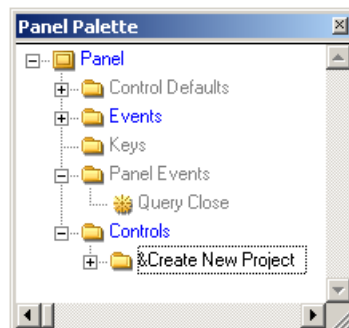
2. Right-click the top left corner of the panel and choose Create Push Button.

A button is added to your panel, with the button text selected.

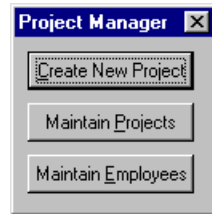
3. Enter **&Create New Project** to change the text, and press Enter.

The & in the text indicates that the letter that follows appears as an underlined access key on the screen.

The new button you created shows up in the Panel Palette under the Controls folder:



4. On the panel, click below the Create New Project button to clear it.
5. Repeat Steps 2-4 to create two more buttons below the Create New Project button, with the captions Maintain &Projects and Maintain &Employees.
6. Resize and arrange the panel and its buttons so that it looks like this:



7. Save your changes, but keep the Panel Designer open.

Creating and Mapping Logical Events

At this point, the buttons you just created will not do anything if you click them. You need to define a logical event for each button so that you can define what should happen when that button is clicked.

Use logical events to link actions end users take to code in an action diagram. In the following steps, you create logical events that are triggered when your end users click each of the three buttons.



After you define these events, you modify the Project Manager function to call the wizard and property sheet functions when each logical event is triggered.

To create logical events on a panel:

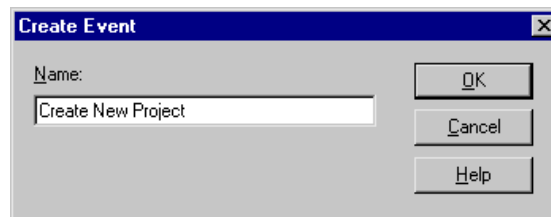
1. In the Design Window, select the Create New Project button.
2. Right-click the button and choose Event Mappings from the pop-up menu.

The Event Mappings dialog appears.

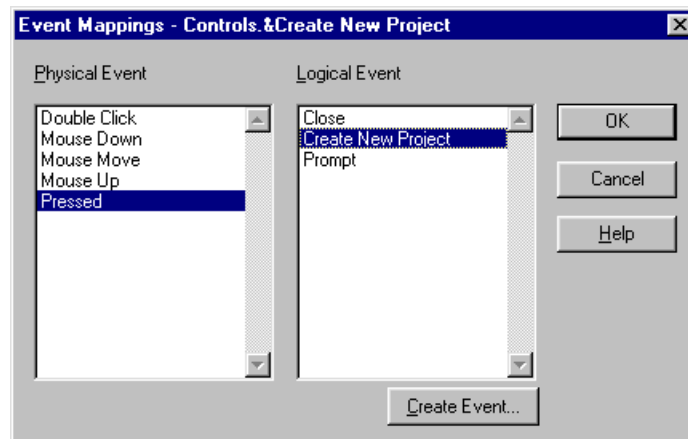
Note that there are already two logical events defined: Close and Prompt. Next, you add your own logical event.

3. Click Create Event.

The Create Event dialog appears:



4. Enter **Create New Project** and click OK.
5. Your new logical event is now available in the Logical Event list. Select Pressed in the Physical Event list, and Create New Project in the Logical Event list:



Now, when end users click the Create New Project button, the Create New Project logical event is triggered. Later, you add action diagram code to call the Project Wizard function when this event is triggered.

6. Click OK.

7. Repeat Steps 2-6 to create logical events called Maintain Projects and Maintain Employees. Associate the logical events with the Pressed physical events for the corresponding buttons.
8. Save your changes but keep the Panel Designer open.

Event Handling in the Action Diagram

You have now defined logical events and linked them to physical events for the buttons you created. Next, you must tell the application what to do when those logical events are triggered. You do so within the Events Handler construct of the action diagram for your Project Manager function.

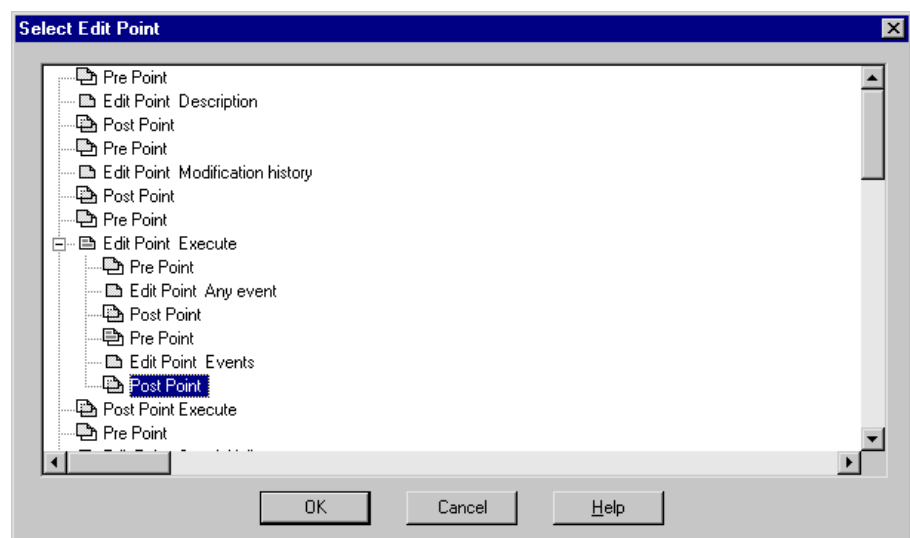
Calling the Wizard

When end users click the Create Project button, you want to call the Project Wizard function so that they can define a new Project record and the Task records that go with that Project.

To do this, add an Event construct specifying what should happen when the Create New Project event is triggered.

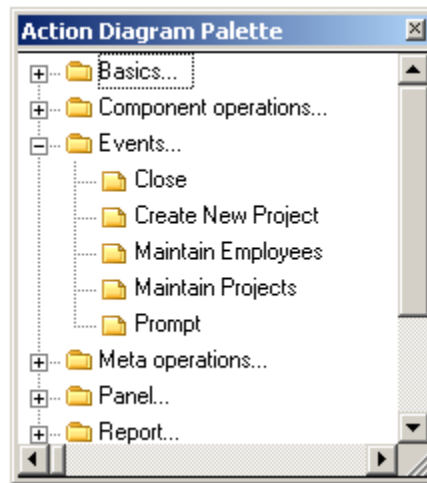
To add action diagram code for the Create New Project event:

1. Choose Edit Scoping Function F11 from the View menu to switch to the action diagram for the Project Manager function.
2. In the Select Edit Point dialog, select the Events Post Point, and click OK:



3. In the Action Diagram Palette (not the Variable Palette), expand the Events folder.

Note that the three logical events you created are listed in the Events folder:



4. Drag the Create New Project event to the input line of the Action Diagrammer, and press Enter.

This adds an Event construct:



5. Set the Object Browser to display functions.
6. Drag the Project Wizard function to the input line of the Action Diagrammer, and press Enter.

A function call statement is added to the action diagram:



This tells the function to display the wizard when the Create New Project button is pressed.

At this point, you will only map the one event. Before you can map the other two events to call the property sheet, you must make a small change to the property sheet function.

7. Save your model and keep the action diagram open for later.

Modifying the Property Sheet

When end users click Maintain Projects on your top-level panel, the property sheet should open to the Projects and Tasks tab. When they click Maintain Employees, the property sheet should open to the Employees tab. Before you can do this, you have to modify the Project Property Sheet function to accept a starting tab number as an input parameter.

To enable the property sheet to open to a specific tab:

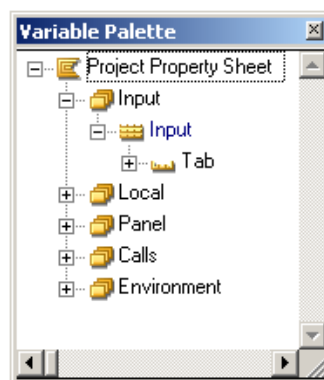
1. Open the action diagram for Project Property Sheet.
2. Click Cancel on the Select Edit Point dialog.

In the following steps, you add the FIELDS/Tab field to the Input variable in the Input variable group. The value of this Tab field specifies which tab of the property sheet to display.

3. Set the Object Browser to display fields.
4. Make sure that library objects are displayed. If they are not, click the Show/Hide Library Objects button.



5. In the Variable Palette, expand the Input variable group.
6. Drag the FIELDS/Tab field from the Object Browser to the Input variable in the Variable Palette:



Now, any function that calls the Project Property Sheet function has to pass in a value for this field. Next, you add a statement to use that value to determine which tab to display.

7. Navigate to the Pre Point before the End Initialize Edit Point.

There is already code in the Pre Point. The statement you add goes at the end, after what is already there.

8. Add the following statement:

Set TabStripL<SelectedTab> = Input<Tab>

This sets the displayed tab to the one indicated in the input variable. The action diagram should look like this:

```

Pre Point End initialize
Go Sub Set panel caption
Call Meta.Options
Pre Point
Edit Point Process options
Post Point
+For Defined Value Function: Project Property Sheet [ActiveShell]
Go Sub ImageList load images
+++Define System: Meta
+++Define Function: Project Property Sheet [FrameParent]
+++Define Name: UISTYLE/LoadOnDemand
+For Defined Value Function: Project Property Sheet [FrameParent]
++Name Defined Function: Project Property Sheet [FrameParent], GlobalPropertyL<GlobalProperty>
Go Sub Set this parents index
Go Sub Set initial shared data values
Go Sub Load child site
Go Sub TabStrip initialize
Go Sub Initialize tabs
Go Sub Set panel caption
Set TabStripL<SelectedTab> = Input<Tab>

```

9. Save your model and close the action diagram.

Calling the Property Sheet

You have now changed the property sheet so that when you call it, you can specify which tab to display. Next, you add the calls from the Project Manager function to the property sheet.

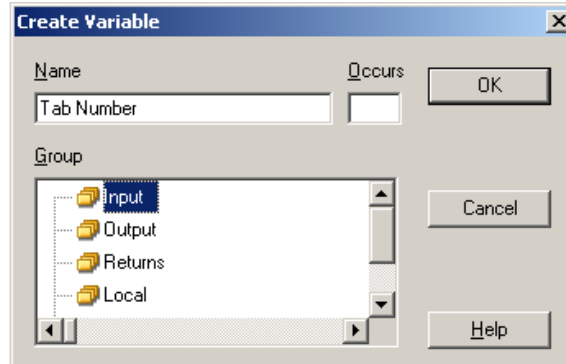
To call the property sheet:

1. Switch back to the action diagram for the Project Manager function and go to the Events Post Point. If necessary, you can use the menu items on the Window menu to switch between editors.

- Click below the Events construct you already added.

Before you can add the next event construct, you need to create a local variable. You then add a field to the variable that you use when calling the property sheet.

- Right-click one of the variable groups in the Variable Palette, and choose Create Variable from the pop-up menu.
- Select the Local variable group from the Group list.
- Enter **Tab Number** in the Name field, and click OK.

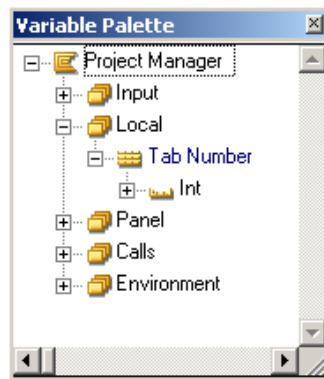


This creates a variable called Tab Number in the Local variable group. Next, you add a field to the variable.

- In the Variable Palette, expand the Local variable group.

CA Plex provides a field called Int in the FIELDS Pattern Library that stores literal values for integers. Use the field in this function to specify the tabs in the property sheet by their number (Tab 1 and Tab 2).

- Set the Object Browser to display fields, make sure library objects are showing, and drag the FIELDS/Int field from the Object Browser to the Tab Number variable:



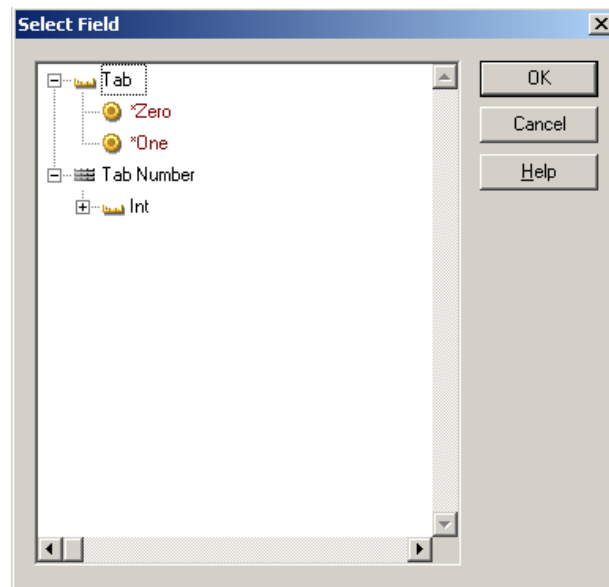
Next, tell the Project Manager function what to do when the Maintain Projects and Maintain Employees buttons are clicked.

8. In the Action Diagram Palette, expand the Events folder.
9. Drag the Maintain Projects logical event from the Action Diagram Palette to the input line of the Action Diagrammer, and press Enter.
10. Drag the Project Property Sheet function from the Object Browser to the input line of the Action Diagrammer, and press Enter. Notice that CA Plex automatically assumes you are entering a Call statement.

The Parameter Mapping dialog appears. Next, tell the Project Manager function what value to pass to the Project Property Sheet function. The Project Property Sheet function uses that value to determine which tab to display.

11. Click Map Parameter.

The Select Field dialog appears:



Notice that both the Tab Number variable and the Int field you added to it are displayed.

12. Expand the Int field.
13. Select the value 2 and click OK.

This tells the property sheet to open the Projects and Tasks tab when it is called. Next, you tell the Project Manager function to open the property sheet when the Maintain Employees button is clicked. But this time, you make the Employees tab show.

14. Click OK to close the Parameter Mapping dialog.
15. Click below the Maintain Projects Event construct in the action diagram.
16. Clear the input line of the Action Diagrammer.
17. Drag the Maintain Employees logical event from the Action Diagram Palette to the input line of the Action Diagrammer, and press Enter.

This adds another Event construct to the action diagram.

18. Repeat Steps 8-14 to call the Project Property Sheet function for the Maintain Employees logical event, but instead of mapping to <Int.2>, map it to <Int.*One>.

The action diagram should now look like this.



19. Save your model and close the action diagram.

Generating and Building

You have now created a logical event for each of the three buttons you created. You have mapped those logical events to physical events, and, you specified what should happen whenever those logical events are triggered. You also changed the Project Property Sheet function to give it an input parameter, which determines which tab to display. Next, you generate and build the Project Manager and Project Property Sheet functions and test them to verify that they work.

For full instructions on generating and building, see Generating and Building in the chapter "Your First CA Plex Application in 20 Minutes."

Creating an Executable Program

So far, you have always tested the parts of the application you are building from within the CA Plex development environment. To deploy the application after you have finished it, you need to have an executable file.

To create an executable program:

1. In the Generate and Build window, select the Project Manager function.
2. Click the Create Exe button.



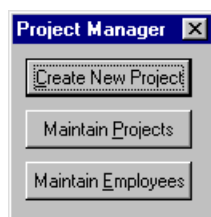
Remember that earlier in this chapter, you added the triple Project Manager **file name** Projects. The Create Exe process uses that triple to determine the filename to create.

3. When prompted to build the .exe file, click Yes.
4. Switch from CA Plex to Windows Explorer, and locate the directory in which your compiled files reside.

Note: By default this directory is C:\Program Files\CA\CA Plex r6.0\Tutorial\Gen\Release.

5. Double-click the file Projects.exe to start the application.

The Project Manager dialog appears:



6. Close the application.

Stopping the Select Data Source Dialog Appearing

Throughout this tutorial you may have noticed that every time a function runs you are prompted to select an ODBC data source (Plex r7.0 Tutorial.dsn). This can be prevented by entering the required DSN (Data Source Name) in the application's initialization (.INI) file. When you created an executable program you also created a corresponding file called Projects.ini.

1. In the Generate and Build window, select the Project Manager function.
2. From the View menu, choose View Source.
3. In the Select Source To View dialog, select the Projects.ini file (the last file in the list) and click OK.
4. Scroll down to the [ODBC Settings] section and update the *DSN=* setting as shown:

```
[ODBC Settings]
SAVE DSN=False
DSN=Plex r7.0 Tutorial.dsn
Userid=
Password=
Database=
Errorlog=
Default Connect=True
```

5. Close the document and save your changes.

Testing the Application

Now that you have generated and built the application, created an executable file for the Maintain Project function, and started that executable file, you can test the application.

To test the application:

1. Double-click Projects.exe to start the application.
2. Click Maintain Employees.

The property sheet appears with the Employees tab visible.

3. Add the following employee:

Value	Enter
Employee ID	Emp04
Employee Name	Bob Strudwick
Employee Hire Status	Full-time
Employee Email	bstrudwick@anonymcorp.com

4. Close the dialog.
5. Click Create New Project.
The project wizard appears.
6. Add the following project:

Value	Enter
Project ID	Proj05
Project Description	Create a data repository for legacy financial information.
Project Start Date	06/15/04
Project End Date	12/01/04

7. Add the following tasks for the project:

Value	Enter
Task ID	DBImport
Task Description	Use db imp
Employee ID	Emp04
Value	Enter
Task ID	Transport
Task Description	Transport
Employee ID	Emp02

8. Click Finish to close the wizard.
9. Click Maintain Projects.
- The property sheet appears with the Projects and Tasks tab visible.
10. Verify that the information you added with the wizard is in the database.
11. Close the property sheet.
12. Close the application.

At this point, you can no longer test the property sheet by running it directly from the Generate and Build window. You can only run the property sheet from the Project Manager function, because you specified an input parameter to tell the property sheet which tab to display.

Chapter Review

In this chapter, you:

- Defined a new function, inheriting from `UIBasicShell`, to serve as an entry point for your application
- Created push buttons and linked them to logical events on the panel of your new function
- Defined three Event constructs in the action diagram to enable the push buttons, so that:

- Clicking Create New Project calls the wizard function you defined in the chapter "Creating a Wizard"
- Clicking Maintain Projects calls the property sheet function you defined in the chapter "Creating a Property Sheet" with the Projects tab selected
- Clicking Maintain Employees calls the property sheet function with the Employees tab selected
- Generated and built the changed functions and tested your application

This chapter introduced the following pattern:

Pattern	Description
 UISTYLE/UIShell	A prototype user interface function with no panel regions and a Close event.

This chapter introduced the following triples:

Triple	Description
FNC file name NME	Specifies a name to assign to the file generated for the source function.
FNC input FLD ...for VAR	Defines an input parameter for a function.

Appendix A: The Tutorial Reference Model

In this tutorial, you created a model for an application that tracks:

- Projects
- Tasks
- Employees

You started with an empty model with no local objects in it. In addition to that empty model, CA Plex comes with a completed version of the tutorial model, known as the tutorial reference model.

If you would like to see what the completed model looks like, you can review the tutorial reference model at any time. This model contains all of the information that you add as you go through the tutorial, plus comments in the functions' action diagrams.

This section contains the following topics:

[Using the Tutorial Reference Model](#) (see page 139)

[What Does It Contain?](#) (see page 140)

[What Does It Look Like?](#) (see page 143)

[Generating and Building](#) (see page 147)

Using the Tutorial Reference Model

The tutorial reference model, tutrefer.mdl, is located in the Plex tutrefer subdirectory. The file started as the tutorial.mdl file, and has all of the information in it that you add when doing the tutorial.

The model uses its own data source, and that data source has a corresponding database. So, you can work with the tutorial reference model without conflicting with the tutorial model, or its data.

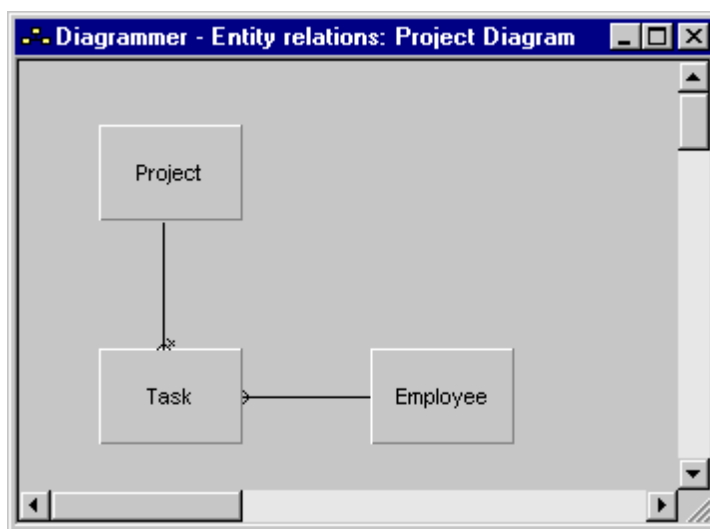
Because it is separate from the tutorial model, you can also use the reference tutorial model to experiment with. As you read about functionality in the online help, you can use the tutorial reference model to try out CA Plex features. Before experimenting with this model, make a backup copy of it, so that you can revert to the shipped version.

What Does It Contain?

The tutorial reference model represents the model that you create when you do the tutorial. It has three entities: Project, Task, and Employee.

The three entities have relationships with each other. The Project entity owns the Task entity; when you delete a project, all of its tasks are deleted as well. The Task entity refers to Employee. This enables you to assign employees to tasks, but when tasks are deleted, the employees are not deleted.

The following shows the entity-relations diagram you created in the tutorial:



Here is a description of the three entities:

Entity	Description	Inherits From
Project	Stores information about projects in the Project ID, Project Description, Project Start Date, and Project End Date fields You define this entity in Defining the Project Entity in the chapter "Your First CA Plex Application in 20 Minutes."	FOUNDATION/EditDetail STORAGE/RelationalTable

Task	Stores information about tasks for projects This entity is owned by the Project entity, so it stores the owning project's key field, and data in the Task ID and Task Description fields. It refers to Employee, so it also stores the Employee's key field. You define this entity in Defining the Task Entity in the chapter "Modeling Entity Relationships."	FOUNDATION/EditDetail STORAGE/RelationalTable FOUNDATION/OwnedCascade
Employee	Stores information about employees in the Employee ID, Employee Name, Employee Hire Status, and Employee Email Address fields You define this entity in Defining the Task Entity in the chapter "Modeling Entity Relationships."	FOUNDATION/EditDetail FOUNDATION/ReferredTo STORAGE/RelationalTable

The three entities store information in ten fields:

Field	Description	Inherits From
Project ID	The key field for projects.	FIELDS/Identifier
Project Description	Stores free-text information about a project This field is set as optional.	FIELDS/LongDescription
Project Start Date	Stores the start date and end date for a project. These fields are both set as optional.	DATE/CheckedDateISO
Project End Date		
Task ID	The key field for tasks.	FIELDS/Identifier
Task Description	Stores free-text information about a task.	FIELDS/ShortDescription
Employee ID	The key field for employees.	FIELDS/Identifier
Employee Name	Stores the name of the employee.	FIELDS/ShortDescription
Employee Hire Status	Stores the hire status of the employee: full-time, part-time, or contract.	FIELDS/Status
Employee Email Address	Stores the email address of the employee.	FIELDS/ShortDescription

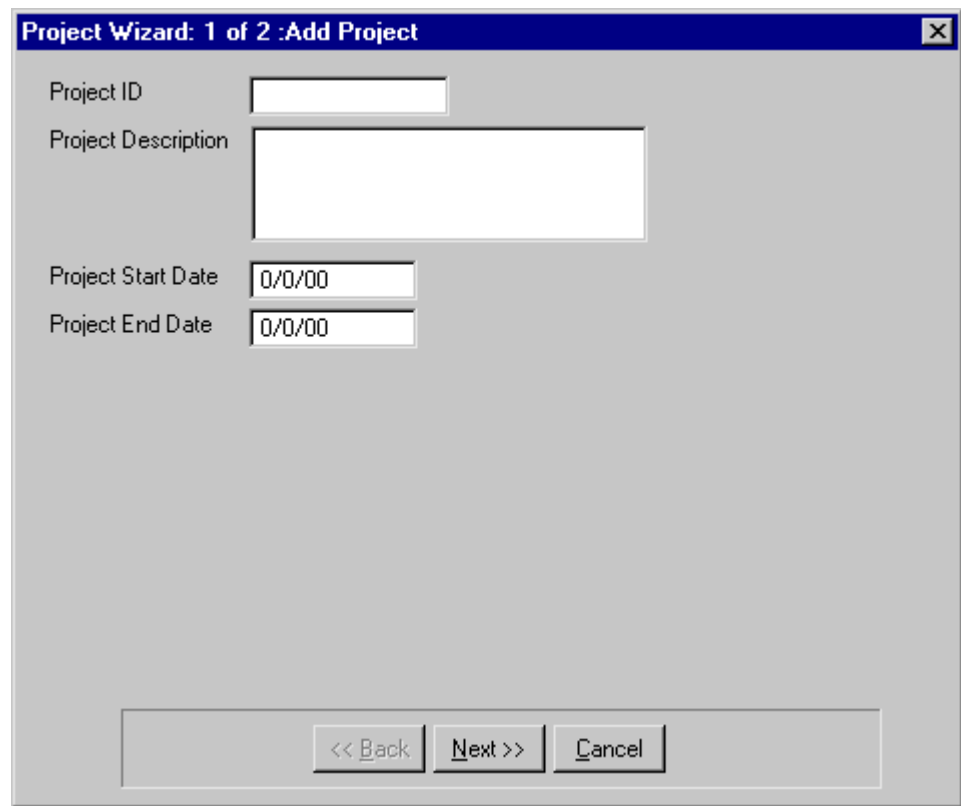
The model has three unscoped functions:

Function	Description	Inherits From
Project Wizard	Enables end users to create projects and tasks for those projects.	UISTYLE/FrameWizard
Project Property Sheet	Enables end users to maintain projects and tasks entered with the Project Wizard, to add new tasks, and to add and update employees.	UISTYLE/FrameProperty
Project Manager	Displays three buttons to open the Project Wizard and the Project Property Sheet.	UIBASIC/UIBasicShell

What Does It Look Like?

End users add projects and tasks using a wizard, and maintain them using a property sheet. They can add new tasks in the property sheet, but can only add new projects using the wizard. They add and maintain employees using the property sheet.

Wizards were defined in the chapter “Creating a Wizard.”



The screenshot shows a Windows-style dialog box titled "Project Wizard: 1 of 2 :Add Project". The dialog has a blue title bar with a close button (X) in the top right corner. The main area is light gray and contains four input fields with labels to their left: "Project ID" (a single-line text box), "Project Description" (a multi-line text box), "Project Start Date" (a date box showing "0/0/00"), and "Project End Date" (a date box showing "0/0/00"). At the bottom of the dialog, there is a horizontal bar containing three buttons: "<< Back", "Next >>", and "Cancel".

Property sheets were defined in the chapter “Creating a Property Sheet.”

Maintain Employees, Projects and Tasks

Employee Projects and Tasks

Project Id	Project Description
Proj01	Create a database application for the MCreedy account. Conta
Proj02	Create an email client for Internal support staff
Proj03	Convert the ODBC version of the McCreedy application to Syst
Proj04	Port the support staff email client to Java to enable staff to view

Task Id	Task Description
---------	------------------

Project Id: Proj01

Task Id:

Task Description:

Employee ID:

Continue new? ☐

Apply New Delete Refresh

OK Cancel Apply

End users add new tasks and maintain existing tasks in the property sheet (already shown), and add tasks in the second part of the wizard:

Project Wizard: 2 of 2 :Edit

Project Id	Task Id
------------	---------

Project Id: Proj1

Task Id:

Task Description:

Employee ID:

Continue new? ☐

Apply New Delete Refresh

<< Back Finish Cancel

End users add and maintain employees with the property sheet:

The dialog box is titled "Maintain Employees, Projects and Tasks". It has two tabs: "Employees" and "Projects and Tasks". The "Employees" tab is selected. Below the tabs are two icons: a green plus sign and a yellow minus sign. To the right of the icons is a table with two columns: "Employee ID" and "Employee Name". The table contains three rows: "Emp01" with "Heather Eisenman", "Emp02" with "David Castro", and "Emp03" with "Martha Kolman". The first row is selected. To the right of the table is a property sheet with four fields: "Employee ID" (text box with "Emp01"), "Employee Name" (text box with "Heather Eisenman"), "Employee Hire Status" (dropdown menu with "Contract" selected), and "Employee Email Address" (text box with "heisenman@anonymcorp.com"). Below the property sheet is a checkbox labeled "Continue new?". At the bottom of the dialog box are four buttons: "Apply", "New", "Delete", and "Refresh". At the very bottom are three buttons: "OK", "Cancel", and "Apply".

Employee ID	Employee Name
Emp01	Heather Eisenman
Emp02	David Castro
Emp03	Martha Kolman

Employee ID: Emp01
Employee Name: Heather Eisenman
Employee Hire Status: Contract
Employee Email Address: heisenman@anonymcorp.com

Continue new? ☐

Apply New Delete Refresh

OK Cancel Apply

Generating and Building

You can generate and build the tutorial reference model immediately, without having to do anything to it.

To generate, build, and run the tutorial reference model:

1. Open the tutrefer.mdl model.



2. Open a Generate and Build window.
3. Select all of the non library objects in the window.



4. Click the Generate and Build toolbar button.

CA Plex expands the selected entities and highlights all of the generatable objects under it.

5. A Confirm Generate dialog appears, indicating the number of objects that are generated. Not all of the scoped objects are selected. Click Yes.

CA Plex generates those objects, and then summarizes the generation process.

6. CA Plex prompts you to compile and build the objects. Click Yes.

CA Plex starts Microsoft Visual Studio to compile the generated source files.

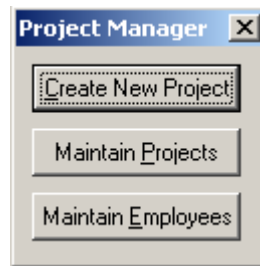
Note: You will see several warnings in the Message Log during the compilation process that indicate that no source for the various views (views are the files that end with the extension VIW) were submitted to the ODBC generator. Do not worry about the warning; your application will function properly.

You can tell that your build is done when the label on the Microsoft Visual Studio task bar button changes from the name of the model to Microsoft Visual C++ or, if you have opened the Visual Studio window so that you could watch the build, you can tell that it is finished when the cursor returns to the top of the build summary.

7. In the Generate and Build window, right click the Project Manager function.
8. Choose Create Exe from the pop up menu.
9. Visual Studio creates an executable file for the function.
10. Switch to Windows Explorer, find the directory that stores your compiled files, and double click Projects.exe. The default location for the compiled files is:

PlexInstallationDirectory/Tutrefer/Gen/Release

11. The tutorial reference model application starts:



12. Click Maintain Employees to add some employee records.
13. When prompted to select a data source, choose Plex r7.0 Tutrefer.dsn.
14. Click Create New Project to add a project and its tasks.
15. Click Maintain Projects to verify that the information you added is there.

Note: For more information, see Testing the Application in the chapter “Putting It All Together.”

Index

B

building. See generating and building • 36

C

child sites • 68
compiling. See generating and building • 36
controls • 45
customer support, contacting • 3

D

data • 23, 41
 preserving • 41
 validation • 23
data types • 23, 61
 Project's fields • 23
 Task's fields • 61
defining • 123
 simple panels • 123
deleting scoped objects • 35
diagram • 15, 55
 links and nodes • 55
 of completed tutorial model (Fig) • 15
diagrams • 17

E

ENT • 20, 33, 68
 function FNC • 68
 has FLD • 20
 is a ENT • 33
entity • 17

F

fields • 17, 29, 78
 adding to variables • 78
 definition • 17
 making optional • 29
FIELDS/ • 23, 61, 130, 131
 Int • 131
 LongDescription • 23, 61
 Tab • 130
FOUNDATION/ • 33, 61, 101
 EditDetail • 33
 EditDetail.Edit • 101

OwnedCascade • 61
functions • 17, 68, 74, 130
 adding input variables • 130
 definition • 17, 74
 unscoped • 68

G

generating and building • 36, 63, 70, 93, 117, 134
 Add Project function • 70
 Project Manager • 134
 Project Property Sheet • 117
 Project Wizard • 70, 93
 Task entity • 70
 the Employee entity • 63
 the Project entity • 36

L

library objects, hiding/showing • 18
links • 55
Local Model • 16
LongDescription field. See FIELDS/LongDescription • 23

M

messages • 17
Model Editor • 19, 23
 dialog • 19
 displaying more detail • 23
 focusing on objects from the Object Browser • 23
model, saving • 38

N

nodes • 55
non-key attribute • 20

O

Object Browser • 16, 18, 23, 29
 definition • 16
 dragging objects to the Model Editor • 23
 hiding/showing library objects • 18
 opening • 18
 setting to display entities • 29
objects • 17, 18, 23, 55
 focusing on • 23

- hiding/showing • 18
- represented in diagrams • 55
- types • 17
- ODBC (open database components) • 9
- open database components (ODBC) • 9
- optionality • 29

P

- Panel Designer • 43, 45, 46
 - the Design Window • 43
 - the Panel Palette • 45
 - the Property Sheet • 46
- panels • 17
- panels, defining simple • 123, 124
- pattens • 13
- preserving data • 41
- Project entity • 19, 31, 33, 36
 - defining • 19, 31
 - generating and building • 36
 - specifying database functionality • 33

R

- RelationalTable entity. See STORAGE/RelationalTable
 - 33
- requirements • 9

S

- saving the tutorial model • 16
- scoped objects • 35, 68
 - defining • 68
 - deleting • 35
- STORAGE/, RelationalTable • 33

T

- tables • 17, 41
 - definition • 17
 - saving data in • 41
- TBL, implement SYS • 41, 64
- triples • 19, 29, 112
 - adding • 19
 - continuation • 29
 - deleting • 19
 - editing • 19
 - parts • 19
 - reordering • 112
 - significance of order • 112
- tutorial model, explanation • 15

U

- unscoped functions • 68
- unscoped functions, displaying • 18

V

- views • 17, 61
 - adding SuperKeys to Task • 61
 - definition • 17