

CA PLEX

Tutorial for System i 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

Accessing Online Help	9
Helpful Hints	10
Getting Started	10
Other Tutorials	10
What Is CA Plex?	10
The Software Development Process	11
Concepts Preview	12
Project Management Application	16
A First Look	18
Opening CA Plex	18
Closing CA Plex	19
Setting Up Generate and Build Options	19
Specifying Build System Settings	19

Chapter 2: Defining Entities 27

Introducing the Object Browser	27
Opening the Object Browser	27
Showing Library Objects	29
Introducing the Model Editor	30
Defining the Project Entity	30
Opening the Model Editor	30
Defining the Project Entity	31
Specifying Attributes for the Project Entity	31
Defining Field Properties	33
Using Inheritance to Define the Project Entity	36
Generating and Building the Project Entity	38
Using Your Generated Application	42
Adding a New Project	42
Deleting a Project	44
Changing a Project	44
Preserving Data	44
Chapter Review	45
Helpful Hints	45
Misnamed Objects	45
Triples	48

Chapter 3: Modifying the User Interface **49**

Introducing the Panel Designer	49
Modifying the User Interface	50
Design Window	50
Property Sheet	53
Changing Individual Controls.....	55
Changing More Than One Control.....	56
Adding Static Text.....	57
Generating and Building.....	58
Chapter Review	59

Chapter 4: Diagramming Entity Relationships **61**

Introducing the Diagrammer.....	61
Choosing Between Editors	63
Creating a Diagram.....	63
Defining Entities with the Diagrammer	66
Defining Attributes with the Diagrammer.....	67
Defining Relationships Between Entities	68
Defining an Owned By Relationship.....	69
Defining a Refers To Relationship	70
Defining Employee and Its Attributes	71
Defining the Employee Attributes.....	71
Defining the Employee Entity.....	73
Refers to Relationships.....	74
Adding Prompt Processing	74
Modifying Panels.....	75
Hiding Fields on Panels	75
Generating and Building the Employee Entity	77
Testing the Application	78
Adding Employees.....	79
Deleting Employees.....	80
Changing an Employee.....	81
Chapter Review	81

Chapter 5: Defining Owned By Relationships **83**

Defining Task and Its Attributes	83
Defining the Attributes for the Task Entity	83
Defining the Task Entity	84
Owned By Relationships.....	85
Defining Restrictor Processing	86

Creating a Key Code on a Panel	87
Introducing the Action Diagrammer	89
Action Diagrammer Windows	89
Action Diagram Windows.....	90
Action Diagram Palette	90
Variable Palette	91
Basic Concepts	91
Adding Functionality to Logical Events	92
Generating and Building the Task Entity and Project.Grid Maintenance Suite.Edit Grid User Interface	97
Testing the Application	97
Chapter Review	100
Helpful Hints.....	101
Owned By Relationships and OBASE/Child	101
 Chapter 6: Putting It All Together	 103
Creating a Top-Level Menu	103
Generating and Building Your Applications.....	107
Testing the Application	107
Chapter Review	108
 Index	 109

Chapter 1: Introduction

This guide is written specifically for analysts, programmers, managers, and system administrators. It provides systematic instructions for using CA Plex in a tutorial format. It requires a familiarity with data modeling concepts and a working knowledge of the operating environments of CA Plex and the applications you are developing. This tutorial guides you through the process of creating a simple model from the design stages through adding data to your finished application.

This chapter provides introductory information about this guide and explains some key concepts that you need to understand before you start using CA Plex. It also describes the application that you will build. For additional information about CA Plex, see *Getting Started* and the *Tutorial for Windows*.

This section contains the following topics:

[Accessing Online Help](#) (see page 9)

[Helpful Hints](#) (see page 10)

[Getting Started](#) (see page 10)

[Project Management Application](#) (see page 16)

[A First Look](#) (see page 18)

[Setting Up Generate and Build Options](#) (see page 19)

Accessing Online Help

The CA Plex online help system contains detailed information about the concepts and functions in CA Plex and systematic procedures for getting started and performing various CA Plex tasks. It also includes descriptions of the CA Plex dialogs, menus, and toolbars.

You can use any of the following methods to access online help:

- From a menu command, select the command on the menu. A short description of the command appears in the status bar.
- From a dialog, click Help or press F1.
- From a class library object, select the object in the Object Browser and press Shift+F1.
- From anywhere within CA Plex, press F1.

Helpful Hints

Be sure to look at the end of the current chapter for the Helpful Hints section. This section gives you additional information or instructions pertaining to the various topics being covered.

Getting Started

This tutorial shows you how CA Plex automates the software development process as you create a sample application. The application you create tracks projects, which consist of various tasks that are assigned to employees. In the process, it introduces a set of editors that form the core of the CA Plex toolset:

- **Object Browser**—Used for viewing the objects in your application
- **Model Editor**—Used for specifying design information that CA Plex uses to generate your application
- **Panel Designer**—Used for designing the user interface elements of your application
- **Diagrammer**—Used for drawing design diagrams
- **Action Diagrammer**—Used for specifying the logical functionality of your application

Other Tutorials

This tutorial is intended for CA Plex users who are developing for System i (AS/400) 5250 environments. However, you also have the option of working through the following guides shipped with CA Plex:

- Tutorial for Windows
- Java Tutorial (see the chapter "Develop the First Application in 20 Minutes" in the *Getting Started*)

What Is CA Plex?

CA Plex is a pattern-based application development tool that works with a variety of generators to enable you to design, generate, compile, and deploy your business applications. CA Plex provides the following basic functionality:

- Model-based application development toolset
- Automatic application code generation
- Class libraries of business objects to accelerate the design and delivery of applications
- Ability to create scalable, business-critical applications
- Pre-built large-scale objects, including reusable application frameworks for standard application functionality, such as date/time processing and security
- Ability to create a model that you can deploy to a variety of platforms, including: System i client/server, System i 5250, ODBC, NT BackOffice, .NET and Java

The Software Development Process

The typical software development process for database applications can be categorized into six phases (not necessarily in this order):

- Data modeling
- User interface design
- Function logic design and implementation
- Generation
- Execution
- Deployment

Data Modeling

Data modeling is the process of determining:

- What data your application uses
- What structure your application uses
- How the various elements of the structure interact

You usually use a diagramming tool for this stage; seeing a visual representation of the objects in your database helps to get a mental picture of your data. Diagrams enable you to specify entities and their attributes, with relationships between entities. For more information on the CA Plex diagramming tool, see the chapter "Diagramming Entity Relationships."

User Interface Design

This stage is usually handled at the same time as the function logic design. In this stage, you create the *panels* that the program uses to enable end users to interact with the data. For more information on CA Plex user interface design tool, see the chapter "Modifying the User Interface."

Function Logic Design and Implementation

In this stage, you determine how the data is manipulated, and how the user interface elements interact. For more information on CA Plex function logic design tool, see the chapter "Defining Owned by Relationships."

Generation

After you finish the data modeling, user interface design, and function logic design, you generate the code from the CA Plex model, and then build the generated code into user-executable program files. For more information on generating and building objects, see the chapter "Defining Entities."

Execution

After successful generation and building, you run the generated application to test its functionality. During this phase, you typically return to some or all of the previous stages to remedy problems and add enhancements.

Deployment



After you finish all of the preceding stages, you deploy your application to your end users. At this stage, you have already created all supporting material, such as help files.







Concepts Preview

The following sections introduce you to concepts and terms that are essential to understanding CA Plex. For more information about any of these topics, search for the term in the online help index.

Objects

The following table presents a list of the different types of objects used in CA Plex models. The list also contains the corresponding abbreviation, icon, and description for each object type.

Object Type & Abbreviation		Icon	Description
Entity	ENT		Entities are <ul style="list-style-type: none">■ Tangible objects, such as customers and orders.■ Intangible objects, such as projects and job titles, about which you want to record information.
Field	FLD		Fields are the pieces of information you want to record about an entity, such as employee names, item prices, and order statuses.

Object Type & Abbreviation		Icon	Description
Function	FNC		Functions are processes and procedures that define the functionality of an application.
Panel	PNL		Panels make up the user interface of your application. You typically inherit panels from libraries, which give you a general layout, and then make changes to them.
Table	TBL		Tables are the physical structures where data is stored.
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 an application. They show their attributes and the relationships between the objects.
Value	VAL		Values hold information, such as the choices in a list box.

Scope

A *scoped object* is an object that belongs to another object. When you refer to a scoped object, you usually use its full name, which includes:

- The name of the object to which it is scoped
- A period
- The object's own name

For example, if you have an entity called Customer that scopes a function called Edit, the Edit function's fully scoped name is Customer.Edit.

A scoped object cannot exist independently of the object to which it is scoped. For example, a panel that is scoped to a function is deleted when the function is deleted.

Some types of objects are unscoped. This means that they exist independently of all other objects in the model. For example, diagrams are unscoped.

Conversely, a table is always scoped to an entity. To create a table object, you must specify the entity to which it belongs.

Entity Relationships

You can specify relationships between entities. The two most common relationships are **owned by** and **refers to** (both are used in this tutorial).

owned by

Creates a parent-child relationship. The child entity cannot exist independently of the parent entity. In the application that you build in this tutorial, the Project entity is the parent of the Task entity. Tasks cannot exist unless they are part of a project. Therefore, if the project is deleted, the tasks should automatically be deleted.

refers to

Creates a link between entities, but both entities can exist independently. In the application you build, you create an Employee entity. The Task entity refers to the Employee entity, by which end users assign employees to tasks. If the task is deleted, the employee still exists.

Triples

Triples are a special kind of construct that CA Plex uses to record design information about an application. As the word implies, a triple has three parts: a source object, a verb, and a target object.

Use *triple* to define the objects in a model. For example, to define that an Employee entity has a field that stores an employee's name, you would create the following triple:

Employee **has** Employee Name

If you wanted Employee Name to be alphanumeric and able to hold 25 characters, you would enter the following triples:

Employee Name **type** Character
Employee Name **length** 25

Important! Notice that the objects and values are not represented with bold text, but the **verb** of the triple is. This is the way triples are formatted in all CA Plex documentation. The verb refers to the relationship you create between the objects, or between the object and the value.

Inheritance

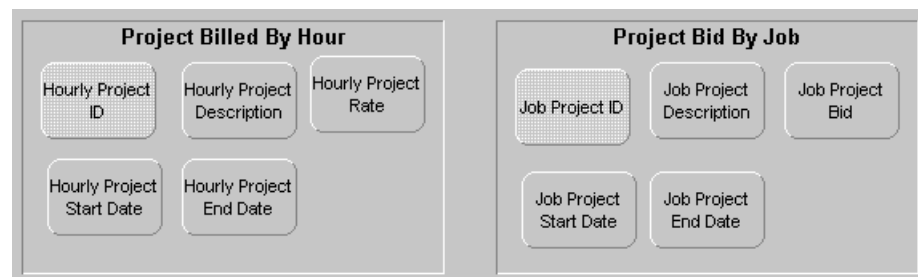
Inheritance enables an object to include the properties of another more generalized object. You specify inheritance for an object using the **is a** verb. For example, FLD **is a** FLD means that the source field inherits all of the properties and scoped objects of the target field.

Another way to assign the data type of character and the length 25 to the Employee Name field is to enter the following triple:

Employee Name is a OBASE/Narrative

OBASE/Narrative is a field that displays 25 characters or settings, which are inherited by Employee Name by way of this triple. So, rather than specifying all the settings for a field, you can choose a field that already exists, and which reflects what you want your object to look like (or acts like you want it to act), to inherit from. Using inheritance saves time.

For example, you could create the following two entities, if you needed to create two types of Project entities in a project management application. One would represent projects that were billed by the hour; the other would represent projects that were bid by job.

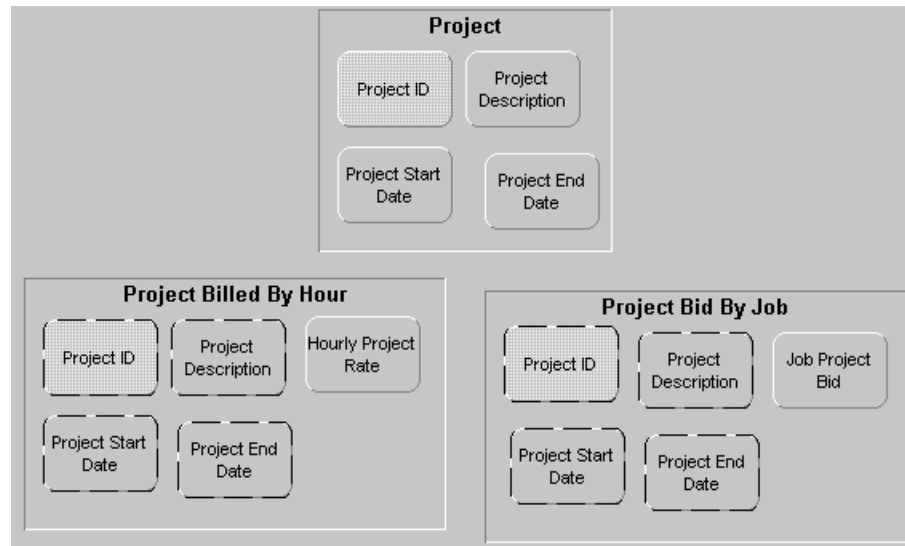


Creating two types of Project entries would require a lot of duplicated effort. The identifier, description, and start and end dates are the same for both entities. Why should you create all of the functionality for these identical fields twice?

Using inheritance enables you to create a single Project entity with all of the common structure and functionality, and create two specialized entities that inherit from that one. Then, you can customize each type of project from there.

The following graphic shows the entities Project Billed By Hour and Project Bid By Job as they would appear after they have inherited from Project.

Note: The inherited attributes have dashed borders.



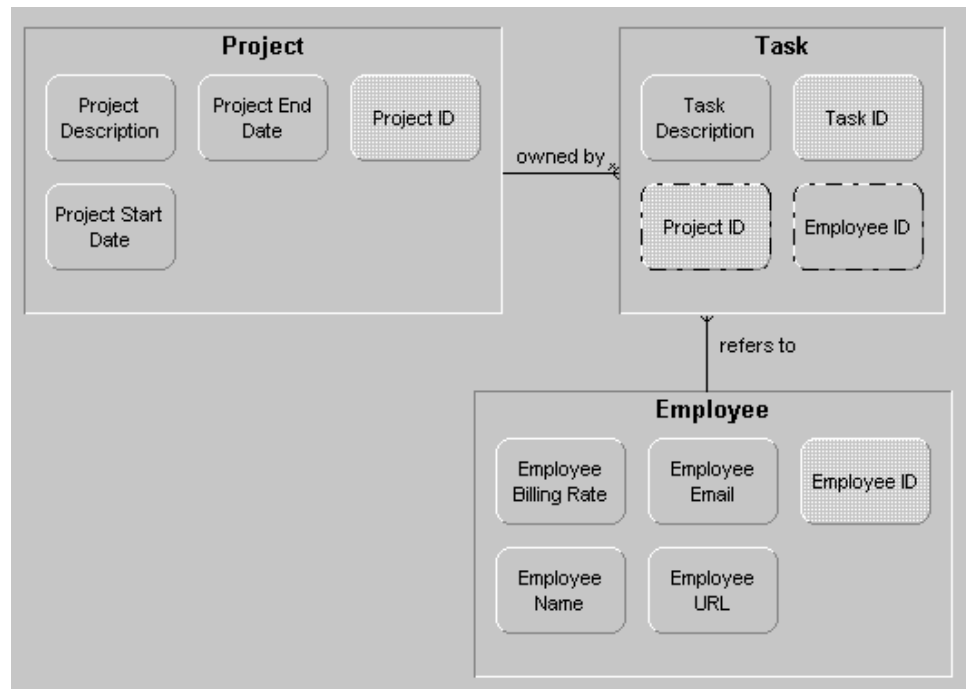
Project Management Application

In this tutorial, you create a basic project management application. In the application, a project consists of a group of tasks, and each task is assigned to one employee. It contains three entities:

- A project entity, with information about projects
- An employee entity, with information about employees
- A task entity, with information about tasks

For the purpose of this tutorial, the model is very simplistic. If you have developed project management systems, you may find parts of the model that you would do differently. Keep in mind that it is only an example.

The following is an entity-attributes diagram of the application:



Look at what this diagram shows about the application. If you have worked with entity-attributes or entity-relations diagrams before, you can see that:

- It tracks projects and projects have tasks
- 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 the owning project is deleted, you will want to delete all of its tasks
- 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 a task is deleted, the employee assigned to it is not deleted

You can see all of this without having to look at any code or a database schema. This diagram shows useful information, which CA Plex uses to generate the application. You will learn how to create this diagram in the chapter “Diagramming Entity Relationships.”

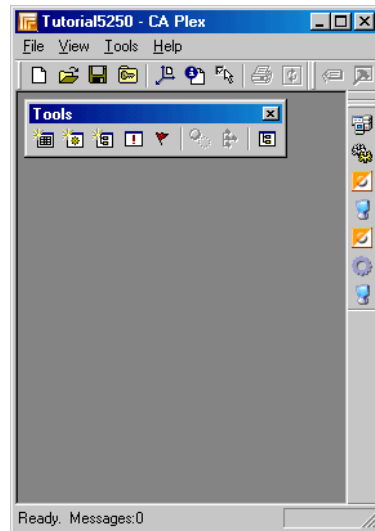
A First Look

This section gives you a quick tour of the CA Plex environment and its online help.

Opening CA Plex

1. Click Start, Programs, CA, CA Plex r7.0, and click the CA Plex icon. CA Plex opens.

This graphic shows you the different ways you can position toolbars. You can dock toolbars just below the menu, on either side, or set them to float in the workspace.



Notice where your toolbar resides in the CA Plex environment.

2. From the CA Plex File menu, select Open.
3. In the Open File dialog, select Tutorial5250.mdl from the Tutorial subdirectory of your installation directory, and click OK.

By default, when the model opens, the Object Browser appears and the name of the model appears in the title bar. These are the only indications that you have opened your model. You will learn about the Object Browser in *Introducing the Object Browser* in the chapter “Defining Entities.”



4. If you need to stop working on this tutorial before reaching the end of the current lesson, you can save it by clicking the Save toolbar button. You are also prompted to save your changes when you close certain editors after making changes.

Note: When the editors prompt you to save, only the changes made in that editor are saved and not the whole model.

Closing CA Plex

From the File menu, click Exit to close CA Plex.

Setting Up Generate and Build Options

When you generate and build code, you need to tell CA Plex about your application development environment. To do this, you set options in the CA Plex Generate and Build window.

The options you specify for a model are stored in a file in the same directory as the model. It has the same name as the model, but has a .bld extension (in the case of this tutorial, the build file is Tutorial5250.bld).

Before continuing in this section, ask your System i Network Administrator what type of connection you will make to the System i server (TCP/IP), and get the following information. For TCP/IP connections:

- A fully qualified network address
Such as myAS400.local.mycompany.com
- Server port
- Code page
- User name
- Password

Specifying Build System Settings

Now that you have the necessary information to set up a network connection to your System i system, you are ready to specify those settings in the Generate and Build options for your model.

The following steps outline how to set up a TCP/IP connection. The process involves three main steps:

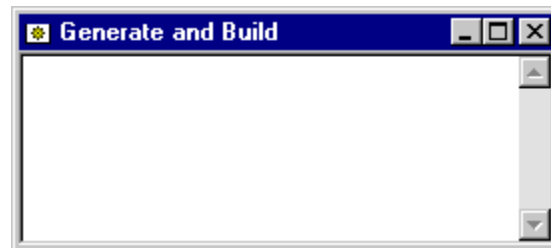
1. Creating a System i system.
2. Specifying settings for the system.
3. Indicating that the model should use the newly created system when building for the System i 5250 environment.

To specify build system settings

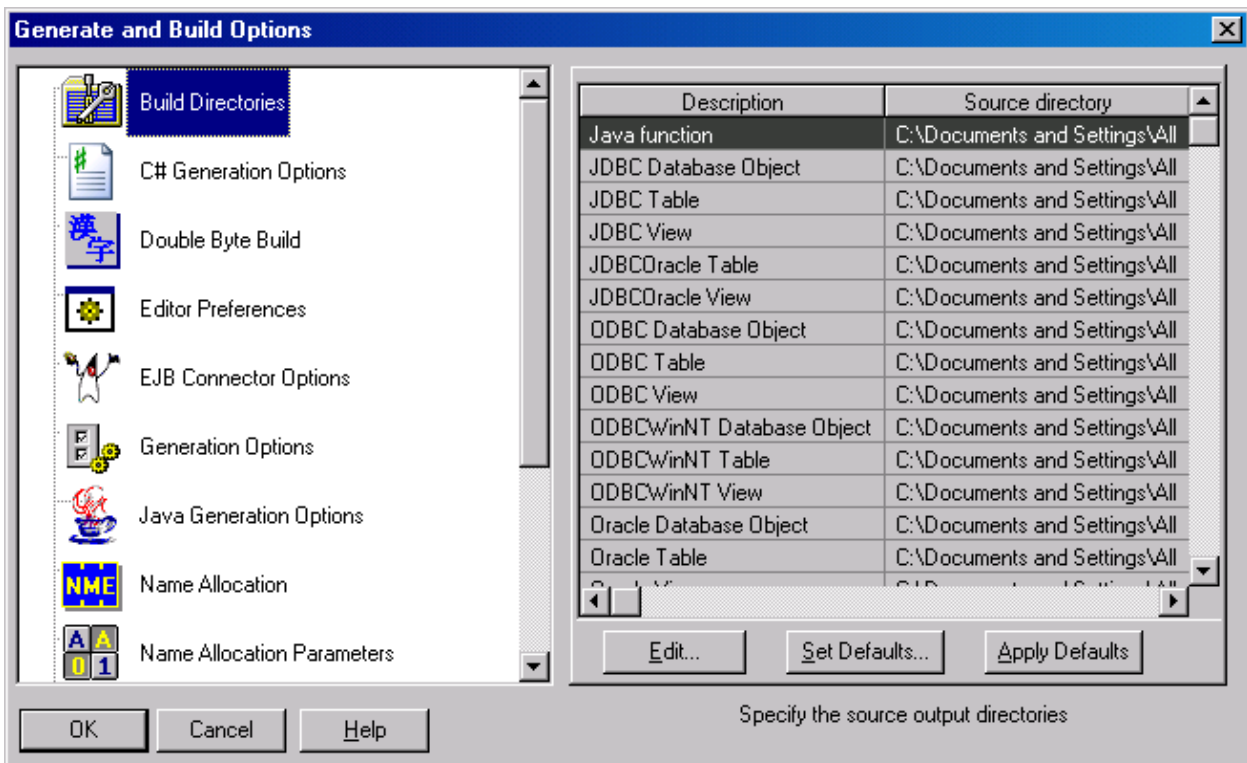
1. If it is not already running, start CA Plex and open Tutorial5250.mdl. For instructions, see Opening , earlier in this chapter.

2. From the Tools menu, choose

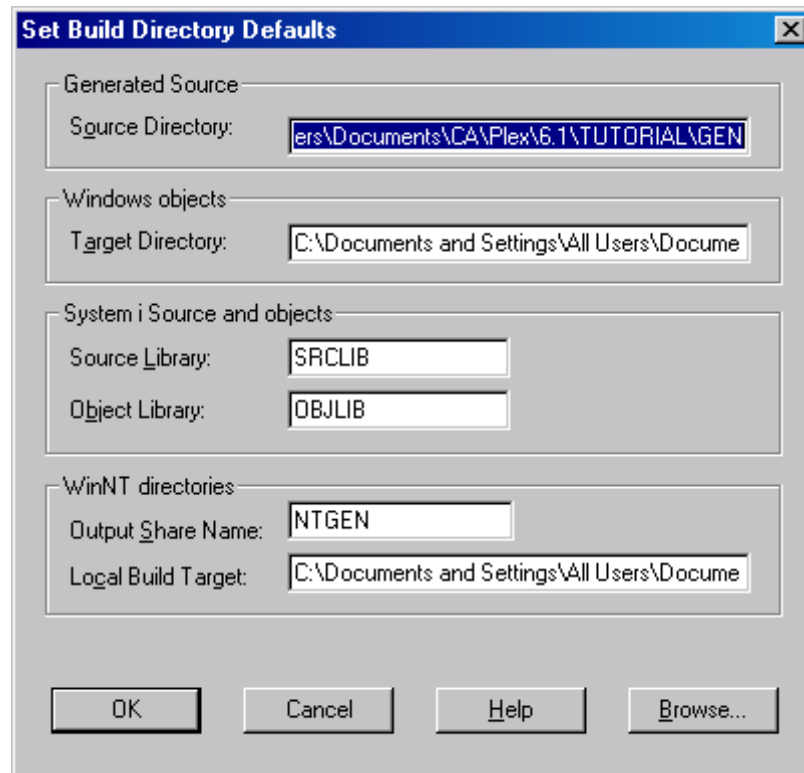
Build window opens



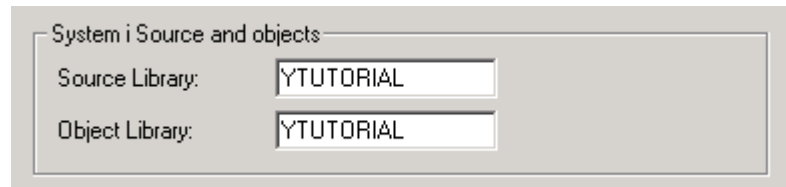
3. Click the Generate and Build Options toolbar button. The Generate and Build Options dialog opens:



4. Click Set Defaults. The Set Build Directory Defaults dialog opens:

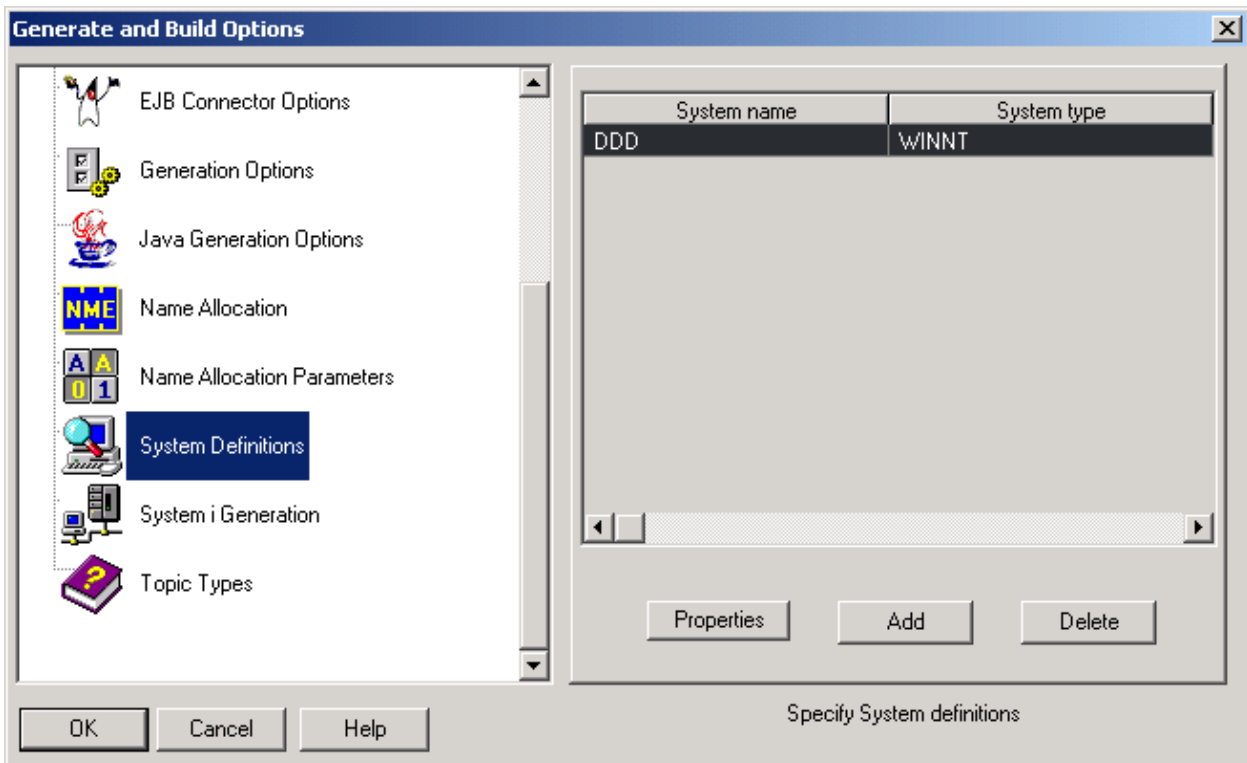


5. Change the Source Library and Object Library fields to YTUTORIAL.



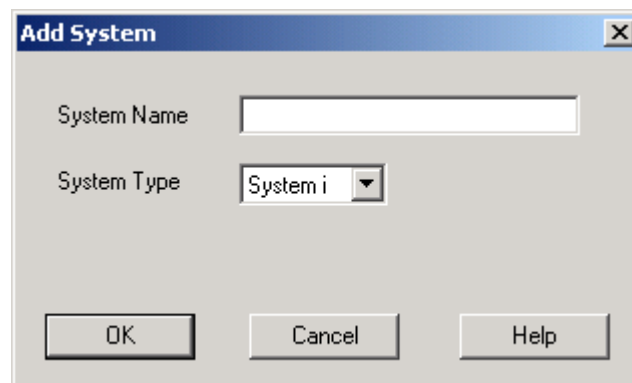
Note: You should have created the YTUTORIAL library on your System i during the CA Plex installation process. For more information, see Getting Started

6. Click OK. The Generate and Build Options dialog opens.
7. To assign the Source directory location as the default location of the generated source directories, Click Apply Defaults.
8. On the left pane of the dialog, select System Definitions. The display changes:

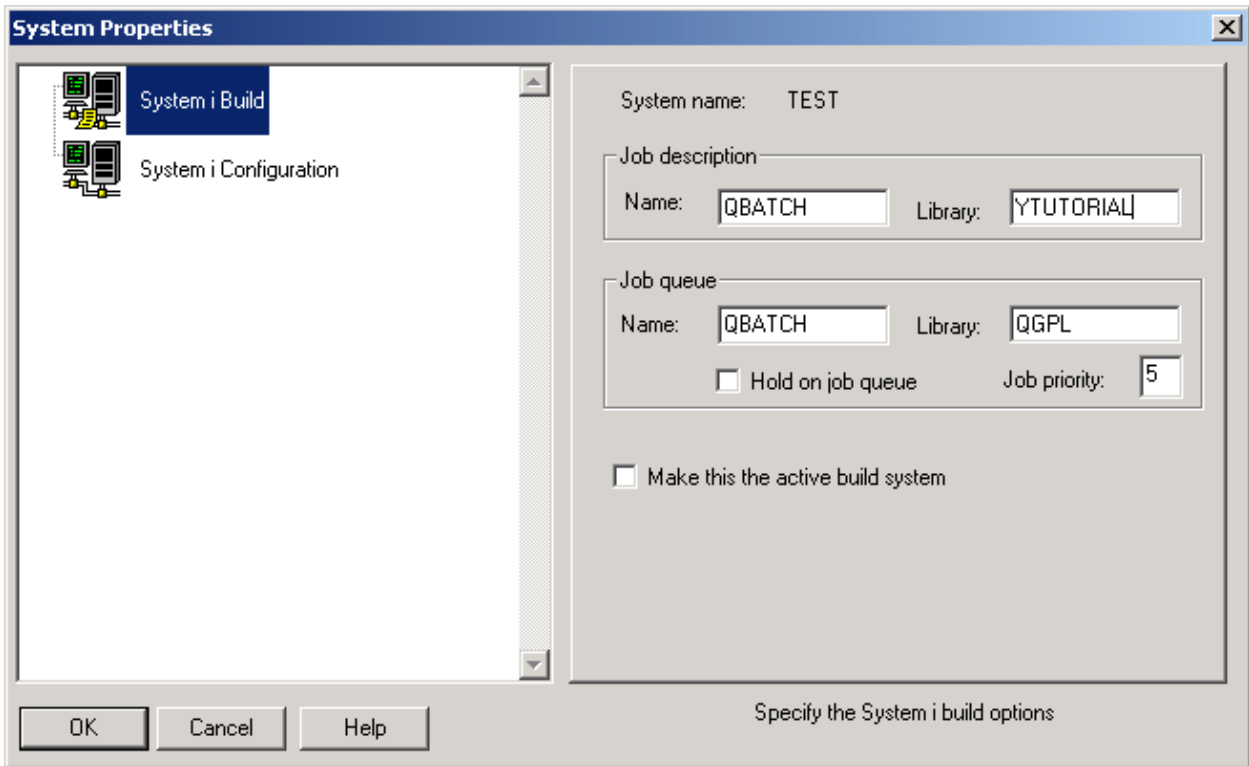


Note: The section on the right looks different on your screen.

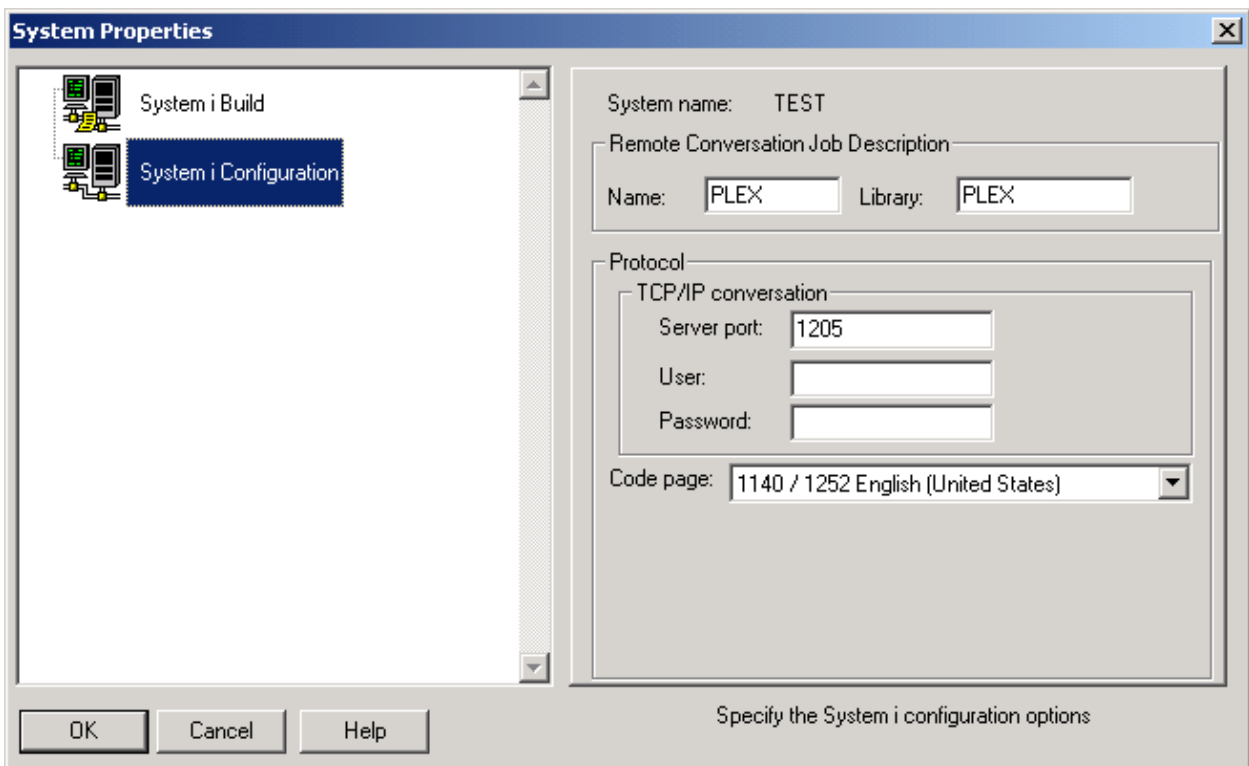
9. Click Add. The Add System dialog opens



10. Type your system name in the System Name field. Remember that you received this information from your System i network administrator before starting this process.
Make sure System i is selected in the System Type field, then click OK.
11. The new system name appears in the Generate and Build Options dialog. Now that you have created a System i system, you need to specify its settings.
12. Select the system name you just added and click Properties. The System Properties dialog opens.
13. Change the Job Description Name to **QBATCH**, and the Job Description Library to **YTUTORIAL**.



14. Make sure that the Hold on Job Queue check box is not checked and that the Job Priority value is set to 5.
15. In the left pane, select the AS/400 Configuration option. The System Properties dialog changes:



16. In the Protocol section, enter the appropriate information in that option's section.

17. Click OK to close the System Properties dialog.

Now that you have created a system definition, you must indicate that CA Plex is to use that system when generating and building for the System i 5250 environment.

18. Click OK again to exit the Generate and Build Options dialog.

Chapter 2: Defining Entities

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 lesson, you define what a project is, and what you want to store about each project. You use the CA Plex Class Libraries to streamline the process.

This section contains the following topics:

[Introducing the Object Browser](#) (see page 27)

[Introducing the Model Editor](#) (see page 30)

[Defining the Project Entity](#) (see page 31)

[Generating and Building the Project Entity](#) (see page 38)

[Using Your Generated Application](#) (see page 42)

[Chapter Review](#) (see page 45)

[Helpful Hints](#) (see page 45)

Introducing the Object Browser

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

In this lesson, you create an entity called Project. The Project entity uses fields to store information about itself, such as its start and end dates. You define functions to enable end users to create, modify, and delete projects. Entities, fields, and functions are all types of objects. For more information on definitions of the various object types, see Objects in the chapter “Introduction.”

Opening the Object Browser

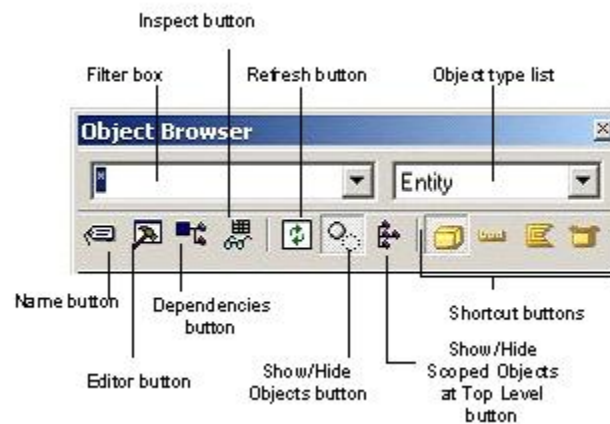


To open the Object Browser, choose Object Browser from the Tools menu, or click the New Object Browser toolbar button. By default, CA Plex opens an Object Browser when you open a local model, even if an Object Browser is already open.

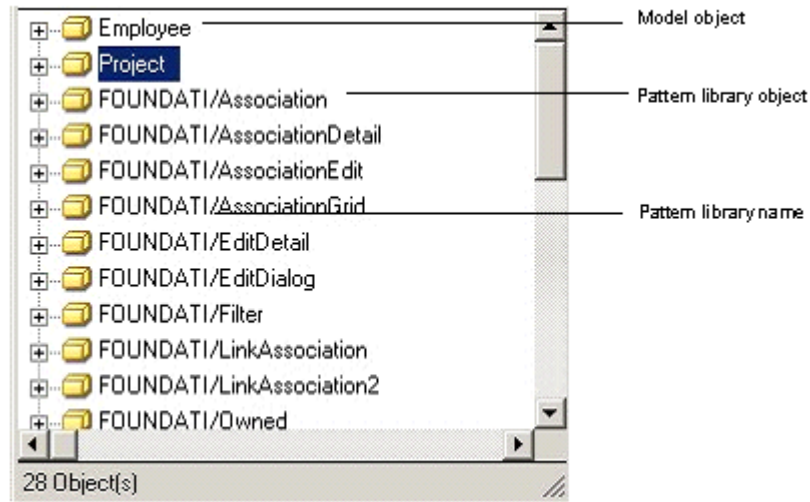
The Object Browser is the bucket that displays all of the objects in a model. In it, you can see the available library objects, and the objects that you define in your model (you will define objects in your model later in this chapter).

A library is a collection of objects that you import into your model. CA provides a set of class libraries with CA Plex; in this tutorial, when you see the term library, it refers to those libraries. However, any model that you import into your model is a library.

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



You select which type of object you want to view in the Object Browser (you see one type of object at a time). 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 to another object (such as an entity or a view) are displayed only when the Object Browser is focused on that object type.



Showing Library Objects



You can use the object list to specify which type of object you want to see. If the Object Browser is empty, click the Show/Hide Library Objects button to show the library objects. The button appears on the toolbar and Object Browser.

You can tell that an object is a library object because, by default, it has the library name in front of the object name. For instance, the fourth entity in the preceding graphic is OBAS400/Entity With Direct Format, which means that this entity comes from the OBAS400 class library.

Introducing the Model Editor

To create an application, you define objects and the relationships between them. In this lesson, you create an entity called Project, specify what information you want your application to store about projects, and add functionality that enables end users to create, edit, and delete projects.

In CA Plex, you specify relationships between objects in a model using triples. A triple is comprised of three parts: the source object, verb, and target object.

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

Project **known by** Project ID

The triple is an instance of the abstract triple ENT **known by** FLD. Whenever you want to designate a unique identifier for an entity, also known as its primary key, you use this triple, replacing ENT with the name of your entity, and FLD with the name of your key field.

Defining the Project Entity

You use the Model Editor to view, add, edit, and delete triples. The following example shows you the Model Editor as it appears after you complete this lesson.

Opening the Model Editor



To open the Model Editor, choose Model Editor from the Tools menu, or click the New Model Editor toolbar button. You use the Model Editor to view, add, edit, and delete triples. The following graphic shows you the Model Editor as it appears after you have completed this lesson.

The screenshot shows the 'Model Editor - Diagram' window. At the top, there are three input fields: 'Project', 'is a ENT', and 'Grid maintained entity'. Below these are two dropdown menus: 'Entity' and '<All>'. The main area is a table with the following data:

Project	is a	OBASE/Grid maintained entity
	known by	Project ID
	has	Project Description Project Start Date Project End Date
Project Description	is a	OBASE/Narrative
	impl name	DC56A
Project End Date	is a	OBDATE/ISO date
	impl name	DC58A
Project ID	is a	OBASE/Code
	impl name	DC55A
Project Start Date	is a	OBDATE/ISO date
	impl name	DC57A

Defining the Project Entity

As described in the section Project Management Application, the application you are building stores information about projects, the tasks that are part of a project, and the employees that are assigned to each task. In the chapter "Diagramming Entity Relationships," you define the Employee entity, and in the chapter "Defining Owned by Relationships," you define the Task entity.

Specifying Attributes for the Project Entity

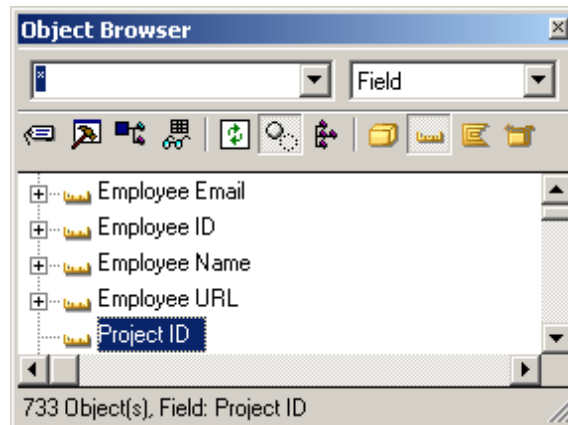
One of the first considerations when defining an entity is what data it will store. In this section, you define fields for the Project entity: an identifier, a description, and start and end dates for a project.

To define the fields of the Project entity:

1. If the Model Editor is not open, open it.
2. From the object type list, select **Entity**.
3. In the source object field, enter **Project**.
4. From the verb list, select **known by** FLD.
5. In the target object field, enter **Project ID**, and then press Enter.

You just created the triple Project known by Project ID. This triple creates a primary key for the Project entity. For more information on how to change misnamed objects, see the section Misnamed Objects in this chapter.

6. If the Object Browser is not open, open it.
7. On the Object Browser, click the Fields toolbar button to display field objects.



8. Project should still appear in the source object field of the Model Editor. From the verb list, select *has FLD*.
9. In the target object field, enter **Project Description**.
10. Press Enter.

You have created the triple Project **has** Project Description, which defines the field Project Description for the Project entity. You will 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 may have more than one project with the same text in the description field.

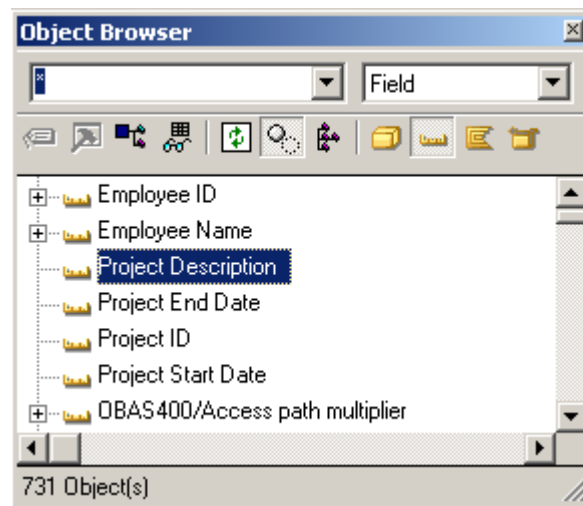
11. Repeat Steps 8 and 9 to create the following triples:

Project **has** Project Start Date

Project **has** Project End Date

12. On the Object Browser, click the Refresh toolbar button. The Object Browser shows the new fields:





Defining Field Properties

The fields you just defined represent character and date data types. These represent different kinds of fields. The identifier and description fields will hold text, and the start and end dates will hold dates. Currently, your model only indicates that these fields exist, and that they belong to the Project entity, but they have no information as to the type of data they store.



In the next step, you further define Project's fields using *inheritance*. Inheritance is the mechanism that enables an object to adopt the properties of another more general object. For more information on this concept, see Inheritance in the chapter, "Introduction."

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

- Validate data entered in the fields (which ensures that an end user does not accidentally enter the year in the month position of the Project Start Date field, for example)
- Display data to the screen appropriately (such as displaying an underscore to show how long the Project Description field entry can be)
- 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. For more information on showing library objects, see Show Library Objects in this chapter.
2. Drag the Project ID field from the Object Browser to the source object field of the Model Editor (to select the object in the Object Browser, click the name, not the icon to the left of the name).

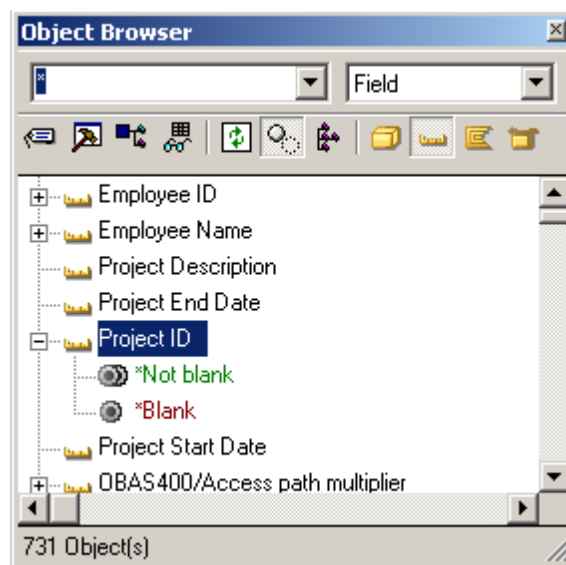
Note: The cursor changes to a closed parcel icon  when you are dragging an object from the Object Browser. It changes to an open parcel icon  when it is over an editor, or any area that can accept the object.

3. From the verb list, select **is a FLD**.
4. From the Object Browser, drag the library object OBASE/Code to the target object field, then press Enter.



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

5. Click the plus icon to expand the field. You can see that Project ID now has the values *Blank and *Not blank, but you cannot tell much else about what it inherited from OBASE/Code.



- 6. Drag the Project ID field from the Object Browser to the body of the Model Editor.
When you drag one or more objects to the body of the Model Editor, the display changes to show you the triples that define those objects.

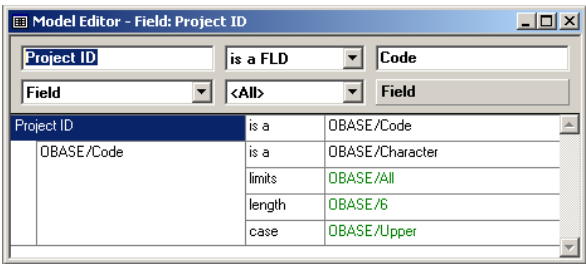


- 7. If you want to make changes to an object and its scoped objects, but do not want to be distracted by information about the other objects in your model, use this drag and drop technique. To clear the focus so that all of the model's objects are again displayed, click the Clear Focus toolbar button.

When you drag the Project ID field to the Model Editor, it focuses on this field, showing the triple Project ID is a OBASE/Code. This triple 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 changes to show another level of detail.



Now, you can see that Project ID has inherited a data type of character and a length of six through the inheritance triple OBASE/Code is a OBASE/Character.

Note: In Step 4, you dragged the library object OBASE/Code from the Object Browser to the Model Editor to enter it in the target object field. You can actually type the value into the target object field (without the library name) to accomplish the same thing.

Important! If you mistype the object name, you are likely to create a new object with the mistyped name.



9. Change the Model Editor display back to show a single level by clicking the One Level toolbar button.
10. Reset the Model Editor display by clicking the Clear Focus button.

11. Drag Project Description from the body of the Model Editor to the source object field.

12. Enter **Narrative** in the target object field and press Enter.

Notice that the Model Editor displays the triple as:

Project Description **is a** OBASE/Narrative

This indicates that you correctly spelled the class library field, as the Model Editor replaced your entry with the library object by the same name.

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. By default, fields that inherit from OBASE/Narrative have a specified length of 25. To enable your end users to specify a longer project description, add the following triple:

Project Description **length** 30

14. Now, using either the method explained in Steps 2 through 4, or the method explained in Steps 11 and 12, create the following triples:

Project Start Date **is a** OBASE/ISO Date

Project End Date **is a** OBASE/ISO Date

This inheritance gives the fields an ISO date format and includes built-in functionality to ensure that the date values end users enter are valid dates.

Use the process described in Steps 7 and 8 to look at the characteristics these fields inherited from the class library fields.

Using Inheritance to Define the Project Entity

You have now defined the fields in which the Project entity will store data, and specified the class library fields from which those fields inherit. In this 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. The inheritance triple gives your entity the objects necessary to display and process a user interface, and to read data from and write data to a database.

To add functionality to the Project entity:

1. Click the Entity button on the Object Browser.

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.

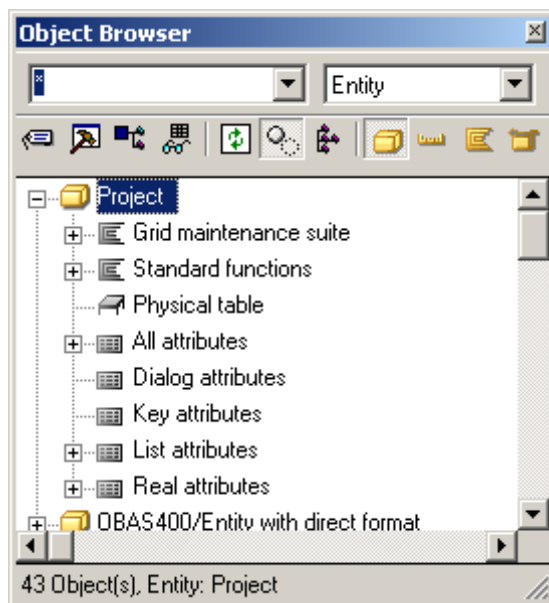
2. Drag Project from the Object Browser to the source object field of the Model Editor.
3. From the verb list, select **is a** ENT.
4. In the target object field, enter **Grid Maintained Entity** and press Enter.

You just created the triple Project **is a** OBASE/Grid Maintained Entity. This indicates that Project inherits the structure and functionality of the Grid Maintained Entity object in the OBASE class library.

For more information on defining entities, look in the online help index. For information on the Grid Maintained Entity object, select the pattern in the Object Browser and press Shift+F1.



5. By inheriting from Grid Maintained Entity, the Project entity now has some scoped objects. In the Object Browser, click the Refresh button.
6. The Project entity you just created appears at the top of the Object Browser. Click the plus sign to the left of the entity icon to expand Project:



You can see that among the objects it inherited are several functions (with scoped panels) and views, and a physical table. These objects enable Project to display a user interface, and to store data to and retrieve data from a database.

Specifically:

- Project.Grid Maintenance Suite.Edit Grid User Interface displays the panel scoped to it
- List Panel, which is scoped to the Edit Grid User Interface function, stores the layout of the Edit Project panel
- The functions scoped to the views access the database to add, delete, and change records

You will learn more about panels in *Modifying the User Interface*, and functions and views in the chapter "Defining Owned By Relationships."

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. Now, you are 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 executable 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 created.

To generate and build the Project entity

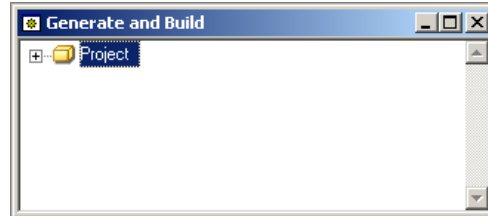
1. From the Tools menu, choose Generate and Build, or click the New Gen and Build toolbar button.



The Generate and Build window appears. The Message Log pops up when you open the window. You can ignore its messages for now, and minimize it.

2. If the Generate and Build window shows library objects, turn them off with the Show/Hide Library Objects toolbar button.

The Generate and Build window now shows only the Project entity:



3. Select the Project entity.

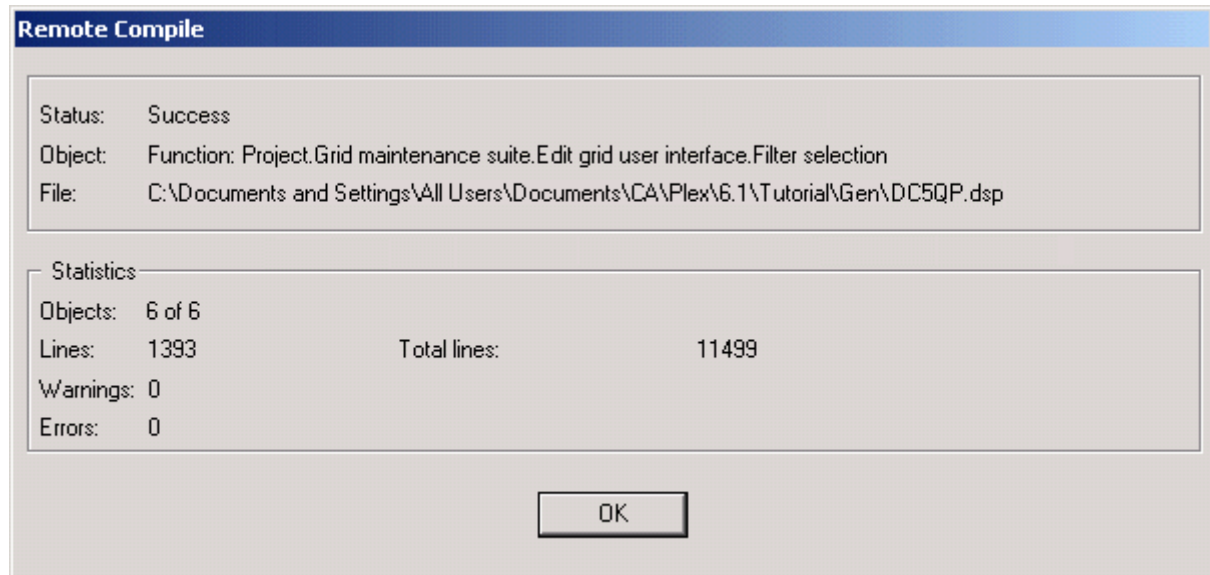


4. Click the Generate and Build toolbar button. When you click this button, the Project entity expands, showing all of its scoped objects that can be generated. Those that will be generated are highlighted.
5. A Confirm Generate dialog appears, indicating the number of objects that are generated. Not all of the scoped objects are selected. Click Yes.
6. CA Plex generates those objects, and then summarizes the generation process. Click OK on the summary dialog when the generation completes (until the generation is complete, the button says Cancel).

Note: Generating these objects causes warnings; one indicates that the length of a field on a panel differs from its length in the program, and the other indicates that no event was found for a grid subfile selector value *Blank. Disregard these warnings, as they do not affect your compiled application.

7. CA Plex prompts you to compile and build the objects. Click Yes both times.

CA Plex then sends the generated code to the System i you specified previously (For more information, see Specifying Build System Settings). The window shows the progress of transferring the source code on the Remote Compile dialog.

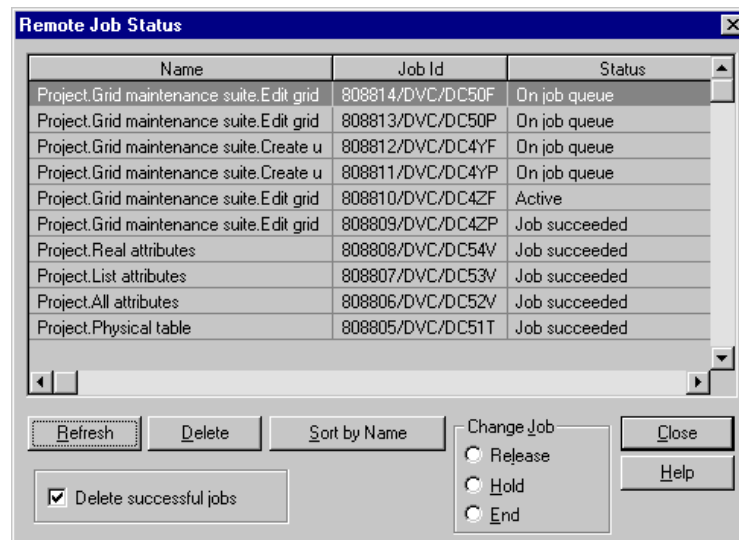


- After the code is transferred to the System i server, the server compiles it. Click OK on the Remote Compile dialog.



- Click the AS/400 Build Status toolbar button to display the status of your build.

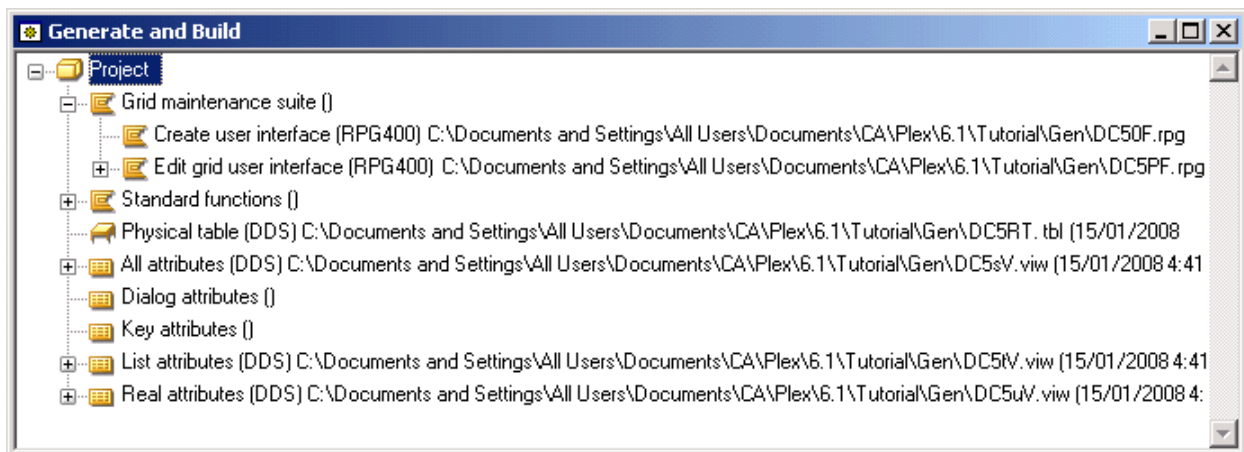
The Remote Job Status dialog appears.



10. Click Refresh until all of the jobs indicate that they successfully compiled.
11. When the build is done, close the Remote Job Status dialog.
12. Save your model.
13. From the Build menu, choose System i Message File, then Generate and Build.

Note: Building the message file generates an error, indicating there was no text for "Format ODBC time value." Do not worry about this error; it will not keep your generated application from running properly.

14. From the Build menu, choose System i Field Values File, and then Generate and Build.
15. Find the function Project.Edit Grid User Interface, in the Generate and Build window, and note its implementation name (following the object name).



16. Start a 5250 session.
17. Enter **call AA4ZF** ", where AA4ZF is the implementation name you noted in Step 15.

Your generated application starts, and should look like this:

```
Date . . 7/23/98 Edit Project                               Time . . 14:52:57

Type options, press Enter.
3=Copy    4=Delete
? Project ID Project Description      Project Start Date Project End Date

F3=Close F5=Refresh F6=Add F10=Update F17=Filter...
```

Using Your Generated Application

By entering a few triples, you have created a fully functional application. 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.

Adding a New Project

To add a new project:

1. You started your application in the previous section. Now, press F6 to add a new project.
2. On the Add Project panel, enter the following values, and then press Enter.

Value	Enter
Project ID	PROJ01
Project Description	McCreedy database app.
Project Start Date	121098
Project End Date	010199

The display changes after you enter a project.

Date . . . 7/23/98 Add Project		Time . . . 16:09:41
Project ID	<input type="text"/>	
Project Description . . .	<input type="text"/>	
Project Start Date . . .	<input type="text"/>	
Project End Date	<input type="text"/>	
Last Project added		
Project ID	PROJ01	
No. added	1	
F3=Close F5=Refresh		

3. Add the following two projects to the database:

Value	Enter
Project ID	PROJ02
Project Description	Email client for IS
Project Start Date	030199
Project End Date	040499

Value	Enter
Project ID	PROJ03
Project Description	Chg McCready app to ODBC
Project Start Date	050599
Project End Date	060599

- Press F3 to exit the Add Project panel. Your main panel should look like this:

```
Date . . 8/05/98 Edit Project
```

```
Time . . 9:44:33
```

```
Type options, press Enter.
```

```
3=Copy 4=Delete
```

```
? Project ID Project Description Project Start Date Project End Date
```

— PROJ01	<u>McCready database app.</u>	<u>121098</u>	<u>10199</u>
— PROJ02	<u>Email client for IS</u>	<u>30199</u>	<u>40499</u>
— PROJ03	<u>Chg McCready app to ODBC</u>	<u>50599</u>	<u>60599</u>

```
F3=Close F5=Refresh F6=Add F10=Update F17=Filter...
```

```
Bottom
```

- Press F3 again to exit the application.

For now, do not delete or change any of the projects you entered in the preceding steps. The following steps show you how to do so.

Deleting a Project

To delete a project:

- On the Edit Project panel, tab to the grid selector field for the project you want to delete.

Note: CA Plex uses the term grid for an System i subfile.

- Enter 4 in the field, then press Enter.

Changing a Project

To change a project:

- On the Edit Project panel, tab to the field you want to modify (you cannot modify the primary key field).
- Type over the existing value with the new value, and then press Enter.

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, the next time you build, you will lose all of the data you just entered.

You can prevent the loss of 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).

Remember, 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 you entered in the table will be lost. If you want to preserve data entered after rebuilding, make sure you reset the **implement SYS** value to No.

To set the generation status for the Project entity table

1. From the Object Browser, drag Project.Physical Table to the source object field in the Model Editor.
2. From the verb list, select **implement SYS**.
3. From the target object field, select the value No, then press Enter.
4. Save your model.

Chapter Review

In this lesson, you:

- Defined the Project entity
- Defined fields for the Project entity
- Set the Project entity and its fields to inherit from class libraries to display a user interface, validate data entry, and store data in a database
- Generated, built, and ran the application, adding data to the database

Helpful Hints

This section gives you additional information pertinent to this lesson.

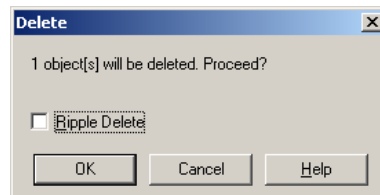
Misnamed Objects

When you mistype the name of a library object, you end up with objects you did not intend to create.

Deleting Objects You Unintentionally Created

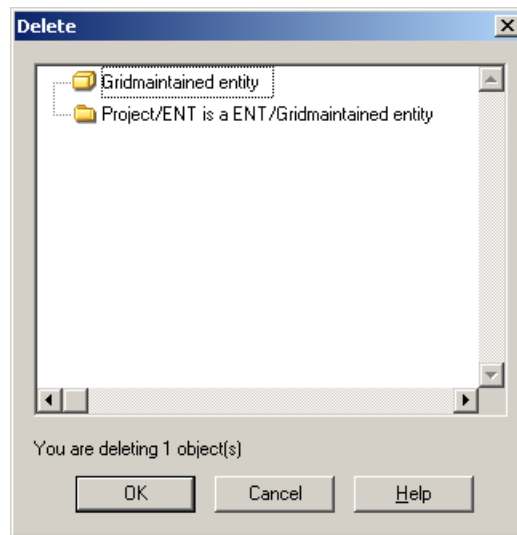
To delete objects you unintentionally created:

1. Set the Object Browser so that it shows the type of object you want to delete.
2. Right-click the errant object, choose Object, and then Delete from the pop-up menu. The Delete dialog appears.



3. Click OK. The Delete Object dialog appears.

This dialog shows you the objects that will be deleted. The following example shows you the objects that are deleted when deleting an entity called Gridmaintained entity.



4. Click OK.

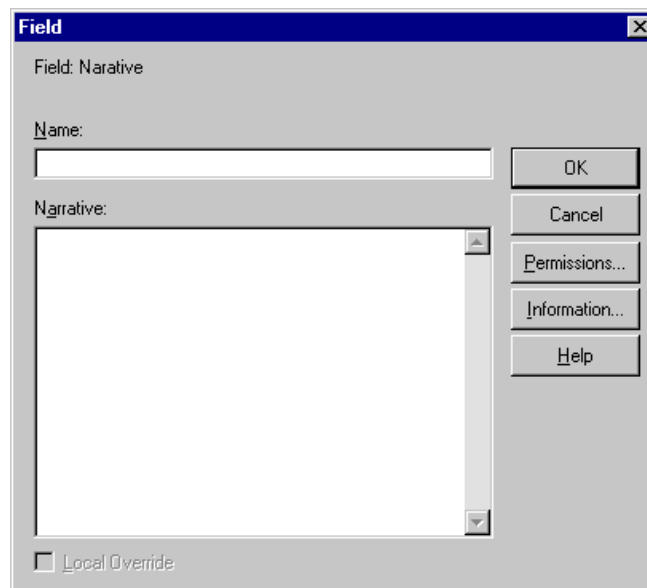
Renaming Objects Whose Names You Mistyped

To rename these objects:

1. Set the Object Browser so that it shows the type of object you want to rename.



2. Select the object and click the Name toolbar button. The Rename Object dialog appears:

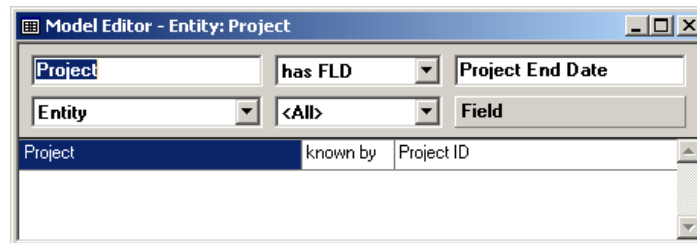


3. Replace the name in the Name field with the new name.
4. Click OK.

Triples

You spend a lot of time working with triples when creating an application. Here are tips on how to select, delete, and change triples:

- To select a whole triple, click the center column of the Model Editor, where the verb is.



- To delete a triple, select the triple in the Model Editor and press the Delete key.



- To change a triple, select the triple in the Model Editor, change the value in the source object field, the verb list, or the target object field (or a combination of these), and click the Change Triple toolbar button.

Chapter 3: Modifying the User Interface

In this chapter, you will learn how to change the way your application looks. You modify the Project entity's panel design, which it inherited from the class libraries, to make it look better.

This section contains the following topics:

[Introducing the Panel Designer](#) (see page 49)

[Changing Individual Controls](#) (see page 55)

[Changing More Than One Control](#) (see page 56)

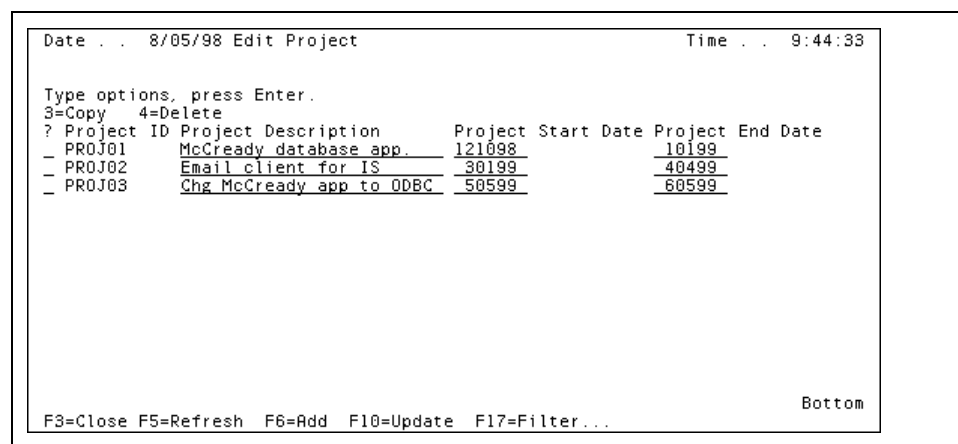
[Adding Static Text](#) (see page 57)

[Generating and Building](#) (see page 58)

[Chapter Review](#) (see page 59)

Introducing the Panel Designer

You probably noticed when you used the application as directed in *Getting Started* in the chapter, "Your First Application in 20 Minutes," that it did not look right. The headings for the columns all start with Project even though the panel title is Edit Project. The long column headings cause the columns to extend all the way to the right edge of the panel, with wide gaps between the values.



In this chapter, you will use the Panel Designer to fix these problems:

- You will shorten the column headings.
- Move the columns closer to each other for easier record reading.
- Add an entry to the Options area above the columns, in preparation for Defining Owned By Relationships in which you define the Task entity.

Modifying the User Interface

User interface refers to all the panels with which your end users interact. You use the Panel Designer to modify the appearance of your user interface.

To open the Panel Designer:



1. In the Object Browser, click the Entity button.
2. Expand the Project entity, then expand the Grid Maintenance Suite function, and then expand the Edit Grid User Interface function.
3. Right-click List Panel and choose Editor from the pop-up menu.

The Panel Designer appears, displaying three windows:

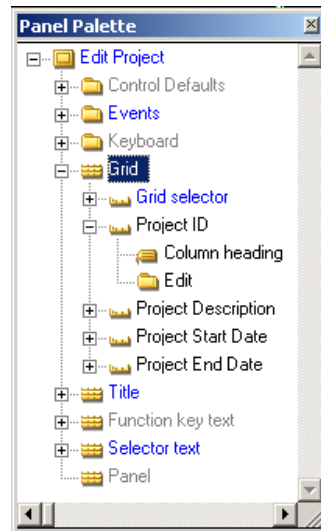
- Design Window: Shows you what the panel will look like when it is generated.
 - Panel Palette: Shows you the various panel elements.
 - Property Sheet: Enables you to make or change settings for the elements of the panel.
4. Arrange the elements so that they do not overlap, enabling you to work with the contents of each.

Design Window

The Design Window is the Panel Designer's main window. Using this window, you can create or change a window and see how your changes affect the panel's appearance. You can select, move, and resize fields and other user interface elements using this window. When you make visual changes to the panel and its elements using other windows, the changes appear here.

A panel's elements are grouped into regions. A region is the equivalent of a DDS format.

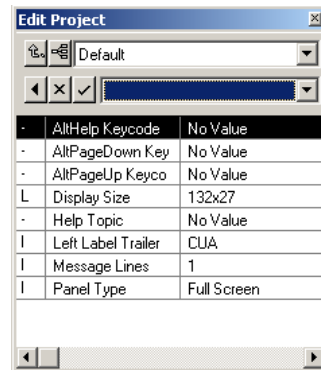
When a region is expanded, as the Grid region is in the following example, you can see the elements contained in that region. In the case of the Grid region, this includes the Grid Selector, Project ID, Project Description, Project Start Date, and Project End Date fields.



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

Property Sheet

The Property Sheet shows you the properties of the elements that you select in either the Design Window or the Panel Palette, and enables you to change the settings for those properties. Among other things, you can change color, (in some cases) size, and position. To see what you can change about an element, select it in the Design Window or the Panel Palette, and check its properties on the Property Sheet:





The properties you can change depend on the type of element selected. For example, you can set the Allow Blank property for a field, which determines if a blank field is a valid entry. But end users cannot enter text for a label, so if you select a label on a panel, the Property Sheet does not display this property.


To visually indicate that the Project ID field is a key field, you will now change it so that its text appears in turquoise.

To change the text formatting for the Project ID field:

1. If you closed the Panel Designer, reopen it.
2. In the Panel Palette, expand Edit Project. Then expand the Grid region, and then the Project ID field.

Notice the icons used. The  icon represents a region. Remember, a region is an area of a panel, and that there are five regions on this panel. In this chapter, you make changes in the Grid region, which is the section that displays the four fields you defined in *Getting Started* in the chapter "Your First Application in 20 Minutes."

The  icon represents a field. You expand the field icon to view or select the components (controls and labels) of the field.

3. Click the different regions  in the Panel Palette.

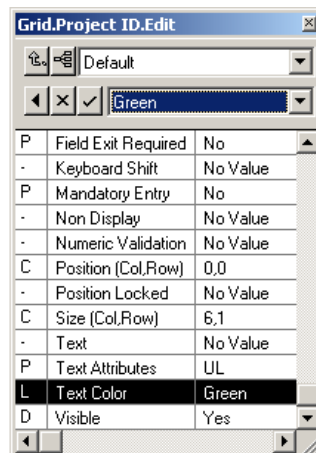
Note: As you select a region, it is highlighted in the Design Window. When a region is highlighted, a dotted line appears around its borders.

4. Under the Project ID field, select Edit control.

This selects the window control part of the Project ID field. Notice that in the Design Window, the control is selected. Notice also, that the Property Sheet changed to show the properties that you can modify for the edit control.

5. On the Property Sheet, click the Text Color property.

The current value for the property (Green) appears in the top area of the Panel Palette:



Equation 1: PLEX--Property Sheet (1)



The area at the top may display as an edit box, edit box with a Three Dots button, or a combo box (as in the previous graphic), depending on the property.

6. From the combo box displayed at the top of the Property Sheet, choose Turquoise.



7. Leave the panel open for the next steps.

Notice that the color of that field changes in the Design Window.

8. Leave the panel open for the next steps.

Changing Individual Controls

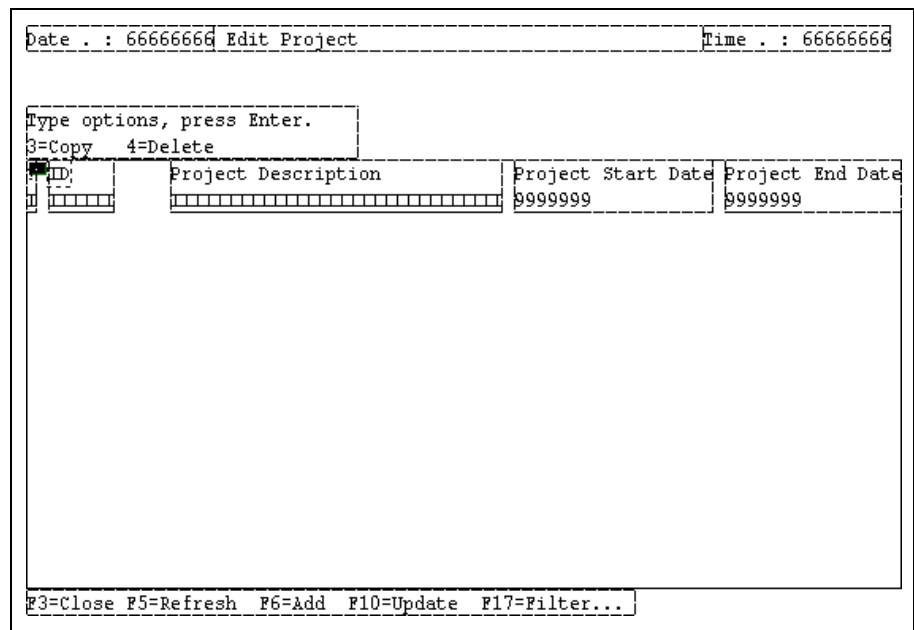
When you edit panels, you can make changes to individual elements, as you did in the previous steps. In this section, you make additional changes to elements on a panel. Currently, the labels for the columns on the panel open in the Panel Designer all start with Project, even though the panel's title is Edit Project. Because this is redundant and takes up screen space, you will delete the word Project from each of these labels.

Next, you will modify the properties of the fields in the Grid region.

To change the field properties:

1. In the Panel Palette, expand the Grid region, and then expand each of the fields under it.
2. Select Column Heading under the Project ID field.
3. In the Property Sheet, change the label's text value from ID to Project ID. Click the Apply button.

Notice that the field's label changes in the Design Window:



4. Repeat Steps 2 and 3 to add the word Project to the other three labels.



5. Click the Save toolbar button to save the changes in your model without closing the Panel Designer.

Changing More Than One Control


When making changes to a panel, you may need to make a change to more than one control. In the following steps, you will move the Description, Start Date, and End Date fields to the left to make space between the right edge of the End Date field and the edge of the screen.

To change multiple controls:

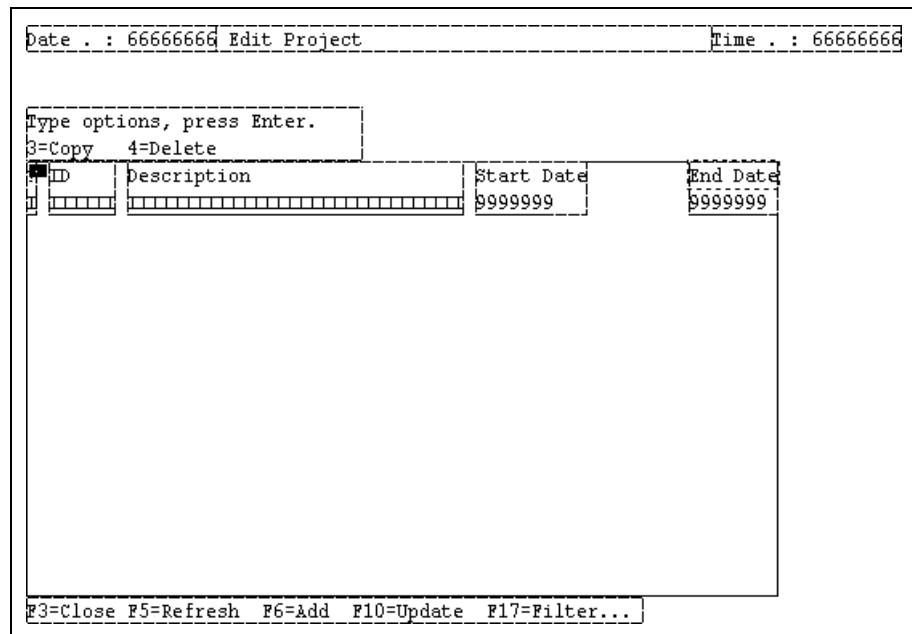
1. In the Panel Palette, select the Project Description field. Press and hold down Ctrl; then select the Project Start Date and Project End Date fields.

Notice that the text in the title bar of the Property Sheet changes to Multi-Select. This indicates that there is more than one component selected in the Design Window. Also, notice that the properties you can change are fewer when you have multiple elements selected; only the properties that they have in common are available.


2. In the Design Window, drag the fields to the left, so that there is only a single space between the Project ID field and the Project Description field.

Note: While dragging the cursor, it may change to a  symbol indicating that the fields cannot be dropped where you have positioned them. Move the mouse down a bit and the cursor should return to normal.

The Design Window should look like this:



The screenshot shows a software interface titled "Edit Project". At the top, there are two fields: "Date . : 66666666" and "Time . : 66666666". Below these is a text box containing "Type options, press Enter." and "3=Copy 4=Delete". The main area is a table with four columns: "ID", "Description", "Start Date", and "End Date". The first row of data shows "ID" with a small square icon, "Description" with a long horizontal bar, "Start Date" with "9999999", and "End Date" with "9999999". At the bottom, there is a status bar with keyboard shortcuts: "F3=Close F5=Refresh F6=Add F10=Update F17=Filter...".

ID	Description	Start Date	End Date
		9999999	9999999

3. Repeat this process to move the Project End Date field so that it has only a single space between it and the Project Start Date field.
4. Keep the Panel Designer open for the next section.

Adding Static Text

Later you will add the ability to assign tasks to projects. In preparation for this stage, you next add static text to your panel that tells the end users how to get to the Add Tasks panel.

To add static text:

1. In the Panel Palette, right-click Selector text, and choose Create Static from the pop-up menu. The Create Static dialog appears:



2. Enter 5=Work with tasks in the Name field. Then click OK.
3. Select the new static item under the Selector text region in the Panel Palette.
4. Notice that, in the Design Window, the selected static text is behind the text at the top of the region. Drag the static text down and to the right so that it is to the right of 4=Delete.
5. Your panel should look something like this:

Date . : 66666666 Edit Project Time . : 66666666

Type options, press Enter.

3=Copy 4=Delete 5=Work with tasks

ID	Description	Start Date	End Date
1		9999999	9999999

F3=Close F5=Refresh F6=Add F10=Update F17=Filter...

6. Close the panel design and save your changes.

Generating and Building

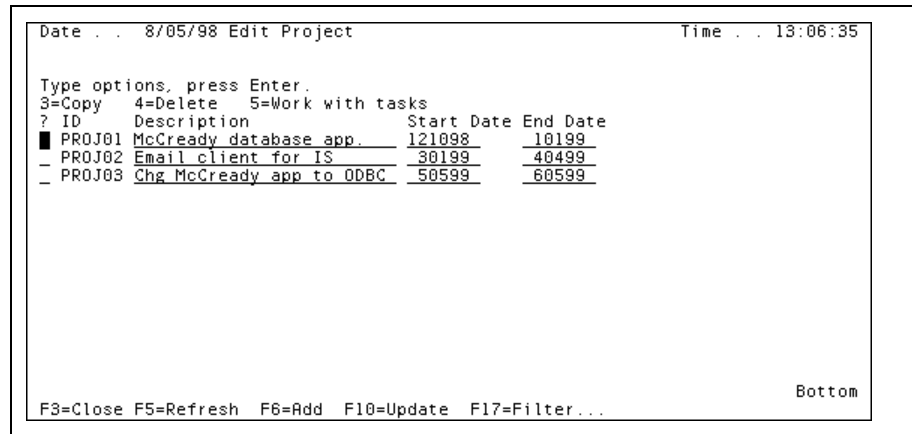
Now that you have modified the panel that lists all of the projects, you are ready to regenerate and rebuild the function it is scoped to. For more information on generating and building instructions, see the Generating and Building section in the chapter "Defining Entities."

Because you only modified the panel scoped to Project.Grid Maintained Entity.Edit Grid User Interface, you do not need to generate the entire project. Open the Generate and Build window, and select Project.Grid Maintained Entity.Edit Grid User Interface. Then, generate and build this function. Doing so generates and builds both the function and the panel scoped to the function.

To see the changes you made:

1. Run the Edit grid user interface function.

The panel should look like this:



ID	Description	Start Date	End Date
PROJ01	McCready database app.	121098	10199
PROJ02	Email client for IS	30199	40499
PROJ03	Chg McCready app to ODBC	50599	60599

F3=Close F5=Refresh F6=Add F10=Update F17=Filter... Bottom

2. Exit the application.

Chapter Review

In this chapter, you:

- Learned about the three windows that comprise the Panel Designer
- Changed panel formatting to distinguish key values from non-key values
- Changed field labels
- Moved fields in a region so that the panel looks better
- Added static text to a region

Chapter 4: Diagramming Entity Relationships

In this chapter, you use the CA Plex diagramming tool to create an Entity Attributes diagram. In the diagram, you define the other two entities in your model: Employee and Task. You also define Employee's attributes and the relationships among all three entities.

This section contains the following topics:

[Introducing the Diagrammer](#) (see page 61)

[Creating a Diagram](#) (see page 63)

[Defining Entities with the Diagrammer](#) (see page 66)

[Defining Attributes with the Diagrammer](#) (see page 67)

[Defining Relationships Between Entities](#) (see page 68)

[Defining Employee and Its Attributes](#) (see page 71)

[Refers to Relationships](#) (see page 74)

[Modifying Panels](#) (see page 75)

[Generating and Building the Employee Entity](#) (see page 77)

[Testing the Application](#) (see page 78)

[Chapter Review](#) (see page 81)

Introducing the Diagrammer

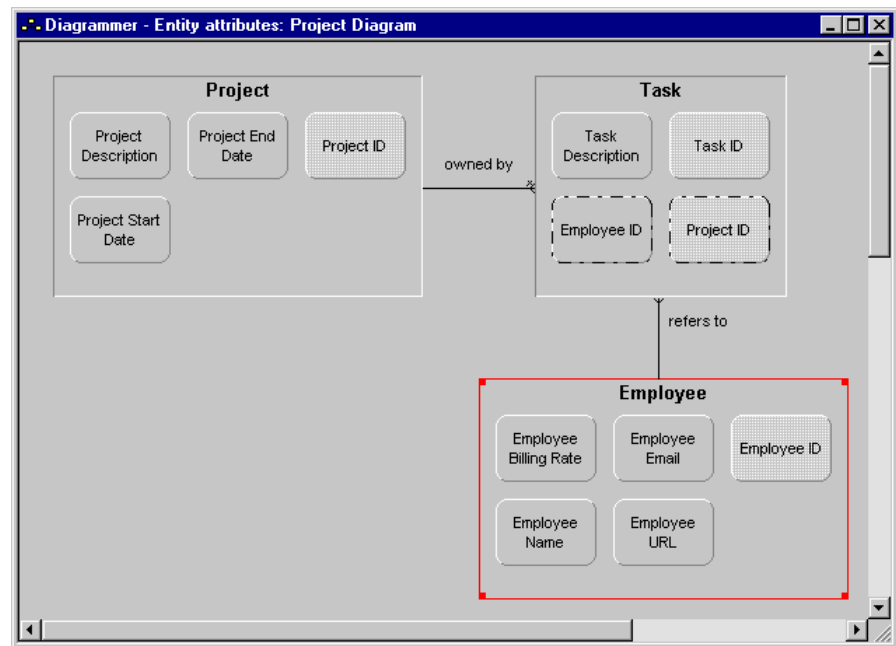
Most of the time, the first step in creating a database application is creating a data model. In the data model, you specify how real-world entities are represented in a database.

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. Each of these is represented by one entity.

Note: Entity Attributes (EA) diagrams in CA Plex are a little bit different than they are in other applications. EA diagrams typically only display entities and their attributes, but in CA Plex, relationships between entities are also displayed.

In CA Plex, EA diagrams provide a visual representation of the entities that are stored in a database, their attributes, and the relationships between the entities. Diagrams enable you to visually communicate the specifics of an application to other developers or business analysts, and in some cases, to your end users.

The following example shows what your EA diagram will look like when you finish this chapter:



In diagrams, objects are represented by rectangles, rounded rectangles, and ovals. These objects are 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, you will create two objects:

- Rectangles that represent entities
- Rounded rectangles that represent attributes of those entities

Together, two nodes, and the link between them, or a node and its sub-node, are equivalent to a triple in an CA Plex model. For example, you can see in the preceding graphic that the Project entity has four sub-nodes: Project ID, Project Description, Project Start Date, and Project End Date. These correspond to the following triples, which you created in the chapter "Defining Entities:"

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

When you define and change objects using the Diagrammer, your changes are reflected everywhere in the model (just as if you had used the Model Editor or another editor to make the changes). Similarly, if you delete a triple in the Model Editor, any corresponding links on diagrams are also removed. But the opposite is *not* true. If you delete a link using the Diagrammer, the triples created when you created the link are *not* deleted.

Choosing Between Editors

For much of the data modeling stage of your application development, you can use either the Diagrammer or the Model Editor because they both create the triples that define your entities, the attributes of your entities, and the relationships between entities. You can use the editor of your choice to create this part of your model.

For the purpose of this tutorial, however, use the Diagrammer to define the remaining two entities, their attributes, and the relationships among all of the entities.

Creating a Diagram

CA Plex supports many types of diagrams (including diagram types that you can make yourself), but this tutorial only shows you how to create an Entity Attributes diagram.

In the diagram you create in this chapter, you:

- Add the Project entity, which you already defined
- Define the Task and Employee entities

To create an Entity Attributes diagram

1. Open the Model Editor.
2. Enter the following triple:

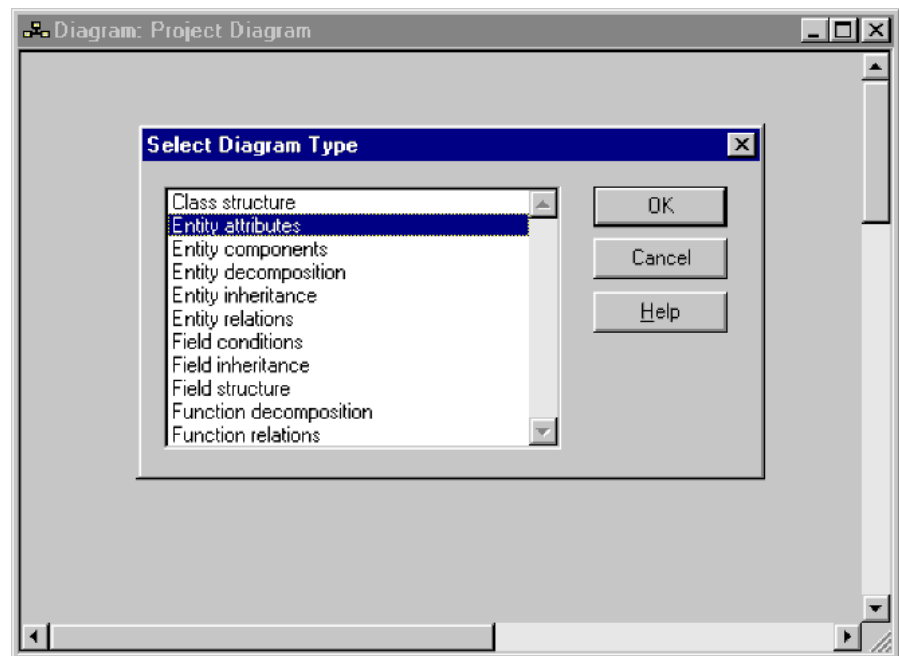
Project **described by** Project Diagram

This creates the diagram object. Remember, you can drag the Project entity from the Object Browser to the Model Editor, rather than typing it in manually.



3. In the Object Browser, select Diagram from the object type list.
4. Select Project Diagram from the list, and click the Editor button on the Object Browser.

The diagram opens with the Select Diagram Type dialog on top:



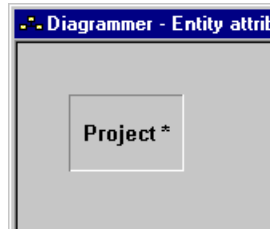
5. Select Entity Attributes and click OK.

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



6. In the Object Browser, click the Entity button to display entities.
7. Drag the Project entity from the Object Browser to the upper-left corner of the diagram.

Notice that the diagram creates a node for the Project entity. Also notice that it has an asterisk (*) after the name:



The asterisk indicates that the object has attributes that the diagram can show.

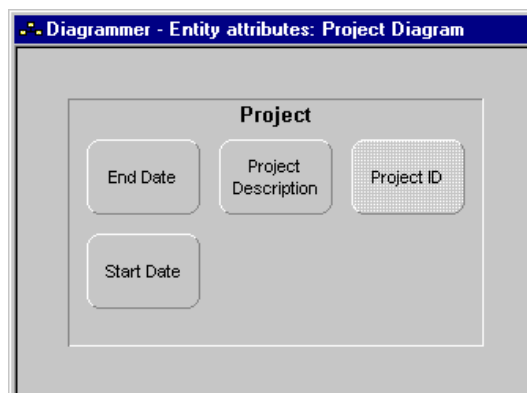
Next, you will change a diagram setting so that it shows the attributes of all entities represented by it.

8. From the Layout menu, choose Always Show Sub-Nodes.



9. Click the Refresh button.

The diagram now displays the attributes you defined for the Project entity in the chapter “Defining Entities.”



Notice that the node automatically resized to show all of its attributes.

The Project ID sub-node has a lighter color to indicate that it is a key. You will see later in this chapter how the Diagrammer uses display characteristics to indicate foreign keys and other special attributes.

Defining Entities with the Diagrammer

You already defined Project, the first of three entities for the application you are building. For more information, see Defining the Project Entity in the chapter "Defining Entities." In this section, you use the Diagrammer to define the Employee and Task entities.

To define the Employee and Task entities

1. If you closed the diagram, reopen it.
2. Right-click the upper-right corner of the diagram, and choose Create Node, and then Entity from the pop-up menu. The New Node dialog appears.

Note: The caption on the dialog is Entity. The caption on this dialog shows you the type of object you are defining.



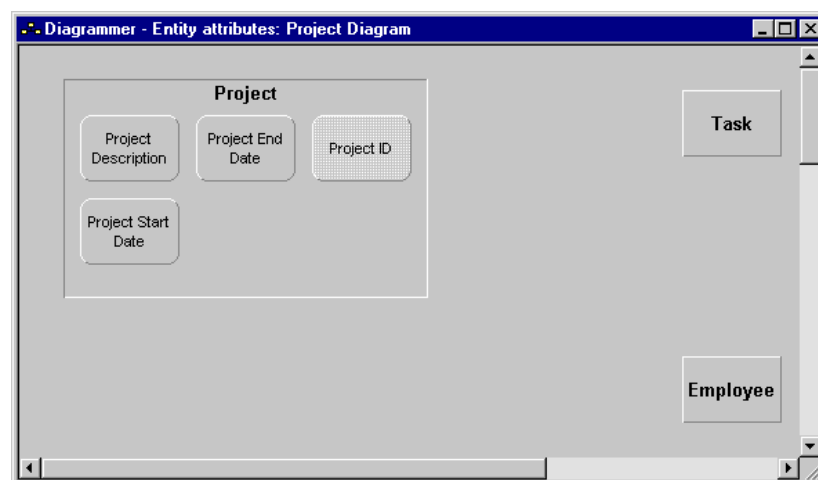
Note: When you create a node by right-clicking in the diagram, the node appears where you click the mouse.

3. Type **Task** and click OK.

Note: The Task node appears where you clicked your mouse.

4. Move your mouse to the lower-right corner of the diagram and repeat Steps 2 and 3 to create a node for the Employee entity.

Your diagram should look something like this:



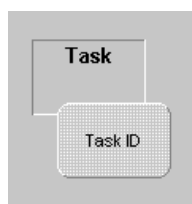
Defining Attributes with the Diagrammer

The Task and Employee nodes do not have asterisks, as the Project entity did. This shows that these entities have no attributes, yet. In this section, you use the Diagrammer to define a primary key and non-key attributes for both entities.

To define attributes for Task and Employee:

1. Right-click the Task node, and choose Create Sub-Node, and then Primary Key from the pop-up menu.
2. In the New Node dialog, type **TASK ID** and click OK.

The Diagrammer creates the sub-node:



3. Notice that the sub-node is not contained within the Task node. Right-click the Task node and choose AutoLayout from the pop-up menu.

The AutoLayout function enlarges parent nodes so that they can hold all of their sub-nodes. It also organizes the sub-nodes within the parent node. In this case, it enlarges the Task node, and puts the Task ID node inside it.

Note: When right-clicking entity nodes, make sure you are clicking the parent node, and not any of its sub-nodes.

4. Right-click the Task node, choose Create Sub-Node, and then Attribute from the pop-up menu.
5. Name the sub-node **Task Description**.
6. For the Employee entity, repeat Steps 1 and 2 to define a primary key called Employee ID, and Steps 4 and 5 to define the following non-key attributes:

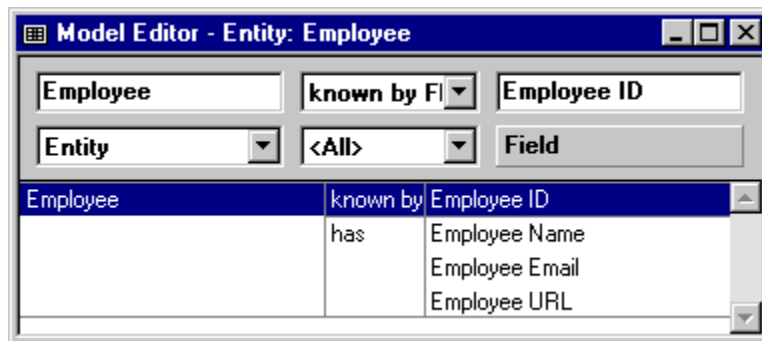
- Employee Name
 - Employee Billing Rate
 - Employee Email
 - Employee URL
7. Select the Employee and Task nodes and use Auto Layout. Make sure that each entity is in a corner of the Diagrammer window.



8. Open the Model Editor.

Note: If it is already open, click the Clear Focus toolbar button:

Equation 2: PLEX--Defining Attributes with the Diagrammer



Defining entities and their attributes in the Diagrammer created corresponding triples in the model. In the next section, you define relationships between the entities (which also creates triples).

Defining Relationships Between Entities

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 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.

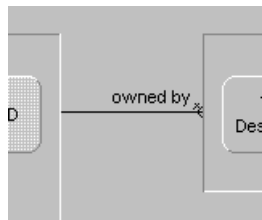
When you define a task for a project, you must specify an employee who is responsible for completing the task. Because an employee can be assigned to more than one task, in more than one project, you do not want to delete the employee record if you delete a task. So, you define a **refers to** relationship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

Defining an Owned By Relationship


To define an Owned By relationship:

1. If you closed the diagram, open it again.
2. Select the Task entity.
3. Hold down the Control key and right-click the Project entity.
4. From the pop-up menu, choose Create Link, and then Owned By.

An owned by link is created between the two entities:



Note: The link shows a one-to-many relationship between the two entities. This indicates that a project can have many tasks, but that a task can belong to only one entity.

5. If the label *owned by* is not visible, you can turn on label display by clicking the View menu, Display Link Labels, Link Names.
6. If the link label appears on top of one of the nodes, click the label to select it (the cursor looks like this  when a link label is selected), and drag the label to a more appropriate location.

Next, you will focus the Model Editor on the Task entity, which shows that a triple was added to your model when you created the **owned by** relationship between Task and Project.

7. Set the Object Browser to display entities. If you do not see the Task entity, refresh the Object Browser.
8. Drag the Task entity from the Object Browser to the bottom section of the Model Editor. This shows all of the triples associated with the Task entity.

Notice that creating the relationship between the Project and Task entities created the triple Task **owned by** Project.



9. Click the Clear Focus toolbar button. Clearing the focus in the Model Editor causes the editor to show all triples in the model.

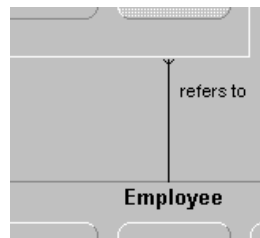
Next, you will define the **refers to** relationship between the Task and Employee entities.

Defining a Refers To Relationship

To define a Refers To relationship:

1. In the Diagrammer, select the Task node.
2. Hold down the Control key and right-click the Employee entity.
3. From the pop-up menu, choose Create Link, and then Refers To.

A refers to link is created between the two entities:



Note: This creates a many-to-one relationship, indicating that an employee can be assigned to many tasks, but that a task can have only one employee assigned to it.

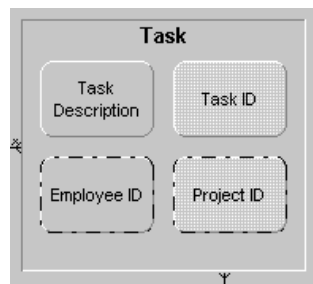
4. Refresh the Model Editor.

Note: Creating the relationship between the Task and Employee entities created the triple Task **refers to** Employee.

You have now defined relationships between the Project, Task, and Employee entities. Defining these relationships caused the primary keys of the Employee and Project entities to become foreign keys of the Task entity. Next, you change the diagram to display these foreign keys.

5. In the Diagrammer, select the Task entity.
6. Right-click the entity, choose Resolved Keys, and then Show.

Your diagram changes to display the Employee ID and Project ID attributes in the Task entity.



Note: The two foreign keys have dashed borders. The Project ID node is displayed in the same color as the Task ID node, as it becomes part of the Task entity's key (because it results from an **owned by** relationship). This type of attribute is usually called a *parent key* because it is the key of the parent entity. The Employee ID node is darker because it is not part of the Task entity's key.

7. Close the Diagrammer. Click Yes to save your changes.

Defining Employee and Its Attributes

Using the Diagrammer, you have defined the Task and Employee entities. You will complete the definition of the Task entity in the next chapter, "Defining Owned By Relationships." For the remainder of this lesson, you continue defining the Employee entity. After that, you generate and build the Employee entity, and then test that part of your application.

Defining the Employee Attributes

The next step in defining Employee is to add triples for Employee's attributes to inherit structure and functionality from class library fields.

1. In the Model Editor, add the following triple:

Employee ID **is a** OBASE/Code

Note: The **is a** verb you use in this step is different from the one that you used in the previous section. There are several verbs that have the same name, but which have a different target object. Make sure to select the verb that matches the target object (in this step, the **is a FLD** verb).

For more information on OBASE/Code, see To Define the Properties of Project's Fields in this chapter.

2. Add the following inheritance triples:

Employee Name **is a** OBASE/Narrative

Employee Email **is a** OBASE/Narrative

Employee URL **is a** OBASE/Narrative

Inheriting from OBASE/Narrative gives each of these fields a length of 25.

These fields may require more than 25 characters to store the whole name, email address, or URL. One of the powerful aspects of inheriting from class libraries, though, is that you can change the default values.

3. Add the following triples to increase the field lengths (the numbers are displayed in red because they are literal values):

Employee Name **length 40**

Employee Email **length 50**

Employee URL **length 50**

By adding these three triples, you effectively changed the database schema, the user interface, and all of the processing for these values to use a different length for each field.

4. Add the following triple:

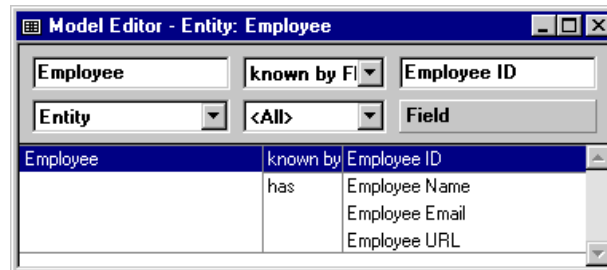
Employee Billing Rate **is a** OBASE/Price

Inheriting from Price causes Employee Billing Rate to store a number with two decimal places.

Defining the Employee Entity

The next step in defining the Employee entity is to add triples for Employee to inherit a user interface from class library entities.

1. Change the Object Browser to show entities again, and click Refresh.
2. Drag the Employee entity from the Object Browser to the body of the Model Editor. The Model Editor should look like this:

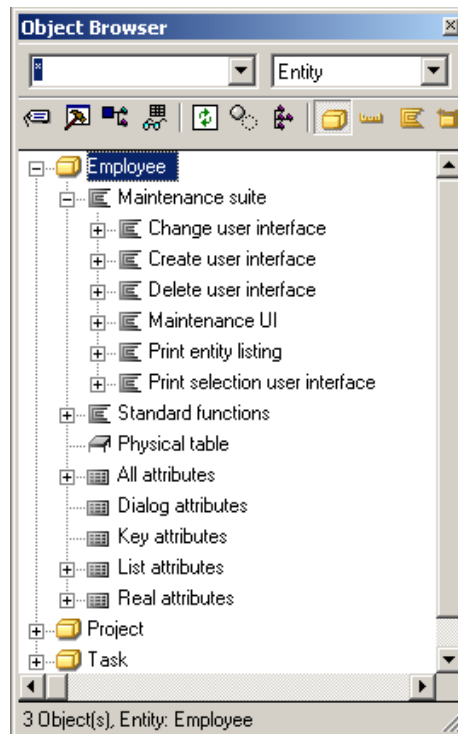


3. Add the following triple:

Employee is a OBASE/User Maintained Entity

This inheritance triple provides Employee with user interface functionality from User Maintained Entity in the OBASE class library.

4. Refresh the Object Browser, and then expand the Employee entity until the Object Browser looks similar to the following example.



Notice that Employee inherited the function Maintenance Suite, which has the functions Change User Interface, Create User Interface, Delete User Interface, and Maintenance UI scoped to it, each of which scopes a panel object. These four functions display panels on which end users can add, change, and delete Employee data. You will see how the functions interact when you generate and build the Employee entity at the end of this chapter. (You will not use the two print functions in this tutorial.)

5. Add the following triple:

Employee is a OBASE/Referenced Entity

This gives Employee functionality that enables end users to select from a list of employees to assign to a task. This is covered in more detail in the following section.

The inheritance triples you created in the previous steps gave the Employee entity the following:

- Functions that let you add, change, and delete employees
- Ability to access a list of employees from the panel that lists tasks

Refers to Relationships

A **refers to** relationship indicates that an instance of one entity stores the key value of one or more instances of another entity. In the application you are building, this is represented by assigning employees to tasks; the task stores the primary key value of the employee. 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.

A **refers to** relationship differs from an **owned by** relationship in that you do not want employee records to be deleted when a task is deleted, as an employee can be assigned to other tasks. However, a task *should* be deleted when its owning project is deleted, as a task is merely a subset of a project.

Adding Prompt Processing

In this chapter, you added prompt processing by creating an inheritance triple, Employee is a OBASE/Referenced entity. This functionality enables end users to display a list of employees by pressing F4 while the cursor is in the Employee ID field on the panel that displays tasks. The list is displayed in a separate panel (the following example shows the panel populated with data you enter at the end of this section):

Date . . . 7/30/98 List Employee		Time . . . 14:36:57
Type options, press Enter.		
1=Select		
? ID	Name	Billing Rate
- EMP02	David Castro	32.00
- EMP03	Martha Kolman	55.00
F3=Close F5=Refresh F17=Filter...		Bottom

End users can select an employee from the list, which puts the corresponding ID in the Employee ID field on the Task panel. This keeps your end users from having to memorize potentially cryptic Employee ID values (such as Emp03), instead letting them pick by name, email address, or some other criterion.

You will be able to use this new functionality at the end of the next section.

Modifying Panels

In this section, you learn how to use the Panel Designer to make another type of change. You change Employee.Maintenance Suite.Maintenance UI.List Panel and Employee.Select Instance User Interface.List Panel so that they only display a few of Employee's fields.

Hiding Fields on Panels

Because the 5250 screen can only display 80 characters horizontally in its low-resolution mode, you cannot display all of Employee's attributes on a single line. You could move a couple of the fields to a second area on the screen, but because users only need enough information to identify the employee they want to modify or delete, you change the display so that the Employee Email and Employee URL fields do not appear.

Because users typically use either the Employee ID or Employee Name to identify the individual, and may use the Employee Billing Rate to decide whom to assign to a task, you will leave these fields visible.

To change the panel that lists Employees:

1. Open Employee.Maintenance Suite.Maintenance UI.List Panel.
2. In the Panel Palette, expand List Employee, and then Grid.
3. Right-click the Employee Email field. From the pop-up menu, choose Visible, and No.
4. Repeat this Step 3 to set the Employee URL field so that it does not display.
Notice that the fields no longer appear on the panel in the Display Window.
5. Edit the labels for the columns to remove the word *Employee* from each. For more information, see the section
6. Change an Individual Control. For more information, see the chapter "Modifying the User Interface."
7. Move the fields so that there is only a single space between them. For more information, see Change Multiple Controls in the chapter "Modifying the User Interface."

The Design Window should look like this:

```

Date . : 66666666 Edit Employee                               Time . : 66666666

```

[illegible]

8. Close the panel. Click Yes to save your changes.

The Select Instance User Interface panel has the same problem as the main Employee panel. Open Employee.Select Instance User Interface.List Panel and hide the Employee Email and Employee URL fields, change its column labels, and move the fields next to each other, as you did in the preceding steps.

When you are done, the Design Window should look like this:

[illegible]

Generating and Building the Employee Entity

You have now:

- Defined the Employee entity and its attributes
- Changed the lengths of the Employee Name, Employee Email, and Employee URL fields
- Fixed problems in the panels

You are now ready to generate and build the Employee entity. For more information on the process of generating and building, see the Generate and Build the Project Entity section in the chapter "Defining Entities."

Open the Generate and Build window and select the Employee entity. Then, generate and build this entity and all of its scoped objects.

This function lists all of the employees in your database. Because you inherited attributes from User Maintained Entity, rather than Grid Maintained Entity, you have a group of functions for maintaining your Employee entity, rather than a single function, as you did with the Project entity.

The Edit Employee panel should look like this:

```
Date . . . 7/29/98 Edit Employee                               Time . . . 15:23:40

Type options, press Enter.
2=Change 3=Copy 4=Delete
? ID      Name                                           Billing Rate

F3=Close F5=Refresh F6=Add F17=Filter... F22=Print
```

Adding Employees

To add employees to the entity:

1. Press F6. The Add Employee panel appears. This panel is scoped by Employee.Maintenance Suite.Create User Interface.

Date . . . 7/29/98 Add Employee
Time . . . 15:29:48

Employee ID
Employee Name
Employee Billing Rate00
Employee Email
Employee URL

F3=Close F5=Refresh

2. Add the values for the following employees, pressing Enter between each:

Value	Enter
Employee ID	EMP01
Employee Name	Randy Johal
Employee Billing Rate	\$45.00
Employee Email	rjohal@anonymcorp.com
Employee URL	http://www.anonymcorp.com/~rjohal

Value	Enter
Employee ID	EMP02
Employee Name	David Castro
Employee Billing Rate	\$32.00
Employee Email	dcastro@anonymcorp.com
Employee URL	http://www.anonymcorp.com/~dcastrol

Value	Enter
Employee ID	EMP03

Value	Enter
Employee Name	Martha Kolman
Employee Billing Rate	\$55.00
Employee Email	mkolman@anonymcorp.com
Employee URL	http://www.anonymcorp.com/~mkolman

- After you add these employees, press F3 to close the panel.
- Add the following triple to your model to keep the data you 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 "Defining Entities."

Deleting Employees

To delete an employee from the entity:

- On the Edit Employee panel, navigate to the Grid Selector column for Randy Johal.
- In the selector field, type **4**, then press Enter.

The Display Employee panel appears:

Display Employee

Time . . 16:19:40

Date . . 7/29/98
Employee ID EMP01
Employee Name Randy Johal
Employee Billing Rate 45.00
Employee Email rjohal@anonymcorp.com
Employee URL http://www.anonymcorp.com/~rjohal

F3=Close F13=Cancel All

This panel is scoped to Employee.Maintenance Suite.Delete User Interface.

- Press Enter again.

Changing an Employee

To change information about an employee:



1. On the Edit Employee panel, navigate to the Grid Selector column for the record you want to change.
2. In the selector field, type **2**, then press Enter. The Change Employee panel appears.

Date . . .	7/29/98	Change Employee	Time . . .	16:14:28
Employee ID	EMP01			
Employee Name	Randy Johal			
Employee Billing Rate . . .	45.00			
Employee Email	r.johal@anonymcorp.com			
Employee URL	http://www.anonymcorp.com/~r.johal			
F3=Close F5=Refresh F13=Cancel All				

3. Navigate to the field you want to change.
4. Overwrite the contents of the field, then press Enter.
5. Exit the application.

Chapter Review

In this chapter, you:

- Used the Diagrammer to define the Task and Employee entities and their attributes
- Added triples so that the Employee entity inherits from two more class library entities: OBASE/Referenced Entity and OBASE/User Maintained Entity
- Added triples so that the attributes you defined for the Employee entity inherit from one more pattern field, OBASE/Price
- Tested the Employee entity user interface function by creating employees
- Hide fields on panels so that only pertinent information is displayed

Chapter 5: Defining Owned By Relationships

In the chapter, "Diagramming Entity Relationships," you created the Task and Employee entities. Then you specified relationships between them: Project owns Task and Task refers to Employee. In this chapter, you add inheritance triples for the Task entity, and then add processing that enables end users to add tasks to a project and to assign employees to those tasks.

This section contains the following topics:

[Defining Task and Its Attributes](#) (see page 83)

[Owned By Relationships](#) (see page 85)

[Defining Restrictor Processing](#) (see page 86)

[Introducing the Action Diagrammer](#) (see page 89)

[Generating and Building the Task Entity and Project.Grid Maintenance Suite.Edit Grid](#)

[User Interface](#) (see page 97)

[Testing the Application](#) (see page 97)

[Chapter Review](#) (see page 100)

[Helpful Hints](#) (see page 101)

Defining Task and Its Attributes

You defined Task in the Diagrammer, but it does not have any functionality yet. In this section, you further define the Task entity.

Defining the Attributes for the Task Entity

The first step in defining attributes for the Task entity is to add triples for the attributes to inherit structure and functionality from class library objects.

To specify the properties of Task's attributes:

1. If the Model Editor and Object Browser are not open, open them.
2. Add the following triples:

Task ID **is a** OBASE/Code

Task Description **is a** OBASE/Narrative

These are the same objects that the Project ID and Project Description fields inherited from.

Defining the Task Entity

You define the Task entity by adding triples so that Task inherits a user interface and database functionality from class library objects.

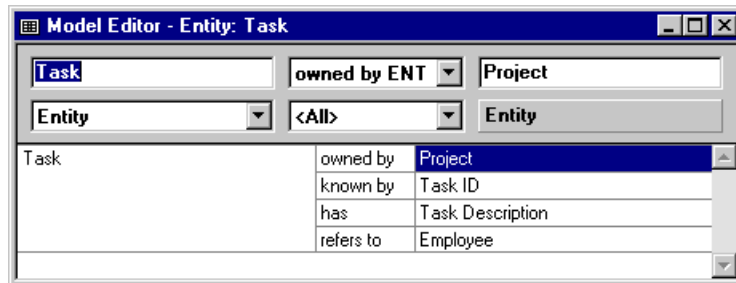
To define the Task entity:



1. Refresh both the Model Editor and Object Browser, and click the Entity button on the Object Browser if entities are not already showing.

Notice that the Task and Employee entities are both visible in the Object Browser.

2. Drag the Task entity from the Object Browser to the body of the Model Editor. The Model Editor should look like this:



These are the triples that CA Plex added to your model when you created the diagram in the preceding lesson.

3. Add the following triples:

Task is a OBASE/Grid Maintained Entity

You created this same triple for the Project entity. For more information, see [Add Functionality to the Project Entity](#) in the chapter "Defining Entities." It gives Task basic user interface and database functionality.

4. Add the following triple:

OBASE/Child

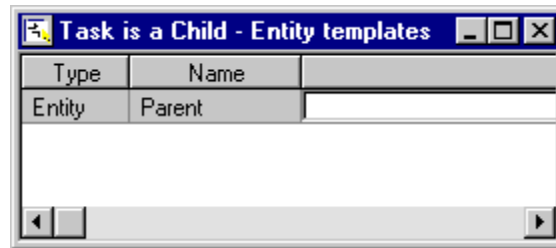
This triple gives Task special processing based on its relationship with its owning entity, Project. You used the Diagrammer to define the triple, Task **owned by** Project, which describes the relationship between the entities. For more information, see [Defining Relationships Between Entities](#) in the chapter "Diagramming Entity Relationships." The Task **is a OBASE/Child** inheritance triple provides the Task entity with cascade delete functionality. This functionality causes all tasks owned by a project to be deleted when the owning project is deleted.

5. Select the Task **is a OBASE/Child** triple in the Model Editor by clicking in the center column.



- Click the Editor button on the toolbar. The Template Editor appears.

You use this dialog to specify the owner of an entity that inherits from OBASE/Child.



- Click in the right-most column, and enter **Project**.
- Close the Template Editor. Click Yes to save your changes.
- Refresh the Model Editor. Notice that CA Plex added the following triple:

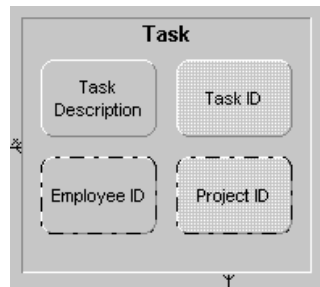
Task **replaces** OBASE/Parent
 ...by Project

This reflects the change you made in the Template Editor. The second line (...by Project) is called a *continuation triple*.

The indentation and ellipsis before the continuation verb indicate that the triple qualifies or modifies the triple above it.

Owned By Relationships

Remember, the Task entity in the diagram you created in the chapter "Diagramming Entity Relationships," had two foreign keys, Employee ID and Project ID:



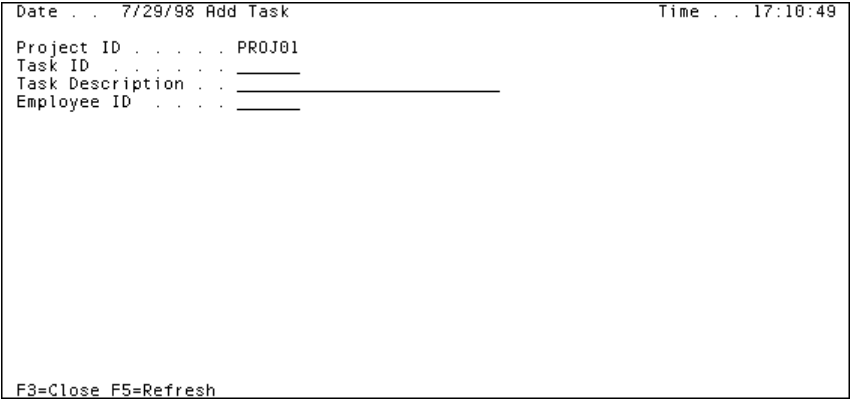
You can tell by the shading that Project ID is part of the key for Task (with Task ID). This field was added to the Task entity when you created the triple Task **owned by** Project. It was added because it is the key of the owning entity.

The addition of the parent key field means that when creating or accessing a task, the owning project must be specified.

How does this affect your application? Well, it means that you must provide a way for end users to specify the owning project when creating or accessing a task. To this end, you will design your application so that end users can only create and access instances of the child entity (Task) after selecting an instance of the parent entity (Project).

In your application, you will add a key code to the Edit Project panel. Remember, you already added the static text for that key code. For more information, see Add Static Text in the chapter "Modifying the User Interface."

You will add functionality so that when end users open the Edit Project panel and enter the key code to work with tasks for a project, the Edit Task panel appears. When the end users add tasks, the Project ID field is automatically populated with the ID of the project selected in the Edit Project panel:



```
Date . . . 7/29/98 Add Task                                     Time . . . 17:10:49
Project ID . . . . . PROJ01
Task ID . . . . . 
Task Description . . . 
Employee ID . . . . . 
F3=Close F5=Refresh
```

Defining Restrictor Processing

When an end user opens the Edit Task panel from the Edit Project panel, the Edit Task panel should not list all of the tasks for all of the projects, as this would be confusing. It is better to show only the tasks that were created for a specific project. The functionality that you set up to accomplish this is called restrictor processing, because it restricts database access functions so they only display the instances associated with a particular key value, in this case the primary key of the project the end user selected.

Next, you add a key code to the Edit Project panel. You then add functionality so that when an end user enters that key code, the project calls Task.Grid Maintenance Suite.Edit Grid User Interface, the function that displays the Edit Task panel.

Creating a Key Code on a Panel

Adding a key code to a panel involves three steps:

1. Adding a static to the panel to tell the end users what the key code is.
2. Creating a *logical event* on the panel.
3. Adding a list value to the grid selector.

You already added the static text telling the end user what key code to use to open the Edit Task panel. For more information, see Add Static Text in the chapter "Modifying the User Interface." In the following steps, you create the logical event that causes the panel to appear, and then you create a grid selector value for the panel.

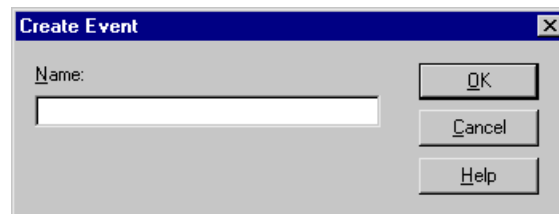
Creating a Logical Event

A logical event is a construct you define to specify what happens in response to physical events, such as when the end user presses a function key or enters a subfile option. You can map a logical event to more than one physical event. For example, you can map a logical event called ClosePanel to two physical events: pressing Enter or pressing F3.

To add a logical event to the Edit Project panel:

1. Open Project.Grid Maintenance Suite.Edit Grid User Interface.List Panel.
2. Expand the Panel Palette so that you can see the Events folder.
3. Right-click the Events folder, and choose Create Event from the pop-up menu.

The Create Event dialog appears:



4. Enter **Modify Tasks** and press OK.

You have now created a logical event. Later, you will add functionality to the Project.Grid Maintenance Suite.Edit Grid User Interface function to tell it what to do when the Modify Tasks logical event is triggered.

5. Leave the panel open so that you can perform the following tasks.

Creating a Subfile Option

You create subfile options so end users can enter a value in the subfile option field on the panel (the left-most field on the Edit Project panel) to perform an action (in this case, open the Edit Task panel).

The following graphic shows you the Edit Project panel with a 5 in the subfile option field for the first project.

```

Date . . . 8/05/98 Edit Project                               Time . . . 16:42:44

Type options, press Enter.
3=Copy   4=Delete   5=Work with tasks
? ID      Description                               Start Date End Date
5 PROJ01  McCready database app.                   121098    10199
PROJ02  Email client for IS                         30199    40499
PROJ03  Chg McCready app to ODBC                   50599    60599

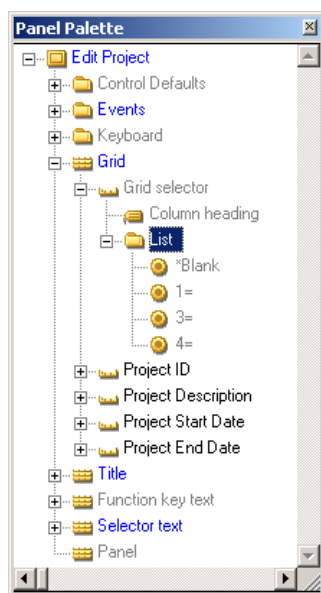
F3=Close F5=Refresh F6=Add F10=Update F17=Filter...          Bottom

```

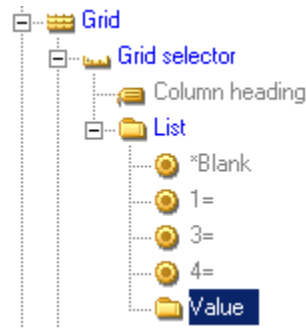
Note: CA Plex uses the term *grid selector* for the System i subfile option.

To create a subfile Option value:

1. In the Panel Palette, expand the Grid region, then the Grid Selector field, and finally, the List folder. The Panel Palette should look like this:



2. Right-click the List folder, and choose Create Value from the pop-up menu.
3. A subfile option value, called Value, is added under the List folder. Select the new value:



4. In the Property Sheet, specify the following settings:

Value	Enter
Text	5=
Event	Modify Tasks
Value	Five

5. Close the panel and save your changes.

Introducing the Action Diagrammer

Now that you have defined an event and specified the condition that triggers that event (entering the value 5 in the grid selector field), you can define the processing that occurs in response. To do this, you modify the Project.Grid Maintenance Suite.Edit Grid User Interface function. In CA Plex, the editor you use to edit a function is called the Action Diagrammer.

Action Diagrammer Windows

Use the Action Diagrammer to create, edit, and display functions. The editor consists of three parts:

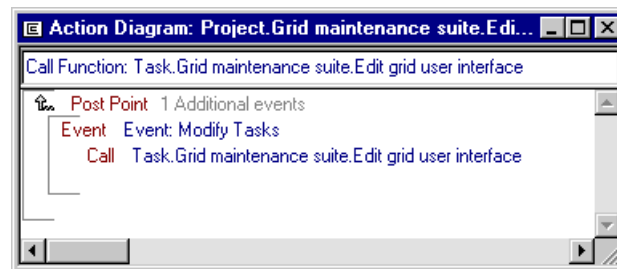
Action Diagram window—Enter instructions and comments using this window

Action Diagram palette—Select Action Diagram syntax from this palette

Variable palette—Select the variables created for the function from this palette

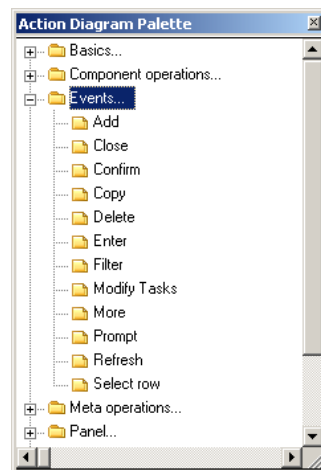
Action Diagram Windows

The Action Diagram window displays the function logic (the following graphic shows what your action diagram looks like when you are complete this chapter). The Action Diagram window has an input line at the top, where you enter new instructions. Specify where to add the new logic in the body of the window.



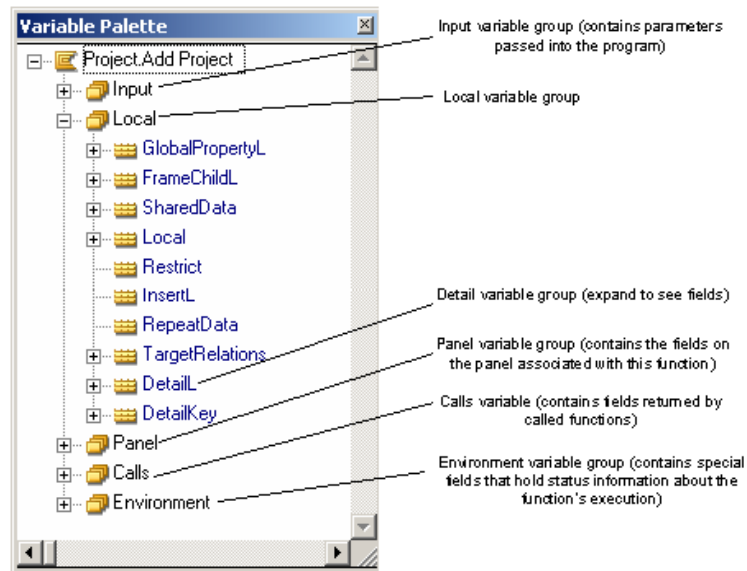
Action Diagram Palette

The Action Diagram Palette holds the various syntax elements available for you to use to create the function logic. It is organized according to the type of action they perform.



Variable Palette

The Variable Palette is used to view and edit the variables available to a function. In this window, you can see both real and inherited variables, and also add fields to them.



Basic Concepts

Instructions on using the Action Diagrammer to its full potential are beyond the scope of this tutorial, but understanding the following concepts will be helpful when using this editor.

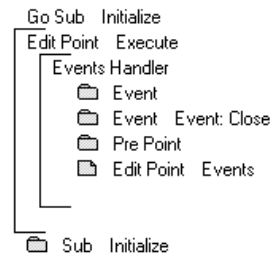
Functions

Functions are objects you create to 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
- Interface with external procedures and programs

Constructs

Constructs are the vertical brackets that enclose instructions or other constructs. They correspond roughly to the levels of indentation that you typically use with third generation languages, such as C++. You can create complex structures with constructs. The following diagram shows you two constructs, one embedded in another.



Statements

Statements occupy one or more lines of the action diagram and cause an action or set of actions to occur. For example, the Call statement calls a function from another function.

Collection and Edit Points

When you modify or add to inherited functions, you can only make changes and additions in certain places, called collection and edit points. These are places that the ancestor function indicated are appropriate for new instructions.

Events Handlers

Each function that has a panel scoped to it has an Events Handler. This is a type of switchboard that processes all of the events a function recognizes. It calls various subroutines in response to the events that are triggered (such as the Modify Tasks event you created earlier in this chapter). Later, you will add processing to support the Modify Tasks event to the Events Handler in the Project.Grid Maintenance Suite.Edit Grid User Interface action diagram.

Adding Functionality to Logical Events

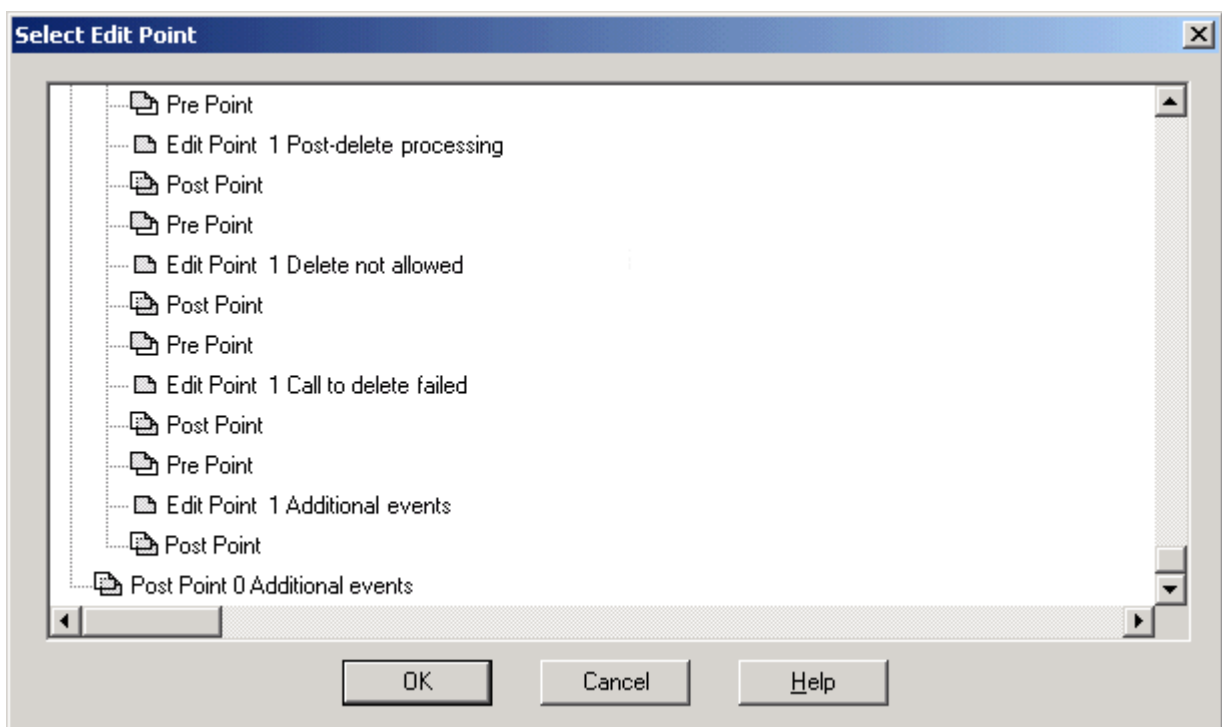
Next, you will modify Project.Grid Maintenance Suite.Edit Grid User Interface so that when the Modify Tasks logical event is triggered, the function calls the Task.Grid Maintenance Suite.Edit Grid User Interface function.

Adding Functionality

1. In the Object Browser, right-click Project.Grid Maintenance Suite.Edit Grid User Interface and choose Editor.

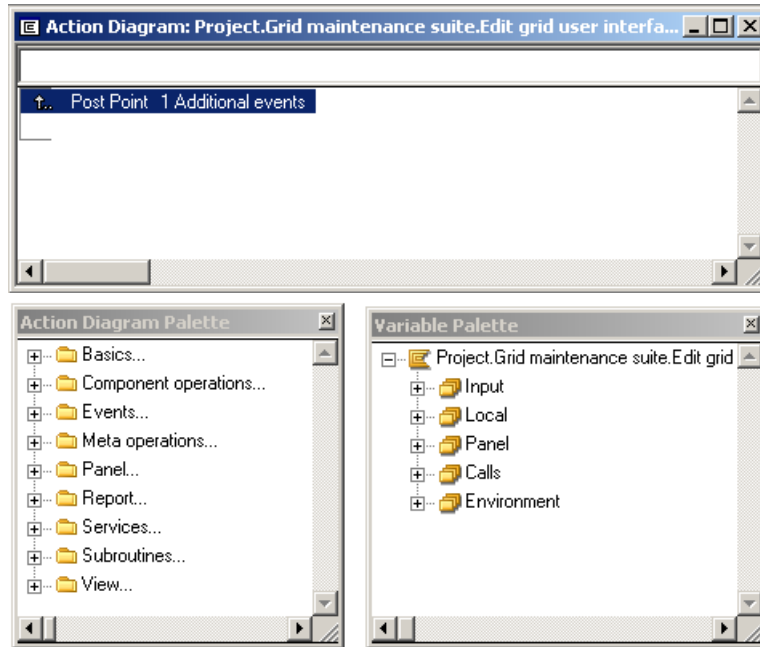
The action diagram opens, with the Select Edit Point dialog over it. Before you can make a change or addition to a function, you must specify where you want to make the change. This dialog enables you to select the edit or collection point into which to insert new constructs.

2. To define your events in the Events Handler, select the post point after Edit Point 1 Additional events (which is under Edit Point 0 Additional events).

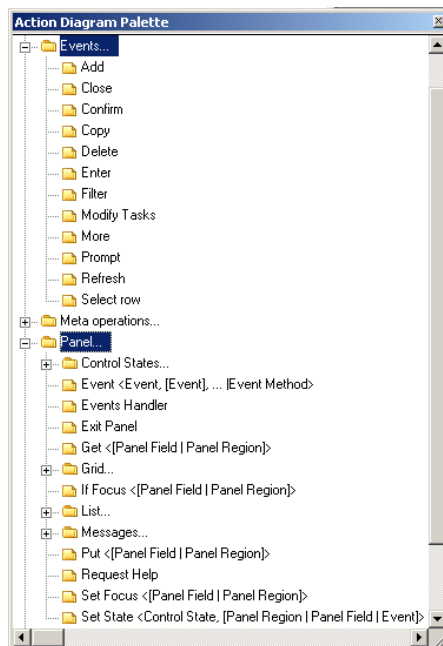


3. Now, you can see the three windows that make up the Action Diagrammer.

If these windows overlap, drag or resize them so that you can work with all of them at once.



4. In the Action Diagram Palette, expand Panel, and then Events.



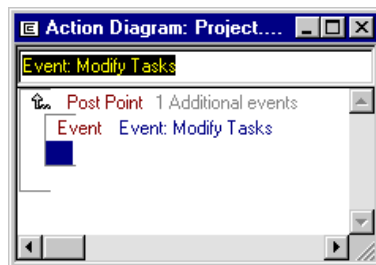
The Action Diagram Palette is a container for all the instructions you can use in an action diagram, grouped by similarity. The Panel folder holds all of the syntax elements you use when working with panel functionality.

In the Events group, notice the Modify Tasks logical event. You previously created this earlier in the chapter; see [Creating a Logical Event](#).

5. Drag the Modify Tasks logical event from the Action Diagram Palette to the input line at the top of the main window.

The main window automatically adds the text Event: to the line when you drop the event, indicating that you are adding the functionality for an event.

6. Press Enter. The Action Diagrammer adds the statement in the input line to the body of the Action Diagram:

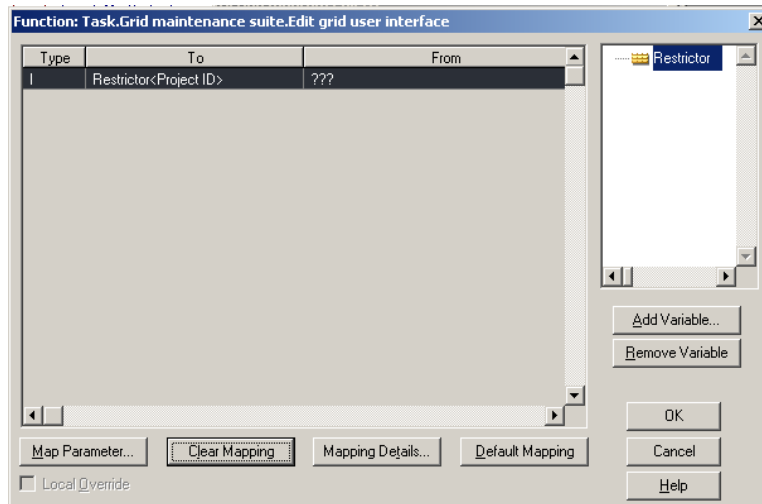


You specified that the function will do something when the Modify Tasks logical event is triggered. Next, you specify *what* the function does.

7. From the Object Browser, drag Task.Grid Maintenance Suite.Edit Grid User Interface to the input line of the Action Diagrammer (make sure you are dragging from Task, and not Project).

This time, the Action Diagrammer adds the syntax "Function: " to input line, so that the function calls the Task.Grid Maintenance Suite.Edit Grid User Interface function when the logical event is triggered.

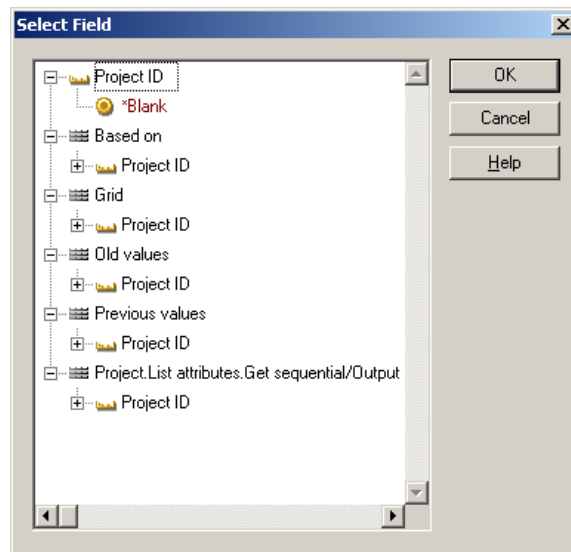
8. Press Enter. The Parameter Mapping dialog appears:



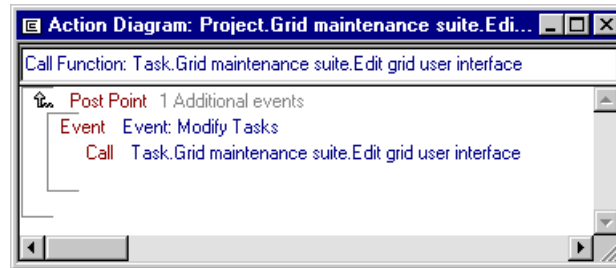
When you call one function from another, often you need to pass information. In this case, when you call the function Task.Grid Maintenance Suite.Edit Grid User Interface, you need to tell it which Project is selected in the Edit Project panel so that Task.Grid Maintenance Suite.Edit Grid User Interface knows which tasks to display on its panel.

Next, you indicate which variable in the Project.Grid Maintenance Suite.Edit Grid User Interface function to map it to.

9. Click Map Parameter. The Select Field dialog appears. This dialog shows you all of the values CA Plex could use for the Project ID field. Because you want to see the tasks assigned to the project that an end user selected on the Edit Project panel, you choose a Project ID field that comes from the grid on that panel.



10. Select the Project ID field that appears below the Grid entry and click OK. This indicates that the function will use the Project ID selected in the Grid region on the Edit Project panel to determine which tasks to display.
11. Click OK on the Parameter Mapping dialog. The Action Diagrammer appears. The Action Diagrammer should look like this:



Note: If you need to change the parameter mapping in the future, open this action diagram and double-click the Call line.

12. Close the Action Diagrammer and save your changes.
13. Save your model.

Generating and Building the Task Entity and Project.Grid Maintenance Suite.Edit Grid User Interface

Now that you have set up a way to edit the tasks for a project, you only have to generate and build the following to see the results of your work. For more information on the process of generating and building, see Generating and Building in the chapter "Defining Entities."

To generate and build the following:

- Task entity and all of its scoped objects, because you have fully defined this entity
- Project.Grid Maintenance Suite.Edit Grid User Interface, because you added a key code to the Edit Project panel to display the Edit Task panel

Testing the Application

You now have a fully functional application. You can enter projects and tasks, and also assign employees to those tasks. Do the following to test the application:

1. Run Project.Grid Maintenance Suite.Edit Grid User Interface.
2. Navigate to the grid selector field for Proj01 and enter **5** in the field.
3. Press Enter to open the Edit Task panel. The Edit Task panel appears:

```

Date . . . 7/30/98 Edit Task                                     Time . . . 14:01:02

Type options, press Enter.
3=Copy   4=Delete
? Project ID Task ID Task Description                           Employee ID

F3=Close F5=Refresh F6=Add F10=Update F17=Filter...
  
```

4. Press F6 to open the Add Task panel. The Add Task panel appears. This is the panel scoped to Task.Grid Maintenance Suite.Create User Interface.

Notice that the Project ID field is already filled.

```

Date . . . 7/30/98 Add Task                                     Time . . . 14:27:11

Project ID . . . . . PROJ03
Task ID . . . . . _____
Task Description . . . _____
Employee ID . . . . . _____

F3=Close F5=Refresh
  
```

5. Enter the following data for a task:

Value	Enter
Task ID	TASKA
Task Description	Create a data

The third field you need to fill, Employee ID, requires an exact match to the ID for the employee to whom you want to assign the task. However, you may not know to whom you want to assign a task or an employee's ID, so perform the next step.

6. Navigate to the Employee ID field and press F4. The List Employee panel appears.

```

Date . . . 7/30/98 List Employee                               Time . . . 14:36:57

Type options, press Enter.
l=Select
? ID      Name                                           Billing Rate
- EMP02   David Castro                                   32.00
- EMP03   Martha Kolman                                55.00

F3=Close F5=Refresh F17=Filter...                               Bottom

```

Remember, you already changed this panel to hide the Employee Email and Employee URL fields, for more information, see Hiding Fields on Panels in the chapter "Diagramming Entity Relationships."

7. Navigate to the grid selector field for Martha Kolman and enter **1**.
8. Press Enter. The Add Task panel reappears with the Employee ID filled in:

```

Date . . . 7/30/98 Add Task                                   Time . . . 14:36:57

Project ID . . . . . PROJ03
Task ID . . . . . TASKA
Task Description . . . Create a data model
Employee ID . . . . . EMP03

F3=Close F5=Refresh

```

9. Press Enter to accept the task's settings.
10. Repeat the preceding steps to add the following tasks:

Value	Enter
Task ID	TASKB
Task Description	Verify model with clients

Value	Enter
Employee	Martha Kolman

Value	Enter
Task ID	TASKC
Task Description	Decide which libs. to use
Employee	Martha Kolman

11. Press F3 to close the Add Tasks panel, and again to close the Edit Task panel.
12. On the Edit Project panel, type a space to remove the 5 from the grid selector region for PROJ01.
13. Navigate to the grid selector field for PROJ02 and enter 5.
14. Press Enter, then enter the following tasks for this project:

Value	Enter
Task ID	T130
Task Description	Architect email module
Employee	David Castro

Value	Enter
Task ID	T131
Task Description	Test email functionality
Employee	David Castro

Chapter Review

In this chapter, you:

- Defined the Task entity.
- Defined restrictor processing for Task, so that its panel only displays tasks that belong to a single project, and so that it automatically populates the Project ID field for new tasks.

- Added a grid selector value to the Edit Project panel, so that users can access the Edit Task panel from the Edit Project panel.
- Created a logical event for the grid selector value, and mapped it to a physical event.
- Added logic to an action diagram to specify functionality for the logical event. In doing that, you mapped parameters for a function call.
- Tested your application to see the changes you had made.

Helpful Hints

The following information offers you hints that are helpful when defining Owned By relationships.

Owned By Relationships and OBASE/Child

You already added the triple shown next using the Diagrammer. For more information, see Define an Owned By Relationship in the chapter "Diagramming Entity Relationships." You also added the following triple.

Task owned by Project

This specified a relationship between these two entities. That is, that Project owns Task. For more information, see Defining the Task Entity earlier in this chapter.

Task is a OBASE/Child

This gave the Task entity functionality to ensure that when a project is deleted, all of its child tasks are deleted, too. But you may wonder why, if you already added this triple, you also used the Template Editor to create the following triples:

Task **replaces** Owner
...**by** Project

You had already specified the owner in the Task **owned by** Project triple. Why did you seemingly have to specify the owner a second time?

The answer is that you did different things to the model each time you specified the owner. Adding the triple Task **owned by** Project gave Task an additional key field—Key of Project. That is all that happened when you added that triple. The **owned by** triple did not add any functionality to Task.

Task is a OBASE/Child gave Task functionality. Specifically, when you inherit from OBASE/Child, you get the ability to:

- Read only the task records for a specific project (known as restrictor processing). For more information, see Defining Restrictor Processing earlier in this chapter.
- Delete all of the tasks for a project when you delete the project.

However, the functionality that you inherit is abstract. That is, it does not know what is in your model. The pattern entity, OBASE/Child, uses an abstract entity, called Parent, to represent the owner. For the pattern to work correctly in your model, you have to replace this placeholder with the entity you want to be the owner. That is why you create the replacement triple.

Chapter 6: Putting It All Together

In the chapter, "Defining Owned By Relationships," you defined the Task entity, and then added processing enabling end users to add tasks to a project and to assign employees to those tasks.

Now you have an application with panels that enable you to add projects, add tasks to projects, add employees, assign employees to tasks, and access tasks for a project. In this chapter, you will tie all of the panels together with a top-level menu.

This section contains the following topics:

[Creating a Top-Level Menu](#) (see page 103)

[Generating and Building Your Applications](#) (see page 107)

[Chapter Review](#) (see page 108)

Creating a Top-Level Menu

CA Plex creates standard OS/400 programs that you can call from any System i menu system. OBASE provides its own menu system that you will use to create a top-level menu.

To create a top-level menu:

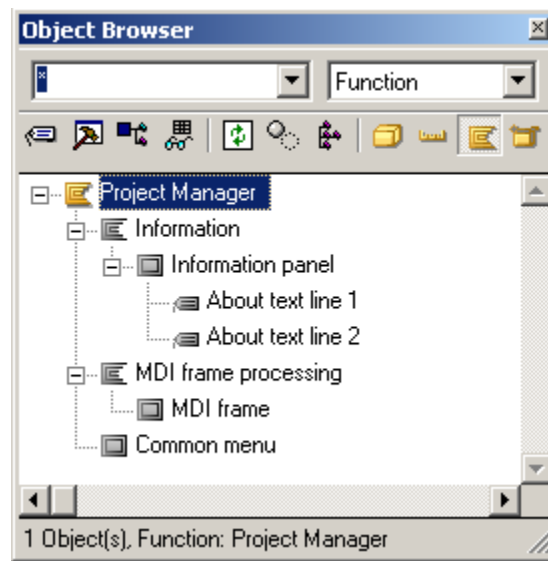
1. This is the first time you will create an inheritance triple to inherit from a function. Change the Object Browser to display functions.
2. In the Model Editor, change the object type to function, and add the following triple:

Project Manager is a OBASE/Default Objects.MDI Template

Note: To get to Default Objects.MDI Template in the Object Browser, display entities, make sure library objects are showing, and expand OBASE/Default Objects.

This defines the unscoped function, Project Manager. Remember, to view unscoped functions in the Object Browser, you must change the Object Browser to view functions. Until now, you have only worked with scoped functions, which you viewed with the Object Browser set to show entities.

3. Expand the Project Manager function to show the objects it scopes.



Project Manager has two functions scoped to it:

- Information that displays information about the application, and MDI Frame Processing, which displays the top-level menu
- It also scopes a panel called Common Menu, which is used primarily with non-5250 applications

Next, you modify the top-level menu to add menu items to it.

4. Open Project Manager.MDI Frame Processing.MDI Frame.
5. In the Panel Palette, select the Selector text region.
6. In the Design Window, move the Selector text region to the center of the panel.
7. Add the following static text items to the Selector text region. For more information, see Adding Static Text in the chapter "Modifying the User Interface."

1. Work with projects

2. Work with employees

8. Rearrange the static texts to make them visible:

Date . . : 66666666 Project Manager Time . . : 66666666

Enter selection. . . I
1. Work with projects
2. Work with employees

F3=Close F13=About

9. Add the following logical events to the panel. For more information, see To Add a Logical Event to the Edit Project Panel in the chapter "Defining Owned By Relationships."

Work with projects

Work with employees

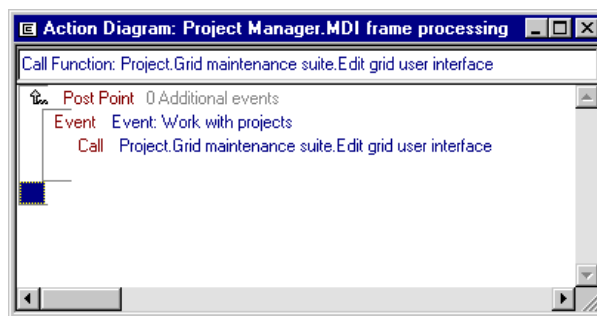
10. Add the grid selector values 1 and 2 to the panel, and associate them with the corresponding logical events. For more information, see Creating a Subfile Option in the chapter "Defining Owned By Relationships."

Instead of adding them under the Grid region's menu selector field, you add them under the Selector text region's menu selector field.

11. Close the panel and save your changes.
12. Edit the function Project Manager.MDI Frame Processing so that the application calls the function Project.Grid Maintenance Suite.Edit Grid User Interface when the Work with projects logical event is triggered. For more information, see Adding Functionality to Logical Events in the chapter "Defining Owned By Relationships."

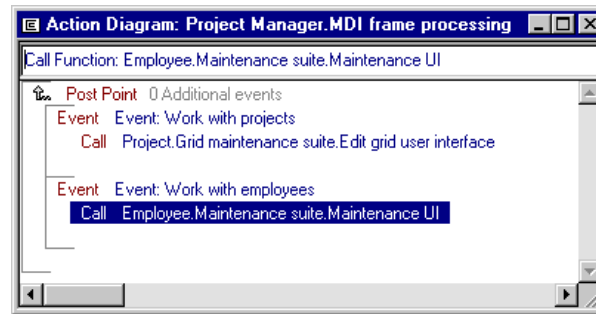
Choose the post point that follows the 0 Additional Events edit point.

13. Click in the Action Diagrammer under the construct you just created. The Action Diagrammer should look like this:



14. Add another instruction to the function so that it calls the function Employee.Maintenance Suite.Maintenance UI when the Work with employees logical event is triggered.

When your action diagram is finished, it should look like this:



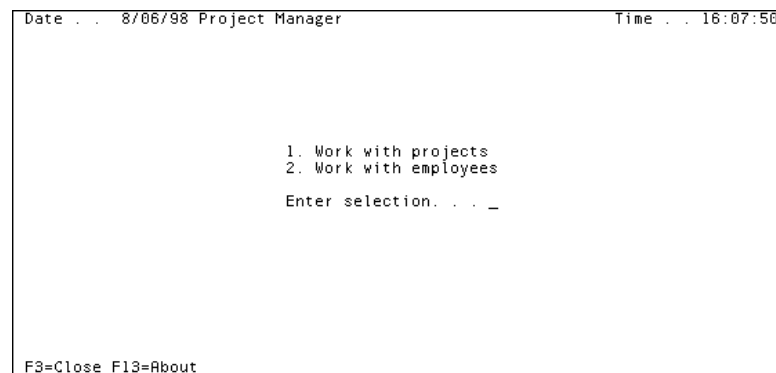
15. Close the function and save your changes.

Generating and Building Your Applications

Now, you are ready to generate and build your application and test it. After you complete these tasks, you will be finished with this tutorial. Generate and build the Project Manager and all of its scoped objects.

Testing the Application

After you generated and built the application, run the Project Manager.MDI Frame Processing function. A top-level menu panel appears that looks like this:



To test your application:

1. Enter **1** and press Enter. The Edit Project panel appears.
2. Navigate to the grid selector field for the first project.
3. Enter **5** and press Enter. The Edit Task panel appears.

Notice that only the tasks assigned to PROJ01 appear in this panel.

```
Date . . . 8/27/98 Edit Task Time . . . 10:48:43

Type options, press Enter.
3=Copy 4=Delete
? Project ID Task ID Task Description Employee ID
- PROJ01 TASKA Create a data model EMP03
- PROJ01 TASKB Verify model with clients EMP03
- PROJ01 TASKC Decide which libs. to use EMP03

F3=Close F5=Refresh F6=Add F10=Update F17=Filter... Bottom
```

4. Close the application.

Chapter Review

In this lesson, you:

- Created a top-level menu structure
- Tested the completed application
- Saw the fully functioning application that you created working

Index

A

accessing online help • 9
action diagrammer • 10, 89, 90, 91
 Action Diagram Palette • 90
 Variable Palette • 91

D

Data types, of Task's fields • 83
deleting • 13, 44, 80
 a project • 44
 employees • 80
 scoped objects • 13
deploying to multiple platforms • 10
diagram • 12, 85
 foreign keys displayed on (Fig) • 85
diagrammer • 10

E

entity • 12
entity relationships • 14

F

field • 12
FLD • 14, 33
 is a FLD • 14
 length NBR • 33
 type SYS • 14
function • 12

G

generators • 10

I

inheritance • 14, 33, 71
 and employee's fields • 71
 and project's fields • 33
Inheritance, and Task's fields • 83

O

OBASE/ • 14, 33, 71, 73, 83, 84, 103
 Child • 84
 Code • 71, 83
 Default Objects.MDI Template • 103

Grid Maintained Entity • 84
ISO date • 33
Narrative • 14, 33, 71, 83
Price • 71
Referenced Entity • 73
User Maintained Entity • 73

object browser • 10, 27, 33
 dragging objects to the Model Editor • 33
 specifying type of objects to display • 27
objects • 12, 13, 33, 61
 focusing on • 33
 represented in diagrams • 61
 scoped • 13
 unscoped • 13
online help, accessing • 9
opening • 30
 model editor • 30

P

panel • 12
panel designer • 10, 49, 50, 53
 design window • 50
 panel palette • 50
 Property Sheet • 53
platforms, deploying to multiple • 10

S

scoped objects • 13
 deleting • 13
software development process • 11

T

table • 12
Task entity • 83, 84, 85, 97
 adding key value • 85
 defining • 83, 84
 attributes • 83
 inheritance • 84
 testing • 97
triples • 14, 30, 48
 defined • 14, 30
 working with • 48
tutorial overview • 10

U

unscoped objects • 13

V

value • 12

verbs, defined • 14

verbs. See also triples • 14

view • 12