

CA Gen

Tutorial

Release 8.5



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 NON-INFRINGEMENT. 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 © 2013 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

Audience	9
Objectives	9
Tutorial Workflow	9
Tutorial Case Study-eGolf Services	10
Software Required to Complete Tutorial	10
Time Allotment	11

Chapter 2: Toolset Overview 13

Objectives and Time Allotment	13
Model-based Development	13
Diagrams	14
100 Percent Code Generation	16
100 Percent Error-free Code	16
Lesson Activity	17
Start the CA Gen Toolset	17
Review the Toolset's Interface	18
Open an Existing Model	19
Diagrams in the Tree View	21
Model Objects in the Tree View	23
Configurable Aspects of the Toolset	25
Save the Model	28
Close CA Gen	30
Review the Model Directory	31

Chapter 3: Analysis 33

Objectives and Time Allotment	33
The Case Study Business Requirements	33
Lesson Objectives and Time Allotment	34
Gathering Business Requirements	34
Develop the Data Model	35
Lesson Objectives and Time Allotment	35
Subject Areas	36
Entity Types and Entities	38
Attributes	40
Relationships	43

Entity Subtypes	47
eGolf Services Data Model	49
Lesson Activity	49
Develop the Process Model.....	58
Lesson Objectives and Time Allotment	58
Activity Hierarchy Diagram.....	59
Functions	59
Processes	61
Elementary Processes.....	63
eGolf Services Activity Hierarchy Diagram	66
Lesson Activity	66
Confirm the Process and Data Model.....	68
Lesson Objectives and Time Allotment	68
Confirmation.....	68
Lesson Activity	70
Detailing the Business Rules.....	82
Lesson Objectives and Time Allotment	82
Action Diagramming Basics	83
Process Logic Analysis	87
Lesson Activity	88
Defining Business Systems.....	146
Lesson Objectives and Time Allotment	146
Business System Definition.....	146
Lesson Activity	146

Chapter 4: Design 149

Objectives and Time Allotment	149
Purpose of Design	149
Application Types	150
Block Mode.....	151
Batch.....	151
Window.....	151
Client/Server	152
Proxies	152
Web	153
Web Client/Server Applications.....	153
Designing Server Procedures	154
Lesson Objectives and Time Allotment	154
Server Procedures	155
Lesson Activity	156
Designing Client Procedures	178

Lesson Objectives and Time Allotment	178
Client Procedures	179
Lesson Activity	197

Chapter 5: Construction and Test 263

Objectives and Time Allotment	263
Generation and Test	264
100 Percent Code Generation	264
Generating the Database	264
Lesson Objectives and Time Allotment	264
Database Generation	265
Lesson Activity	267
Referential Integrity Triggers	277
Lesson Objectives and Time Allotment	277
Referential Integrity	277
Lesson Activity	278
Packaging Procedures into Load Modules	281
Lesson Objectives and Time Allotment	282
Load Module Packaging	282
Lesson Activity	282
Generating the Application	284
Lesson Objectives and Time Allotment	285
Application Generation	285
Lesson Activity	285
Testing the Application	293
Lesson Objectives and Time Allotment	293
Application Test	294
Lesson Activity	295
Web Generation	304
Lesson Objectives and Time Allotment	304
Lesson Activity	304

Appendix A: BLOB 313

Audience	313
Objectives	314
Software Required to Complete this Appendix	315
Time Allotment	316
Analysis	317
Case Study	317
Data Model	318
Activity Model	323

Design.....	331
Server Design	331
Client Design	337
Construction and Test	374
Database Generation.....	374
RI Trigger Generation	378
Load Module Packaging.....	379
Configure the Build Tool to Support External Action Block Linking	382
Application Generation	384
Application Test.....	386

Chapter 1: Introduction

This section contains the following topics:

[Audience](#) (see page 9)

[Objectives](#) (see page 9)

[Tutorial Workflow](#) (see page 9)

[Tutorial Case Study-eGolf Services](#) (see page 10)

[Software Required to Complete Tutorial](#) (see page 10)

[Time Allotment](#) (see page 11)

Audience

This tutorial is intended as an introduction to the features and capabilities of CA Gen, a model-based application development tool.

It is expected that the person completing this tutorial will have some level of analysis and application design experience.

Objectives

After completing this tutorial, you will:

- Have developed an eBusiness application from start to finish using various features of the CA Gen Toolset
- Understand the full lifecycle of a CA Gen development effort
- Recognize the benefits of CA Gen
- Realize the power and flexibility for delivering complete solutions using a single skill set

Tutorial Workflow

CA Gen supports a number of different methods or methodologies, such as Information Engineering, Rapid Application Prototyping, and Component-Based Development. However, regardless of the method you use to design your application, you are still trying to understand these three things:

- What information is required?
- How to collect or display that information?
- How to manipulate the information once it is collected?

From an academic standpoint, it is probably easier to understand the tools and diagrams if you think about application development from the traditional standpoint of:

- Analyzing the requirements
- Designing a solution to meet the requirements
- Constructing and testing the application to meet the solution

The tutorial is organized in this fashion. As you gain experience with the toolset, you will discover a variety of ways to use it to meet your company's specific methods.

Tutorial Case Study-eGolf Services



Given the popularity of the Internet and Golf, we see an opportunity to provide golfers with a few simple online golf services. In exchange for these services, we hope to generate enough hits to our web site to allow us to then sell banner advertising space to the golfing industry. Therefore, what we plan to do is to rapidly develop an extremely reliable, high-performance, and very scalable eBusiness application.

Software Required to Complete Tutorial

CA Gen applications can be deployed to a large number of heterogeneous environments. This tutorial, however, is intended to be completed entirely on a local workstation configured for local construction and test.

Recognizing that everyone may not have such an environment, the Analysis, Design, and possibly parts of the Construction and Test modules can be completed with the CA Gen software alone, depending on your licensing agreements. To compile and test the generated source code, the following third-party software is required:

- To create a database from the generated DDL/DML, you will need a supported DBMS, along with the appropriate SQL precompiler.

- To compile the generated C source code, you will need a supported C compiler.
- To deploy the application to a Web Server, you will need a supported Java compiler, Java Runtime Environment, and a Java Web Server and Application Server.

Note: For information about supported third-party software and setup instructions, see the CA Gen software documentation.

Time Allotment

Introduction

5 minutes

Toolset Overview

40 minutes

Analysis

4 hours 30 minutes

Case Study

7 minutes

Data Modeling

40 minutes

Process Modeling

20 minutes

Model Confirmation

40 minutes

Recording Business Rules

2 hours 40 minutes

Defining Business Systems

3 minutes

Design

8 hours 35 minutes

Application Design

2 minutes

Types of Applications

3 minutes

Server Design

2 hours 15 minutes

Client Design

6 hours 15 minutes

Construction and Test

3 hours 10 minutes

Data Base Generation

40 minutes

Referential Integrity Triggers

15 minutes

Load Module Packaging

15 minutes

Application Generation

45 minutes

Application Test

45 minutes

Web Generation

30 minutes

Chapter 2: Toolset Overview

This section contains the following topics:

[Objectives and Time Allotment](#) (see page 13)

[Model-based Development](#) (see page 13)

[Diagrams](#) (see page 14)

[100 Percent Code Generation](#) (see page 16)

[Lesson Activity](#) (see page 17)

Objectives and Time Allotment

After completing this module, you will be familiar with:

- The concept of model-based development
- The CA Gen Toolset

Allow approximately 40 minutes to complete this module.

Model-based Development

CA Gen is a model-based development environment for designing, deploying, and maintaining high-performance, scalable business applications. It enables developers to deliver complex applications that meet rapidly changing requirements. For example, CA Gen allows developers to:

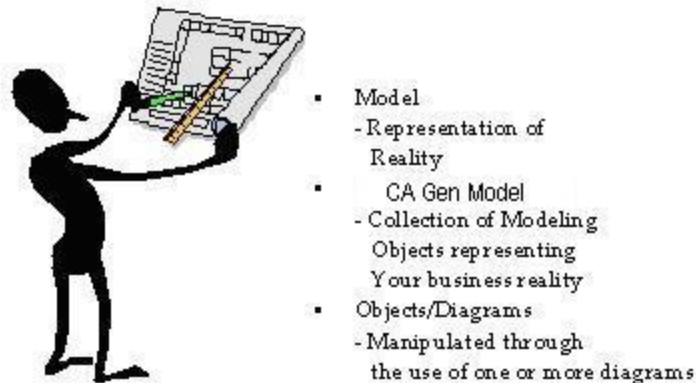
- Build reusable components.
- Web-enable applications.
- Integrate legacy software with new systems.

A model is a representation of reality. A model is not the real thing, but it represents the real thing. Models are generally used to test or perfect a final product. In our case, the model is a schematic description of a business system accounting for its known or inferred properties.

Technically, a CA Gen model is a collection of modeling objects, each of which represent different aspects of your business reality. These objects are created and managed using one or more diagrams in the CA Gen Toolset.

It is not that important to distinguish between the objects on a diagram versus the diagram itself. In fact, most people think of the collection of diagrams themselves as the model of their business, with each diagram representing their business from a slightly different viewpoint.

It may be helpful to think of the different diagrams in an CA Gen model as being analogous to a set of blueprints. A set of blueprints consists of several diagrams, each of which detail different aspects of the same building. There could be one diagram that shows the façade, another for the plumbing, and yet another for the electrical work. They all describe the same building, but each shows it from a different perspective.



Diagrams

Most business systems perform tasks similar to the following:

- Storing information in a database
- Displaying or capturing that information from a screen or a window
- Manipulating the information in some way in between the time it is captured from the screen or window and the time it is stored in the database

CA Gen has more than 30 diagrams; three are directly related to the database design, the window design, and the programming logic. They are:

- The Data Model Diagram (often referred to as the Entity Relationship Diagram)
- The Navigation Diagram

■ The Action Diagram

The other diagrams are used to assist with gathering requirements, model confirmation, or with the deployment of the generated applications.

CA Gen diagrams are integrated. You cannot add A to B giving C in an Action Diagram unless A, B, and C were first described in the Data Model Diagram. Technically, it is not so much that the diagrams are integrated, but rather that the objects created by one diagram can be reused or referenced in another diagram. It is the reuse of the modeling objects that brings about this integration.

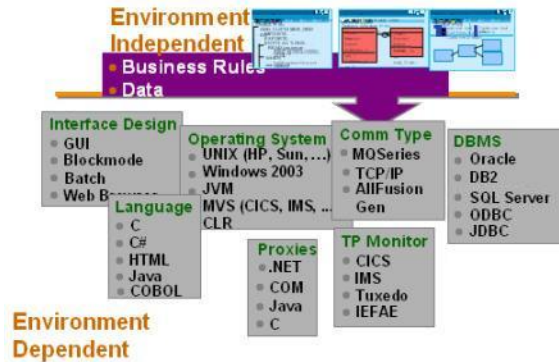
The Toolset also has a number of Wizards, Patterns, and Templates available to assist with the creation of many of these diagrams.



- **Key Diagrams**
 - Data Model Diagram (ERD)
 - Window Navigation Diagram
 - Action Diagram
- **Additional Diagrams**
 - Gathering Requirements
 - Deploying Applications
- **Integrated Nature**
- **Wizards, Patterns and Templates**

100 Percent Code Generation

The real strength of model-based development is that from a single model, we can generate the application for a variety of different environments, just by changing a few generation options. Therefore, your developers can focus their energies on understanding your business requirements, and not have to worry about having strong expertise with every technology in use in your environment. When your environment changes, all you have to do is change the generation options and regenerate the code.



100 Percent Error-free Code

CA Gen generates and provides runtimes for the entire application code. CA Gen also provides everything necessary to deploy the application to a supported environment — the clients, the servers, and the middleware. It generates 100 percent error-free code.

One caveat regarding the error-free code: while the generated code is always syntactically correct, this does not imply that it is logically correct. For example, if you meant to add A to B giving C, but instead specified that A should be added to D, CA Gen may not be able to catch that. However, the software does have mechanisms in place to reduce even these kinds of errors.

From a productivity standpoint, some rather incredible numbers regarding the timesaving benefits of CA Gen have been quoted from time to time. Every CA Gen situation is different, of course. However, to say that CA Gen can take you through your development phase three times faster than before, while reducing subsequent maintenance effort by a factor of ten, would be in line with many past experiences.

Lesson Activity

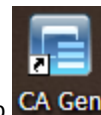


In this exercise, we are going to:

- Start the CA Gen Toolset
- Review the Toolset's interface
- Open an existing model named EGOLF SERVICES
- Point out the different diagrams in the Tree View
- Point out the different model objects in the Tree View
- Point out some of the configurable aspects of the Toolset
- Save the model
- Close CA Gen
- Open Windows Explorer and review the model directory

Start the CA Gen Toolset

Depending on how and where you installed the software, you can either:

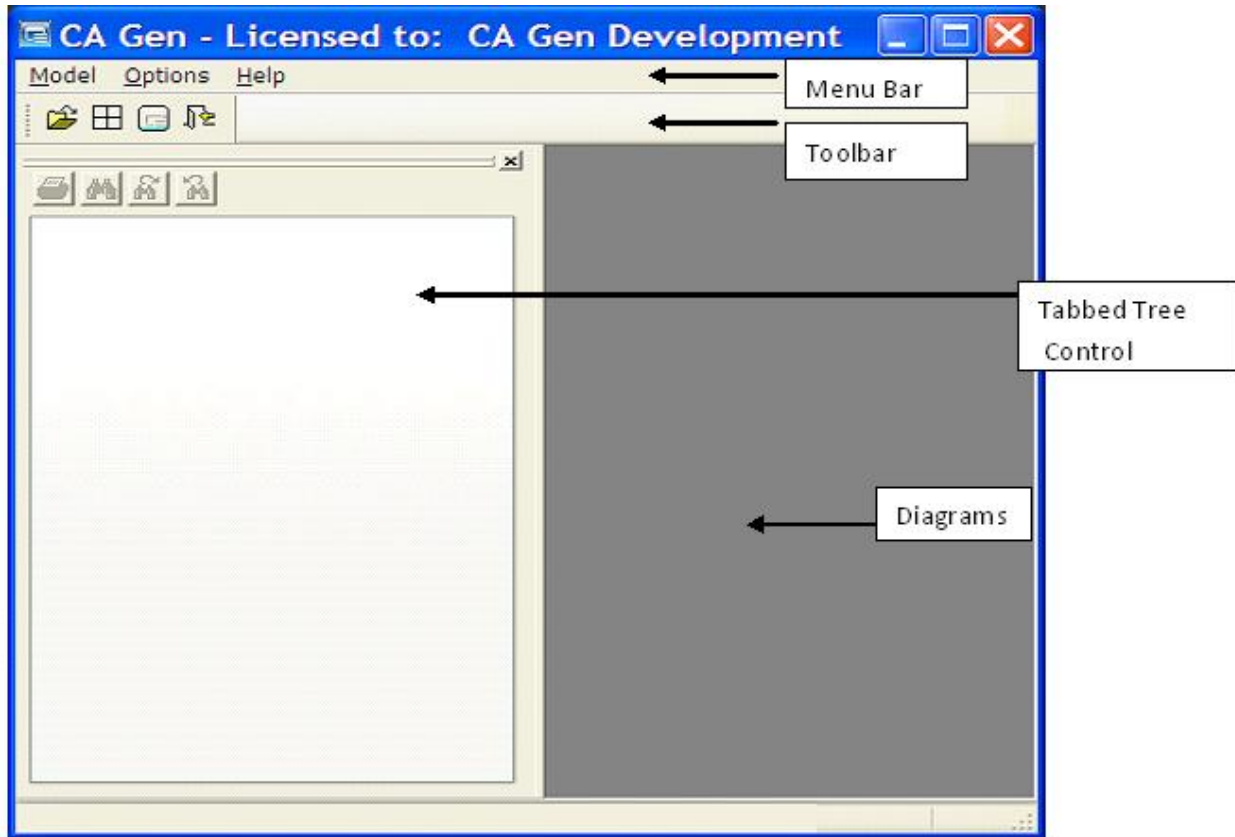


- Double-click the CA Gen icon on the desktop
- Click Start, All Programs, CA, Gen xx, Toolset.

Note: xx is the current CA Gen release number.

Review the Toolset's Interface

When the Toolset first opens, you see two blank panels as shown in the following example.



After opening or creating a new model, the panel on the left contains the Tabbed Tree Control. The panel on the right contains diagrams that we open.

Note: The number of items available on the Menu Bar and Toolbar change when we open or create a model, or when we open different diagrams.

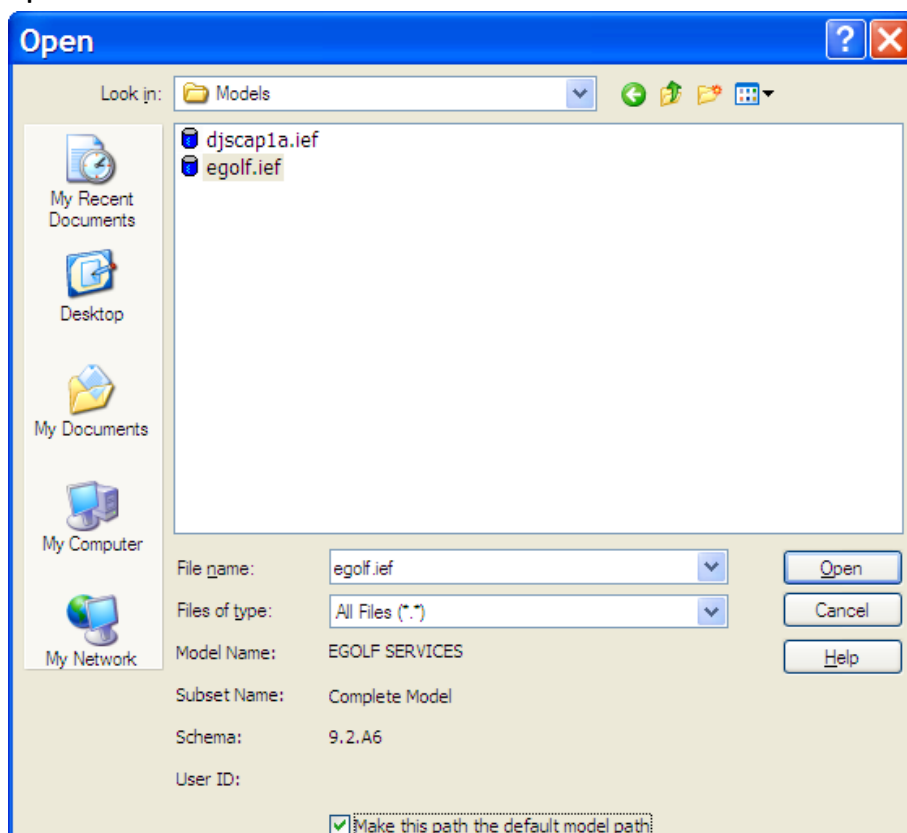
Open an Existing Model

Normally, we might start by creating a brand new model. In this case, we will start with an existing model that has a software (more about components when we get to that point in the tutorial) component copied into it. In addition, we have included some rudimentary bmp and gif images, which will be used for some of the examples in our application.

Follow these steps:

1. Using Windows Explorer create a directory (folder) to contain your models. Throughout this Tutorial, the model directory used in the examples is C:\Models. Copy egolf.ief and djscap1a.ief from the \Samples\Models\Tutorial Models\Starter Models directory to your models directory.
2. Select **Model** from the menu bar, and select **Open Model...** from the drop-down menu.

In the subsequent file Model Open dialog box, change the **Look in:** path to your model directory, select **egolf.ief** from the list of models, check the **Make this path the default model path** checkbox at the bottom of the dialog box, and then click **Open**.



3. Select **Model** from the menu bar, then select **Info...** from the drop-down menu.

Note: The model name is EGOLF SERVICES and the local name is egolf in the Model Information dialog.

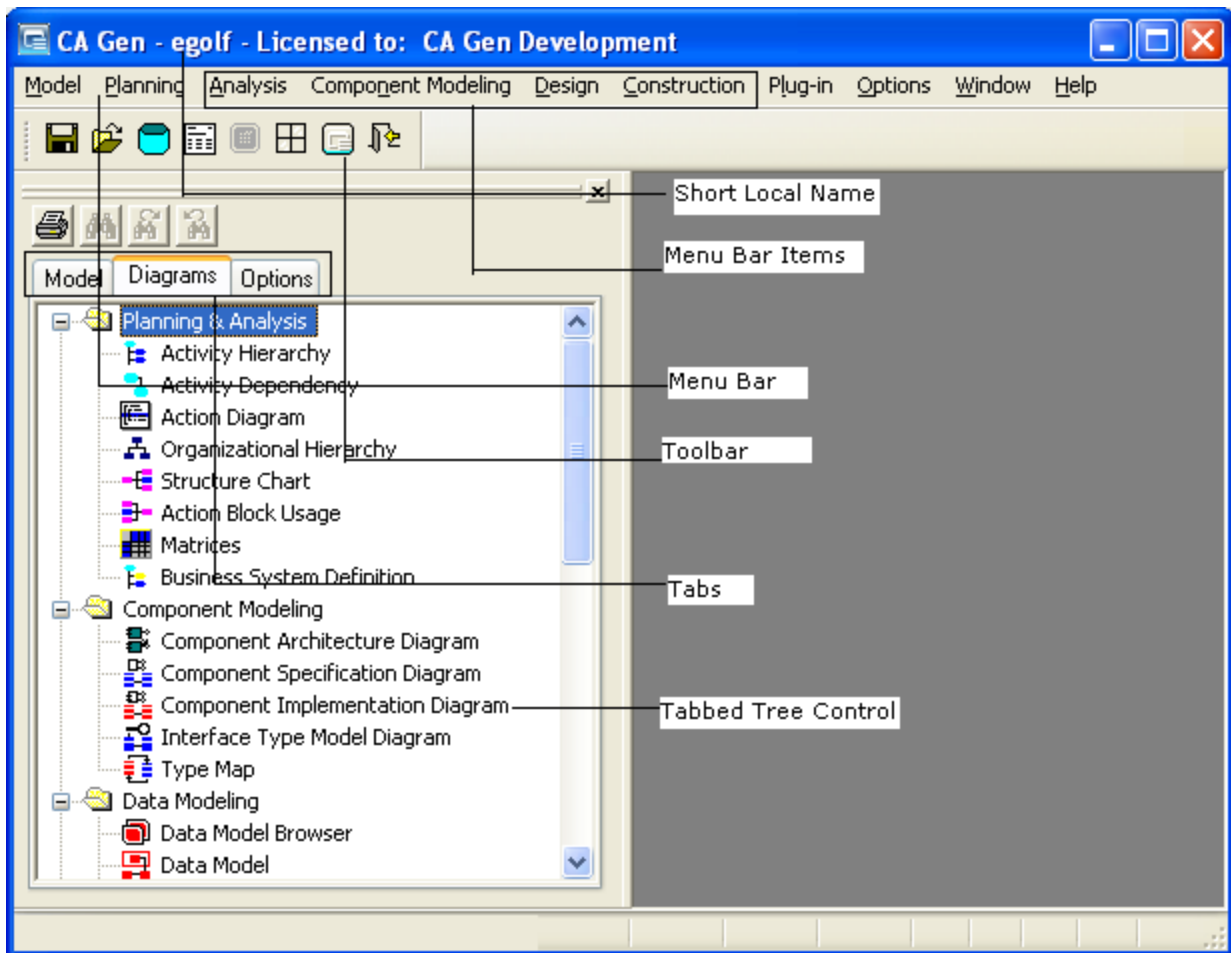
Model Information													
Model name:	EGOLF SERVICES												
Subset:	ALL												
Local name:	C:\models\egolf.ief\												
Model stage:	Business System Design (BSD)												
Model status:	Checked-out												
Model level:	8.0.0.0.05050												
Code Page ID:	1252												
Total number of objects in model:	1,179												
Objects created since last check-out:	0												
Number of objects available before check-in:	2,147,483,645												
<div>Workstation</div> <div>Schema: 9.2.A6 Version: 08.5.00.04014</div>													
<table border="1"><thead><tr><th></th><th>Date:</th><th>Time:</th></tr></thead><tbody><tr><td>Last save:</td><td>Apr. 05, 2010</td><td>11:46</td></tr><tr><td>Last check-out:</td><td>Apr. 05, 2010</td><td>11:46</td></tr><tr><td>Last update:</td><td></td><td></td></tr></tbody></table>			Date:	Time:	Last save:	Apr. 05, 2010	11:46	Last check-out:	Apr. 05, 2010	11:46	Last update:		
	Date:	Time:											
Last save:	Apr. 05, 2010	11:46											
Last check-out:	Apr. 05, 2010	11:46											
Last update:													
<div>OK</div> <div>Help</div>													

Models have a [long] Model name and a [short] Local [model] name. If we were to upload this model to an Encyclopedia, the name of the model as it was stored in the Encyclopedia would be the long model name. However, here on our local workstation, the model is placed in a folder with the local model name followed by the .ief extension. We will look in this folder later in this module.

4. Click **OK**.

Diagrams in the Tree View

In the following example, notice how the window has changed now that our new model has been opened in the Toolset.



The short Local model name appears in the Title Bar. This makes it easy to verify which model you are in.

We also have a number of new items available within the Menu Bar and Toolbar. On the Menu Bar, for example, we have items for Analysis, Design, and Construction, a sequence that roughly describes how we are going to proceed through the remainder of this tutorial. Selecting these Menu items reveals a list of diagrams normally associated with that activity.

There is also a Tabbed Tree Control in the left panel. At the top of the Tabbed Tree Control are three tabs:

- Model

- Diagrams
- Options

Follow these steps:

1. Select the Menu items **Analysis**, **Design**, and **Construction** and review the list of diagrams normally associated with those tasks.
2. If the Diagrams tab is not selected, select the **Diagrams** tab on the Tabbed Tree Control.

With a few exceptions, the same diagrams available through the Menu items are available through the Diagrams tab of the Tabbed Tree Control.

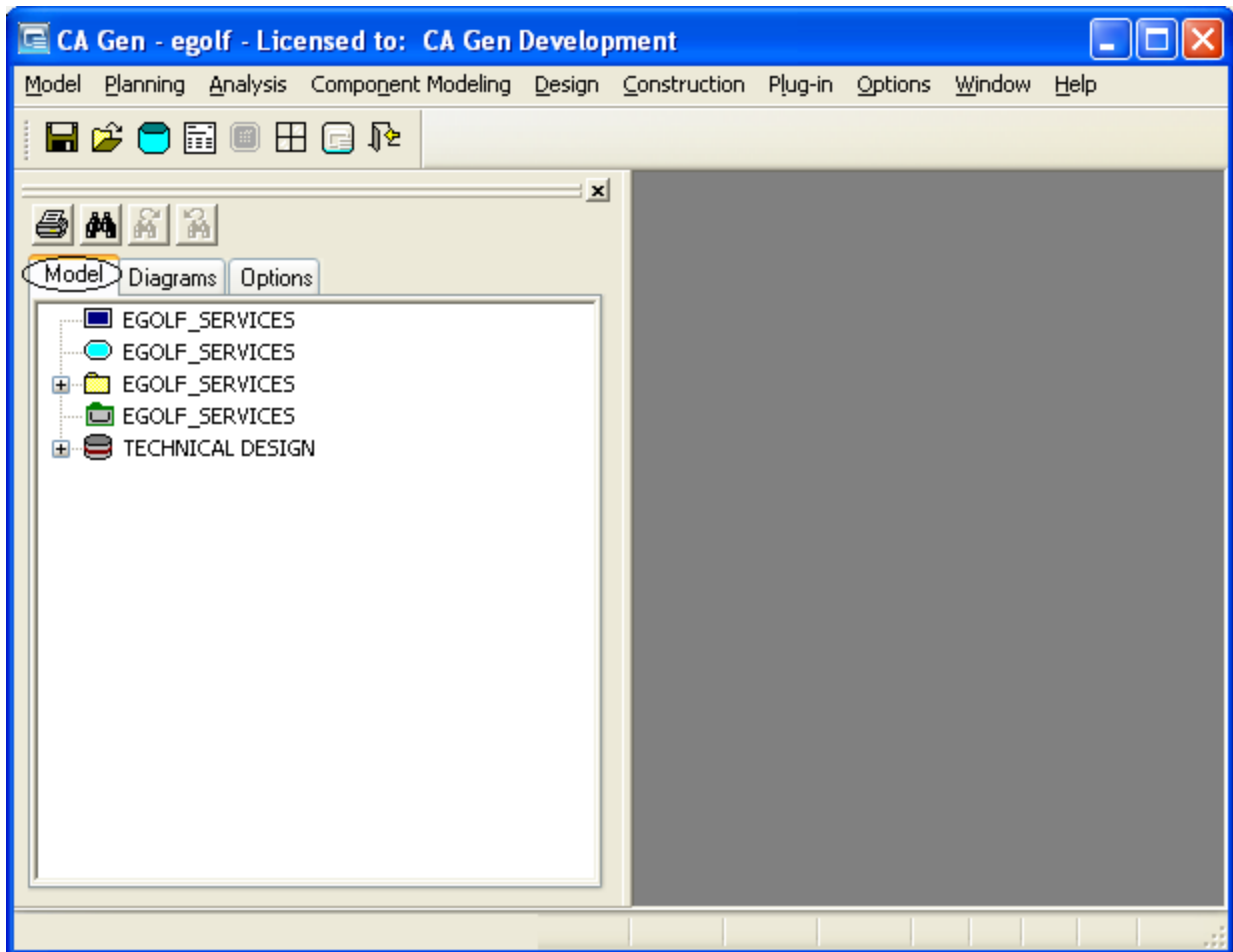
Double-clicking a diagram in the Tabbed Tree Control is a convenient way to launch (or open) the various diagrams. Hereafter, we will refer to the Tabbed Tree Control as just the Tree Control.

Model Objects in the Tree View

As mentioned earlier, you create modeling objects using the various diagrams in the Toolset.

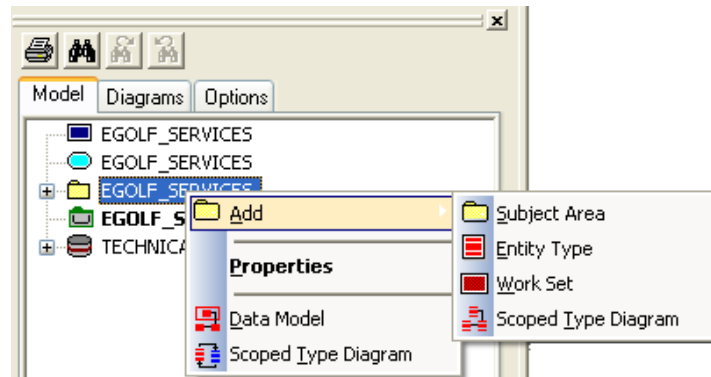
Follow these steps:

1. Select the **Model** tab on the Tree Control.



Note: In addition to the component objects already copied into the model, CA Gen adds a base set of objects to every new model.

2. In the Tree Control, right-click the **EGOLF_SERVICES** subject area (the entry with the manila folder icon next to it) and then select **Add** from the pop-up menu.



In addition to the menus in the Menu Bar at the top of the Toolset, pop-up menus are available by right-clicking an object in the Tree Control or from within any of the diagrams. Use pop-up menus as a shortcut instead of using the main Menu Bar.

From the EGOLF_SERVICES pop-up menu, we can:

- Add additional Subject Areas or Entity Types.
- Review the properties of the selected subject area.
- Launch the Data Model diagram.

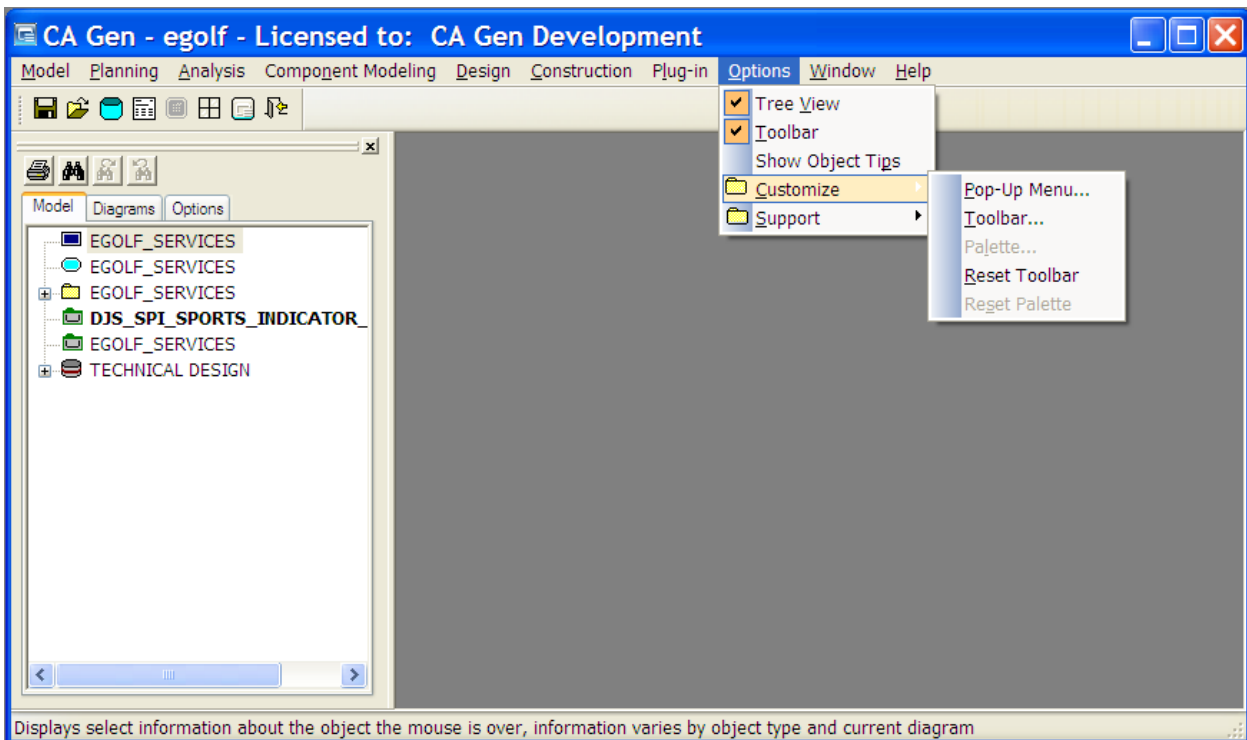
We will learn about all of these tasks in later modules. The pop-up menu items will vary depending on the object selected, and can be customized to your individual preferences.

Configurable Aspects of the Toolset

The Toolset is fully configurable.

Follow these steps:

1. From the Menu Bar, select **Options**, and select **Customize**.

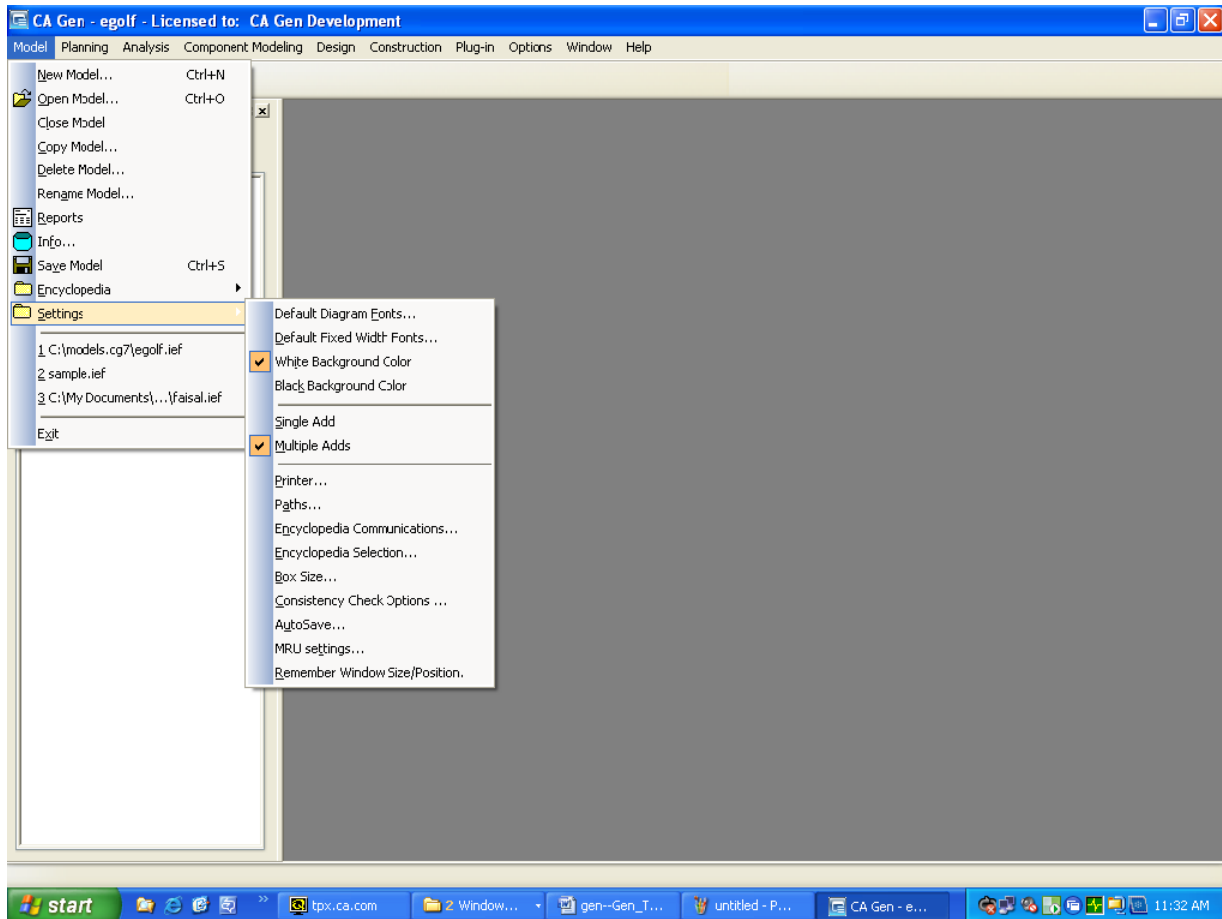


The contents of the pop-up Menus, Toolbars, and Tool Palettes (which we have not seen yet) can be customized from here. For more information about customizing, see the Toolset Help.

In addition, if you would rather not use the Tree View, Toolbars, or Tool Palettes, you can hide them by removing the corresponding checkmark next to their Menu item.

Note: The Palette check box is not visible since no diagram is open.

2. From the Menu Bar, select **Model**, and select **Settings**.



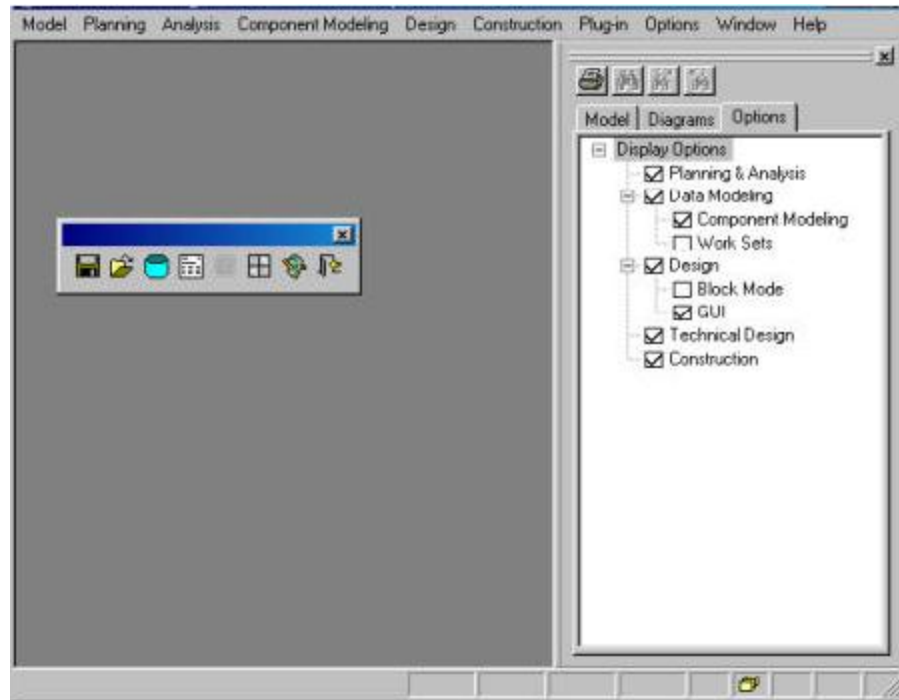
From here, you can set options like the background color for your diagrams, as well as box and font sizes.

Other important settings include:

- The path to your local model directory (Paths...)
- The encyclopedia with which you want to communicate (Encyclopedia Selection...)
- Consistency Check Options (Consistency Check Options...)
- AutoSave settings (AutoSave...)

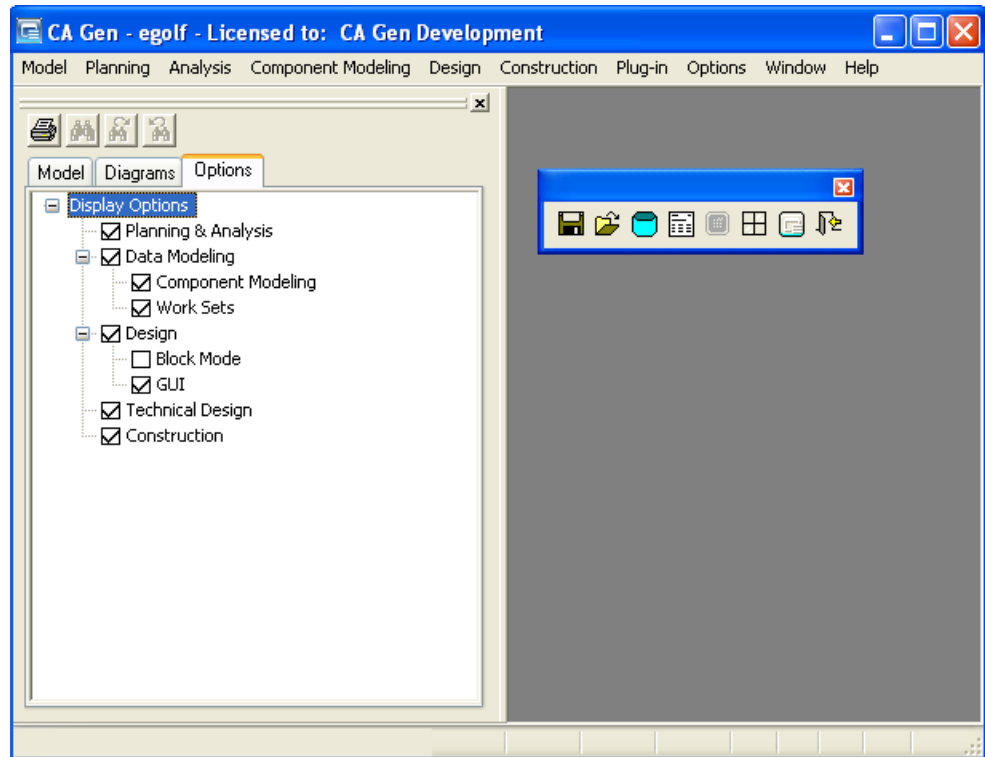
The Multiple Adds setting allows you to do repetitive tasks more efficiently. For example, adding several attributes to an Entity Type.

The Tree View, Toolbars, and Tool Palettes can be moved from their starting positions in the Toolset to any other position in or out of the Toolset. In the following example, the Tree View has been moved to the right side of the Toolset, whereas the Toolbar has been moved in front of the Toolset. This is particularly nice if you can afford a multi-monitor setup. To move them, select and hold their grab bar while dragging them where you want them.



3. Select the **Options** tab on the Tree Control as in the previous example.


Removing or adding checkmarks to the Tree Control Display Options determines what is viewable under the other two tabs. For instance, removing the checkmark from Data Modeling would remove the Data Model diagrams from the Diagrams tab as well as the Subject Area objects from the Model tab.



Save the Model

Changes to the model are not committed until they are explicitly saved. When you save the model, CA Gen saves all of the changes made to all of the objects. For example, if you make three changes to three different objects using three different diagrams, one save will save all of the changes in the model.

Follow these steps:

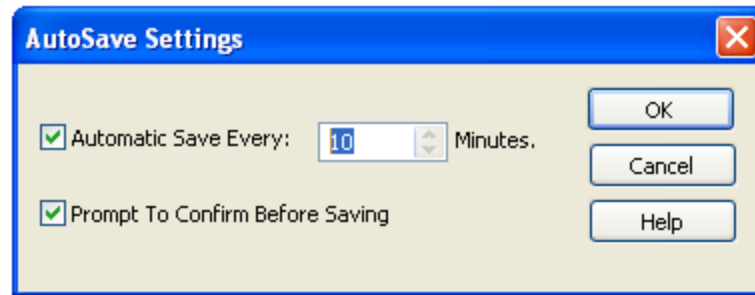
1. Do one of the following
 - Select the  icon from the Toolbar.
 - From the Menu Bar, select **Model**, and then select **Save Model**.
 - Press **Ctrl+S**

AutoSave

The Toolset can be configured to automatically perform saves for you every few minutes.

Follow these steps:

1. From the Menu Bar select **Model, Settings, and AutoSave....**



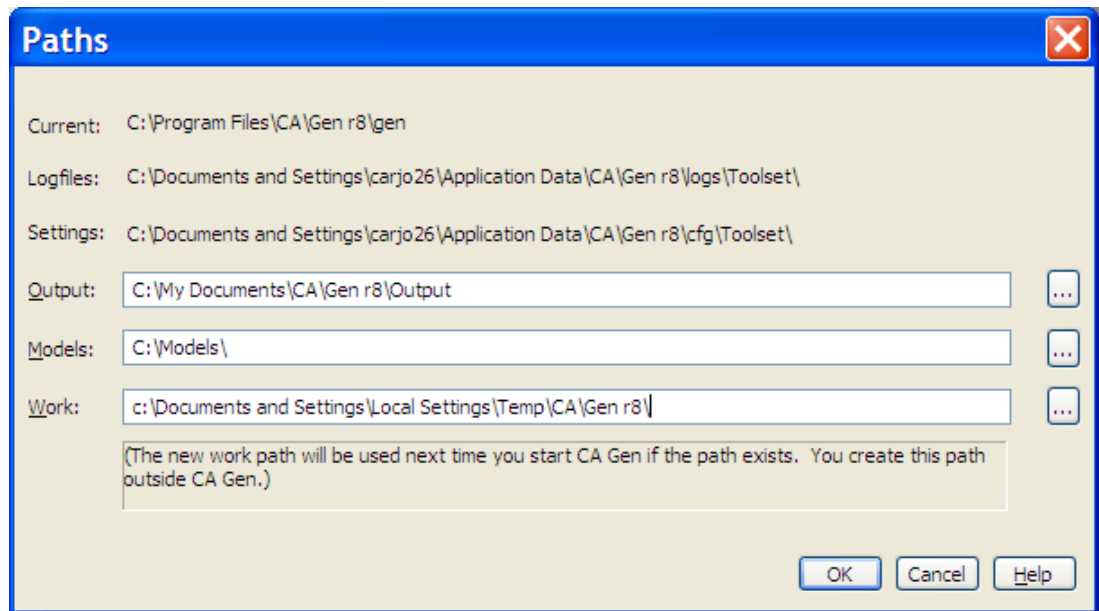
2. Click the **Automatic Save Every** check box and enter the number of minutes in the text box. If you want CA Gen to prompt you before performing the save, check the **Prompt To Confirm Before Saving** check box.
3. Click **OK**.

Close CA Gen


Before closing CA Gen, verify the path to the model directory.

Follow these steps:

1. From the Menu Bar, select **Model, Settings, Paths**.

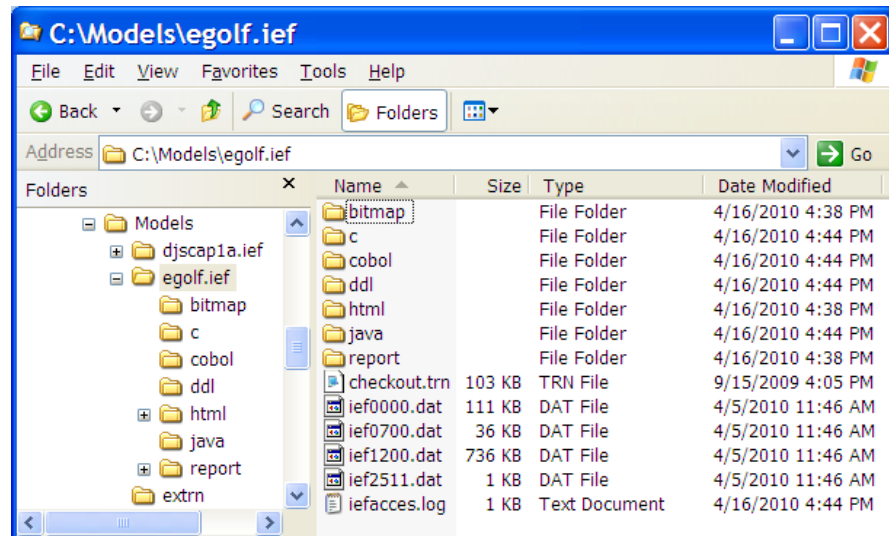


The path to our model is C:\Models. Your path may differ.

2. Note your path and select Cancel.
3. To close CA Gen, do one of the following:
 - Select the  icon in the upper-right corner of the Toolset.
 - From the Menu Bar, select **Model**, and then **Exit**.

Review the Model Directory

Open Windows Explorer and navigate to the model directory. The `egolf.ief` directory contains several folders and four files ending with the extension `.dat`, as shown in the following example:



These four `.dat` files—commonly referred to as the DAT files—comprise your local model. If you need to copy your model to another machine, at a minimum you would need to copy all four of these files, and place them in a directory with the `.ief` extension. The other folders are used for the generated code.

Chapter 3: Analysis

This section contains the following topics:

[Objectives and Time Allotment](#) (see page 33)

[The Case Study Business Requirements](#) (see page 33)

[Develop the Data Model](#) (see page 35)

[Develop the Process Model](#) (see page 58)

[Confirm the Process and Data Model](#) (see page 68)

[Detailing the Business Rules](#) (see page 82)

[Defining Business Systems](#) (see page 146)

Objectives and Time Allotment

After this module, you will have an understanding of the following terms:

- The Case Study Business Requirements
- Data Modeling
- Process Modeling
- Recording Business Rules
- Model Confirmation
- Process Synthesis
- Component-based Development
- Business System Definition

Allow approximately 4 hours and 30 minutes to complete this module.

If you would like to reference a CA Gen model reflecting the completed objectives of this chapter, you can find the model with the CA Gen installation in the \Samples\Models\Tutorial Models\Completed Models directory corresponding with this chapter.

The Case Study Business Requirements

The following sections deal with the requirements for a business case study.

Lesson Objectives and Time Allotment

After this lesson you will understand:

- How requirements are gathered
- The business requirements for the eGolf Services website

Allow approximately 7 minutes to complete this lesson.

Gathering Business Requirements

Business requirements are usually gathered through some combination of informal interviews, formal Joint Application Design (JAD) sessions, or reviewing current system functionality.

The following is a transcript from an informal interview with the President of eGolf Services:

Given the popularity of the Internet and Golf, we see an opportunity to provide golfers with a few simple online golf services. In exchange for these services, we hope to generate enough hits to our Website to allow us to eventually sell banner-advertising space to the golfing industry.

The first service we are going to provide is the calculation of a golfer's Handicap Index, which is an indicator of a player's potential scoring ability on a course of standard difficulty. A player would look up their actual Handicap for a particular Course by cross-referencing their Handicap Index against that Course's Handicap Table.

To accurately calculate a golfer's Handicap Index, we need to know scores of their most recent 20 golf games. Five games can be used for new golfers. In addition, we need to know the Rating and Slope Rating for the course the game was played on.

The way we see this service working is that the golfers will first register with us for the service. It will be free of charge, requiring them to enter only a few facts about themselves, like user ID, password, name, and email address. Thereafter, they can login and enter their scores, and we will then recalculate their Handicap Index based on their most recent entries.

As Analysts, we would review the information obtained through the interview, looking for Entities/Entity Types (things of interest to the business) and Processes (what the business intends to do with these things). Entity Types are documented in the Data Model Diagram (often named as the Entity Relationship Diagram), while the business Processes are documented in the Activity Hierarchy Diagram.

As one can expect, there is (or should be) a correspondence between these two diagrams. From the previous interview, one of the things the business intends to do is to Calculate Golfer Handicap Index. This would be documented in the Activity Hierarchy Diagram. Likewise, it should be reasonably clear that one of the things of interest to the business should be the Golfer, and the Golfer would be documented in the Data Model Diagram. We will discuss each of these diagrams in more detail in the next few sections.

Develop the Data Model

The following sections deal with developing a data model.

Lesson Objectives and Time Allotment

After this lesson, you will be familiar with:

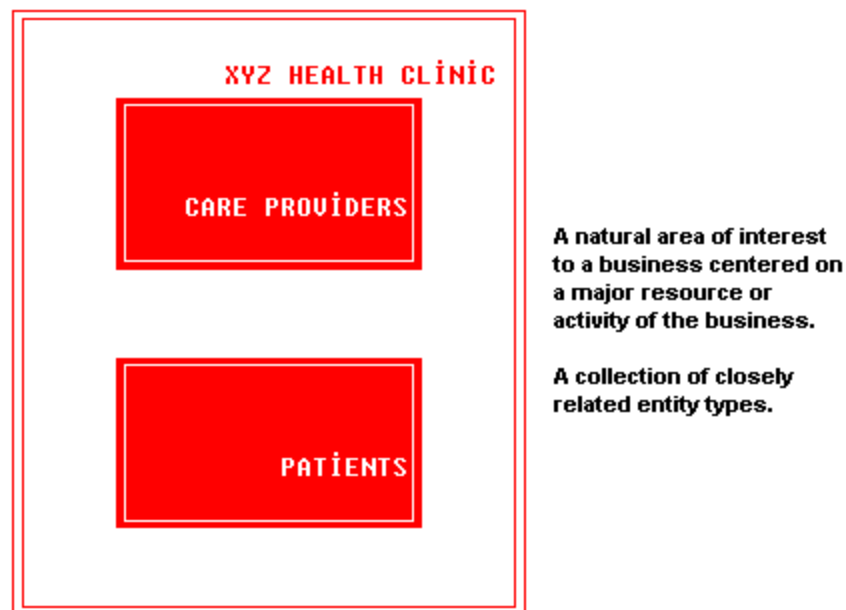
- The five basic modeling objects used in a Data Model Diagram
 - Subject Areas
 - Entity Types
 - Attributes
 - Relationships
 - Entity Subtypes
- The Data Model for the eGolf Services business

Allow approximately 40 minutes to complete this lesson.

Subject Areas

Subject areas are a natural area of interest to a business, centered on a major resource or activity of the business. In simpler terms, a subject area is a collection of closely related entity types (entity types are discussed in Entity Types and Entities).

Let us put our eGolf Services case study aside for a moment and consider another case study we will be using in this tutorial, that of the XYZ Health Clinic. Thinking about a health clinic, what can be a natural area of interest centered on a major resource or activity? In the following partial data model, Patients and Care Providers are likely natural areas of interest to a health clinic. Other subject areas include Appointments, Treatments, and Billings.



In the previous partial data model, we have three subject areas:

- XYZ Health Clinic
- Care Providers
- Patients

The diagramming notation for subject areas is a double red line, or, when the subject area is contracted, a red box with a white bounding line. The naming convention for subject areas is typically a plural noun.

The subject area XYZ Health Clinic is known as the root subject area, since all other subject areas fall within it. The name of the root subject area defaults to the long model name but can be changed if required.

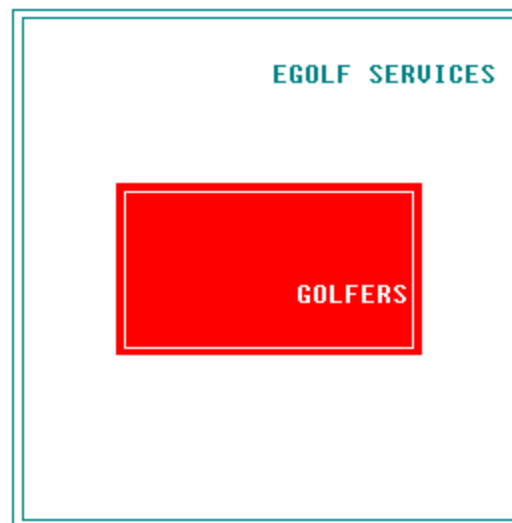
Take a moment to go back and review the interview with the President of eGolf Services and try to identify candidate subject areas. See the next section for suggested answers.

eGolf Services Subject Areas

Reviewing the interview with the President of *eGolf Services*, candidate subject areas include the following:

- Golfers
- Services
- Advertising
- Courses

It is clear that we want to attract Golfers to our Website, which we will do by providing them with Services. This is to attract or sell Advertising. In addition, one of the initial services requires information about Courses. Based on the limited information obtained in the interview (and to reduce the scope of this tutorial), we are going to focus only on Golfers. So within our EGOLF SERVICES root subject area (similar to the XYZ Health Clinic root subject area) we would have another subject area named GOLFERS, as shown in the following example.

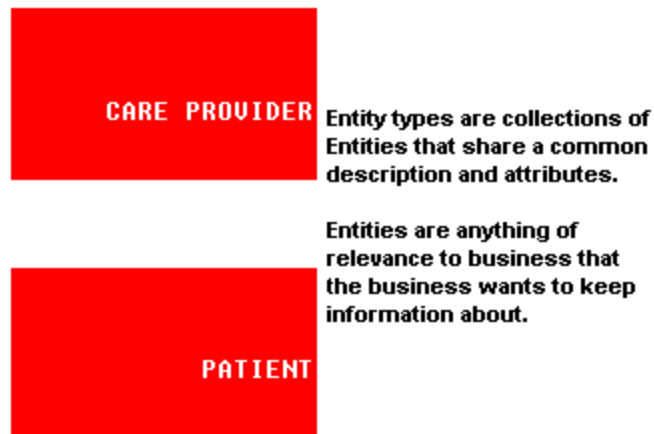


We will see how to document this information in the Toolset at the end of this section.

Entity Types and Entities

Entity types are collections or groupings of entities that share a common description and attributes. Entities are anything of relevance to a business that the business wants to keep information about.

To illustrate the difference between entity types and entities, let us use an example from the XYZ Health Clinic. John Doe can be a patient of the XYZ Health Clinic. Mary Smith can be a Doctor who works at the XYZ Health Clinic. Both are entities, things that are relevant to the business that the business wants to keep information about. However, they are entities of different entity types. John Doe is an entity of the entity type Patient, and Mary Smith would be an entity of the entity type Care Provider. This is because their descriptions are different. The description for a patient can be someone who seeks treatment at the clinic, whereas the description for the care provider can be someone who provides treatments at the clinic. Their attributes (attributes are discussed in Attributes) are likely different as well.



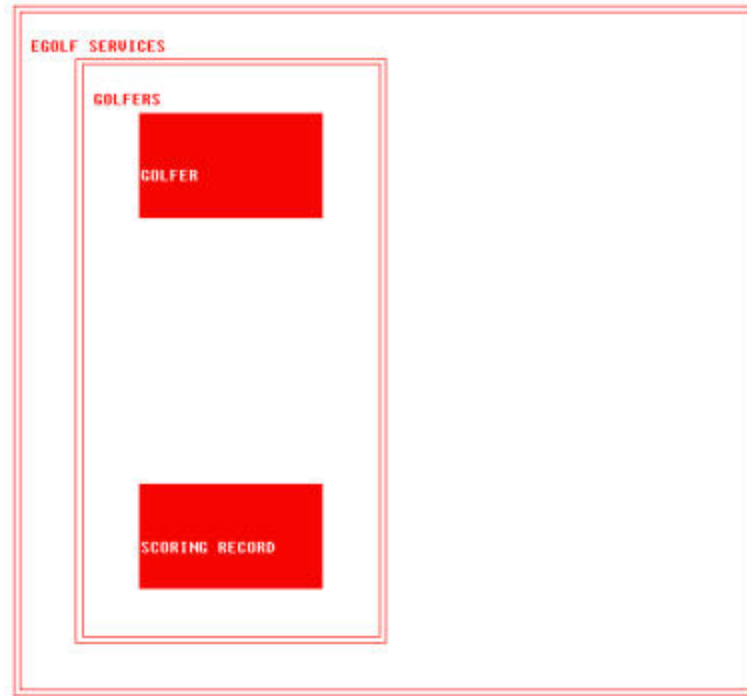
Entity types (not the entities within them) are depicted in the Data Model Diagram and fall within Subject Areas.

The diagramming notation for entity types is a red box. The naming convention for entity types is a singular noun.

Based on our interview with the President of eGolf Services, what might be candidate entity types within the Golfers subject area?

eGolf Services Entity Types and Entities

Golfer should obviously be one entity type. In addition, the golfer's Scoring Record would be another. Both of them represent collections of things that are of interest to the business and which the business wants to keep information about. In addition, since they are closely related, they probably belong to the same subject area, Golfers.



Entity types represent collections of things that are of interest to the business, but they do not always have to be tangible things like a Golfer or a Client. They can also represent conceptual things, like a Cost Center or a Ledger Entry, and active things, like Lecture Attendance or Equipment Breakdown. Of course, tangible things are more readily identifiable.

Attributes

Attributes are facts about entity types that the business cares about. For instance, the patient can have a favorite lollipop flavor, but it is unlikely that the business cares enough about that information to keep it. However, it probably cares about the patient's Name and Address.



Facts about an Entity that the business cares about.

Based on the interview with the eGolf Services President, what attributes do we care about for the entity types Golfer and Scoring Record?

eGolf Services Attributes

Attributes for Golfer include:

- User ID
- Password
- First Name
- Last Name
- Email Address
- Handicap Index

Attributes for Scoring Record include:

- Date
- Time
- Adjusted Gross Score
- Course Rating

- Course Slope Rating
- Note

Some of these attributes were obvious, while others were inferred. During your analysis, it is generally okay to make assumptions or inferences. In this case, we made assumptions and inferences about attributes, but we can have made them about anything. Regardless of what they are about, however, any assumptions or inferences that you make should always be reviewed with the user community.

Additional Attribute Properties

Attributes also have these properties:

Category

Attributes can be one of the following source categories:

Basic

An attribute whose value is supplied to the business, such as a client's name

Designed

An attribute whose value is invented or calculated and never changes, such as an employee number

Derived

An attribute whose value is derived from the values of other attributes, such as the total of an order

Domain

The collection of possible values for an attribute:

Text

A character string

Numeric

Numbers

Date

A valid date

Time

A valid time

Timestamp

A combination of date and time, to the level of microseconds

DBCS Text

Double byte character set

Mixed Text

A combination of double byte and single byte character sets

Permitted Values

Restricts a domain to some subset of permitted values

Length

The maximum number of characters or digits for each of the attributes' values

Number of decimal places (for numeric attributes)

Of its total length, the number of digits to the right of the decimal place

Optional

Whether the attribute must always contain a value

Case Sensitive, for text attributes

Indicates if the attribute value can contain lowercase and uppercase characters. If not, the tool automatically converts them to uppercase.

Varying Length

If selected, the database only stores the actual length of the text string

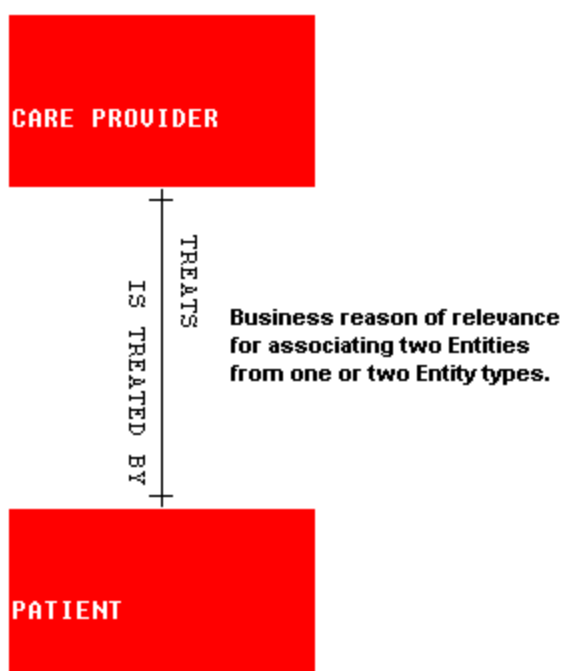
We will see each of these again when we build the data model.

Relationships

Relationships are reasons of relevance to the business for associating two entities from one or two entity types. In the following example, we have Care Providers and Patients. Is there a reason why the business wants to associate specific entities from Care Provider with specific entities of Patient? If the business wanted to know that Dr. Smith treated John Doe, then we would need a relationship to depict this business relevance.

Relationships have two memberships, one each from the point of view of the participating entity types. From the point of view of the Care Provider, Each Care Provider treats Patient. From the point of view of the Patient, Each Patient is treated by Care Provider. To read a relationship, start with either of the two entities and proceed clockwise around the diagram.

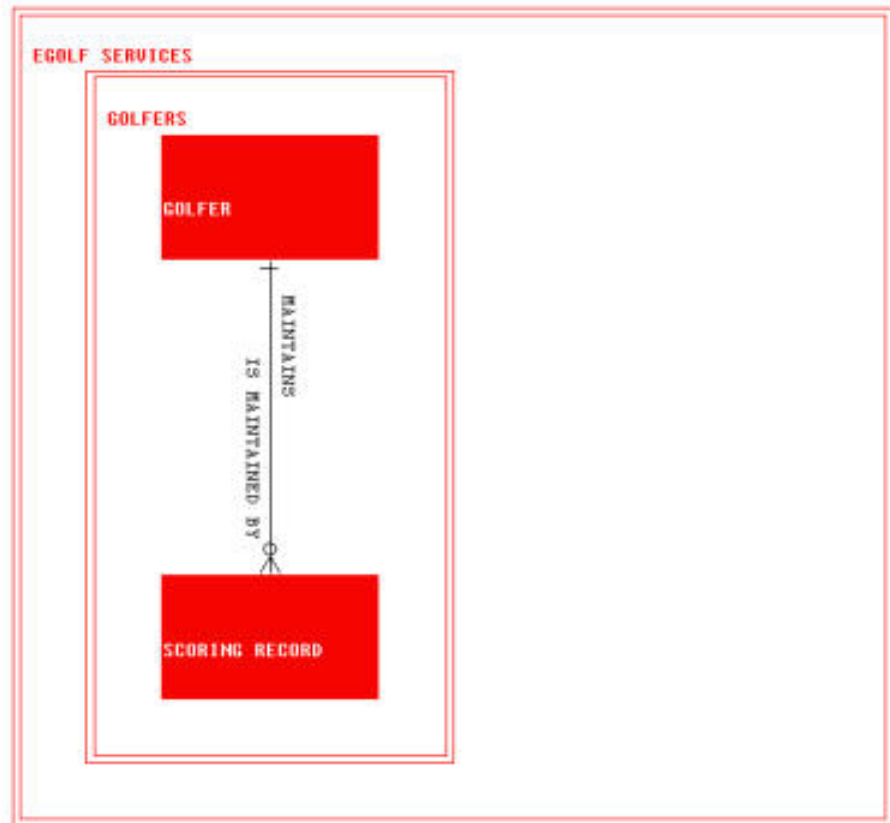
The diagramming convention for relationships is a line between the associated entity types. The naming convention for relationship memberships is typically the active and passive forms of a verb.



What is the relationship between entity type Golfer and entity type Scoring Record?

eGolf Services Relationships

It is the golfer's responsibility to update his or her scoring record. From the point of view of the golfer, this can be expressed as, Each Golfer maintains Scoring Record. From the point of view of the scoring record, it would be, Each Scoring Record is maintained by Golfer. Our data model can look like the following example:



Cardinality and Optionality within Relationships

Relationship memberships are further defined by cardinality and optionality.

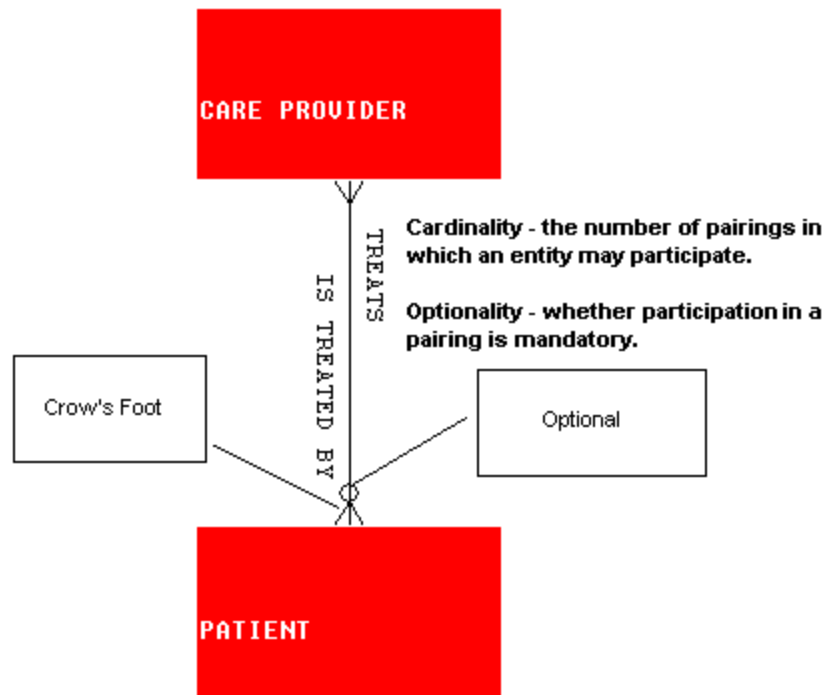
Cardinality

Determines the number of entities on one side of the relationship that can be joined to a single entity on the other side.

Optionality

Specifies if entities on one side must be joined to an entity on the other side.

In the following example, each Care Provider may optionally treat one or more patients. The optionality of the relationship is indicated by the o on the relationship line directly at the top of the crow's foot. The reason the relationship is optional is that we can have a new care provider that has yet to be assigned to a patient. The crow's foot indicates the one or more cardinality.



From the point of view of the patient, one or more Care Providers always treats each Patient. Notice the crow's foot without the o on the line next to the Care Provider. This relationship membership is mandatory (at least for this business), in that a Patient must be assigned to some Care Provider. This is known as a many-to-many relationship.

Note: Other objects within CA Gen support optionality and cardinality. This section deals only with their function in regard to relationships.

Examples of Relationships

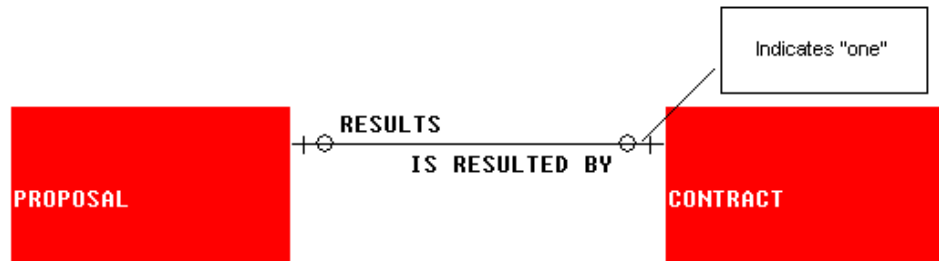
Types of relationships include one-to-one, one-to-many, and many-to-many.

One-to-one Relationship Example

In this example, we have proposals and contracts.

- Each proposal sometimes results in one contract.
- Each contract sometimes is the result of one proposal.

This is an example of a one-to-one optional relationship.



Note: The line crossing the relationship line indicates one.

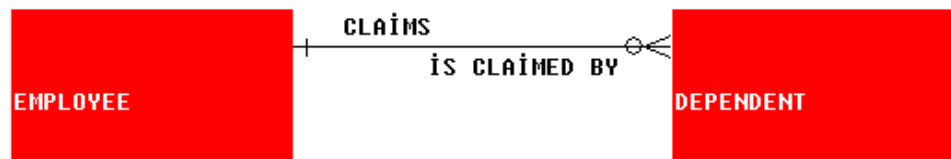
One-to-many Relationship Example

In this example, we have employees and dependents.

- Each employee sometimes claims one or more dependents.
- One employee always claims each dependent.

This is an example of the most common relationship, a one-to-many optional-on-the-many-side relationship.

Note: The crow's foot indicates the "many" and the "o" on the line indicates that the relationship is optional.

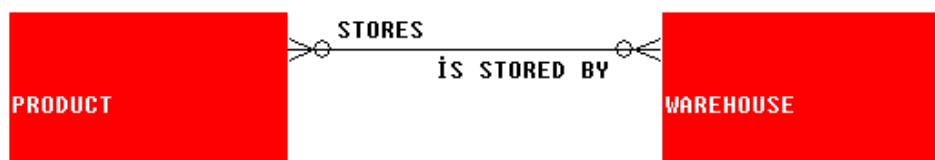


Many-to-many Relationship Example

In this example, we have products and warehouses.

- Each product sometimes is stored in one-or-more warehouses.
- Each warehouse sometimes stores one-or-more products.

This is an example of a many-to-many relationship.

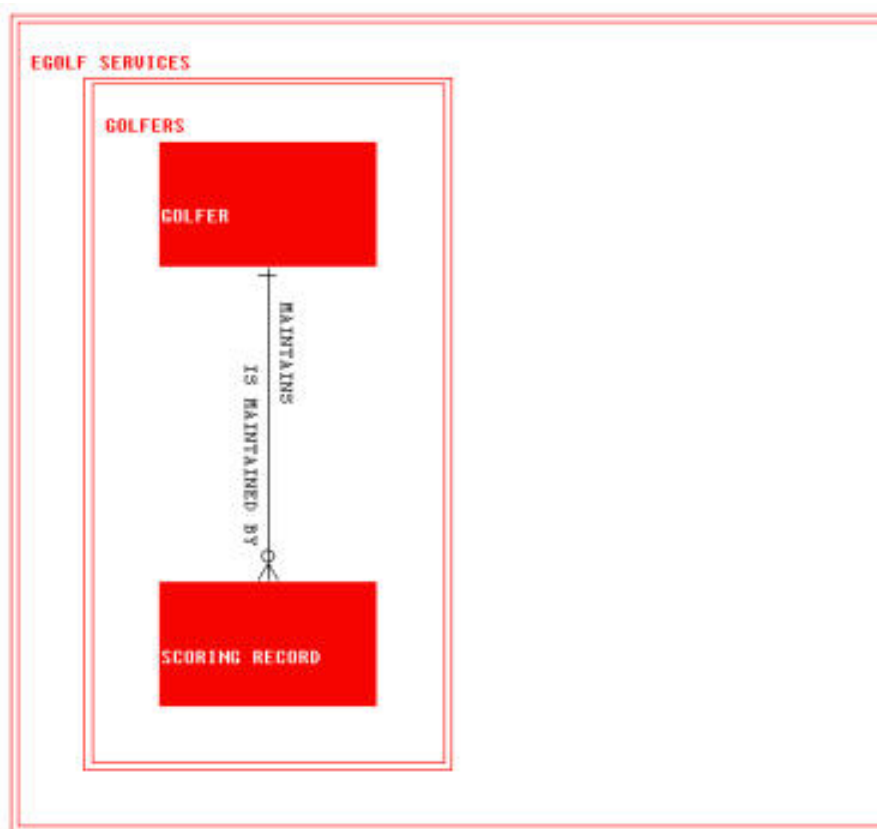


What would be the relationship membership cardinality and optionality for Golfer maintains Scoring Record?

eGolf Services Relationship Cardinality and Optionality

A golfer needs at least five and preferably 20 scoring records to determine their handicap index. Therefore, a golfer maintains one or more scoring records. In addition, a golfer has to register first before logging in to use the system. Therefore, a golfer can exist, even if it is for only a short period, without any scoring records. From the point of view of the golfer, then, Each Golfer sometimes maintains one or more Scoring Records.

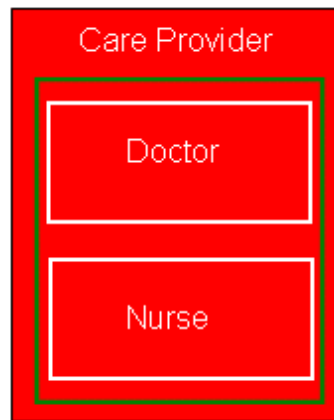
Scoring records are always for one golfer, so from the point of view of the scoring record, one Golfer always maintains Each Scoring Record. Our data model now looks like the following example:



Entity Subtypes

An entity subtype is a collection of entities of the same entity type to which a narrower definition and additional predicates apply. That is to say, entity types can be sub-typed, if there are entities of that type which have unique definitions, attributes, or relationships.

In the following example, the entity type Care Provider can be sub-typed into Doctors and Nurses. There should be unique attributes and relationships that entities of the Doctor sub-type has that entities of the Nurse subtype does not and vice-versa. Additionally, there should be a valid business reason for sub-typing. A valid business reason would exist if there were additional business processes involved with one sub-type and not the other.

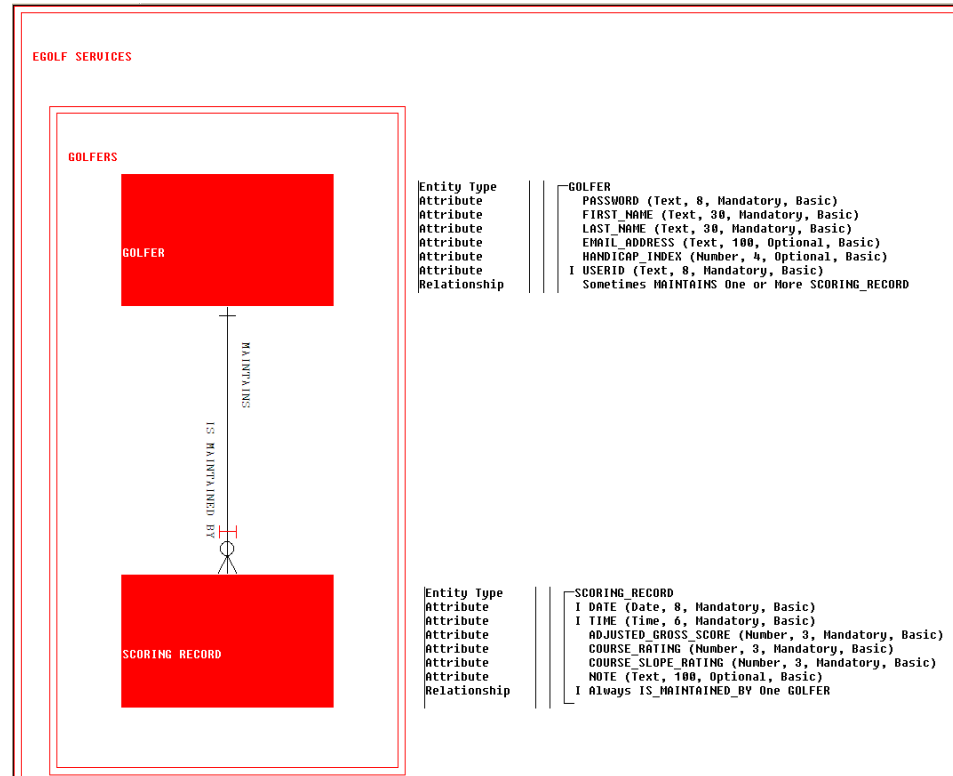


A collection of Entities of the same Entity Type to which a narrower definition and additional predicates apply

The eGolf Services data model does not need subtypes.

eGolf Services Data Model

The following is the eGolf Services solution.



Lesson Activity

In this exercise, we will do the following:

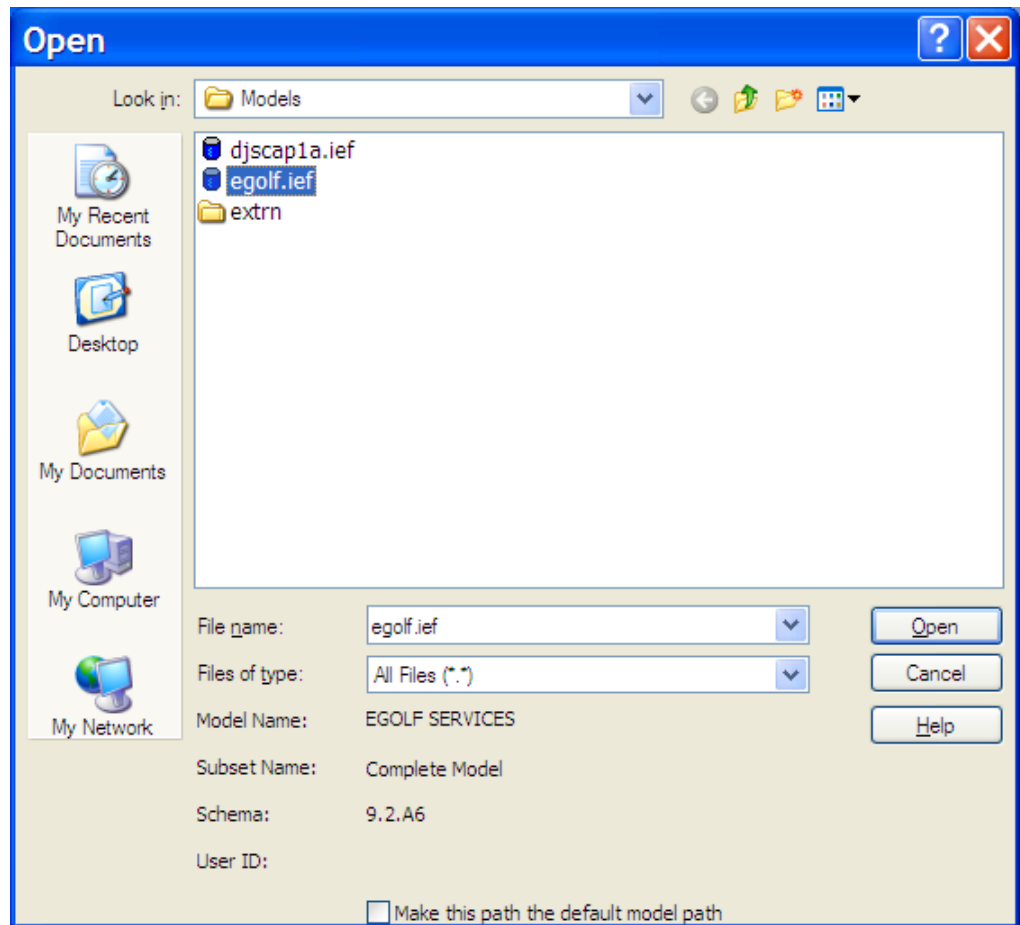
- Open the eGolf Services model
- Build the eGolf Services data model as specified earlier
- Review the data model using the Data Model List
- Review the data model using the Data Model Browser
- Close the data model diagrams

Open the eGolf Services (egolf) Model

Follow these steps:

1. After opening the Toolset, select **Model** from the Menu Bar, and select **Open Model**.

CA Gen displays the Open dialog:

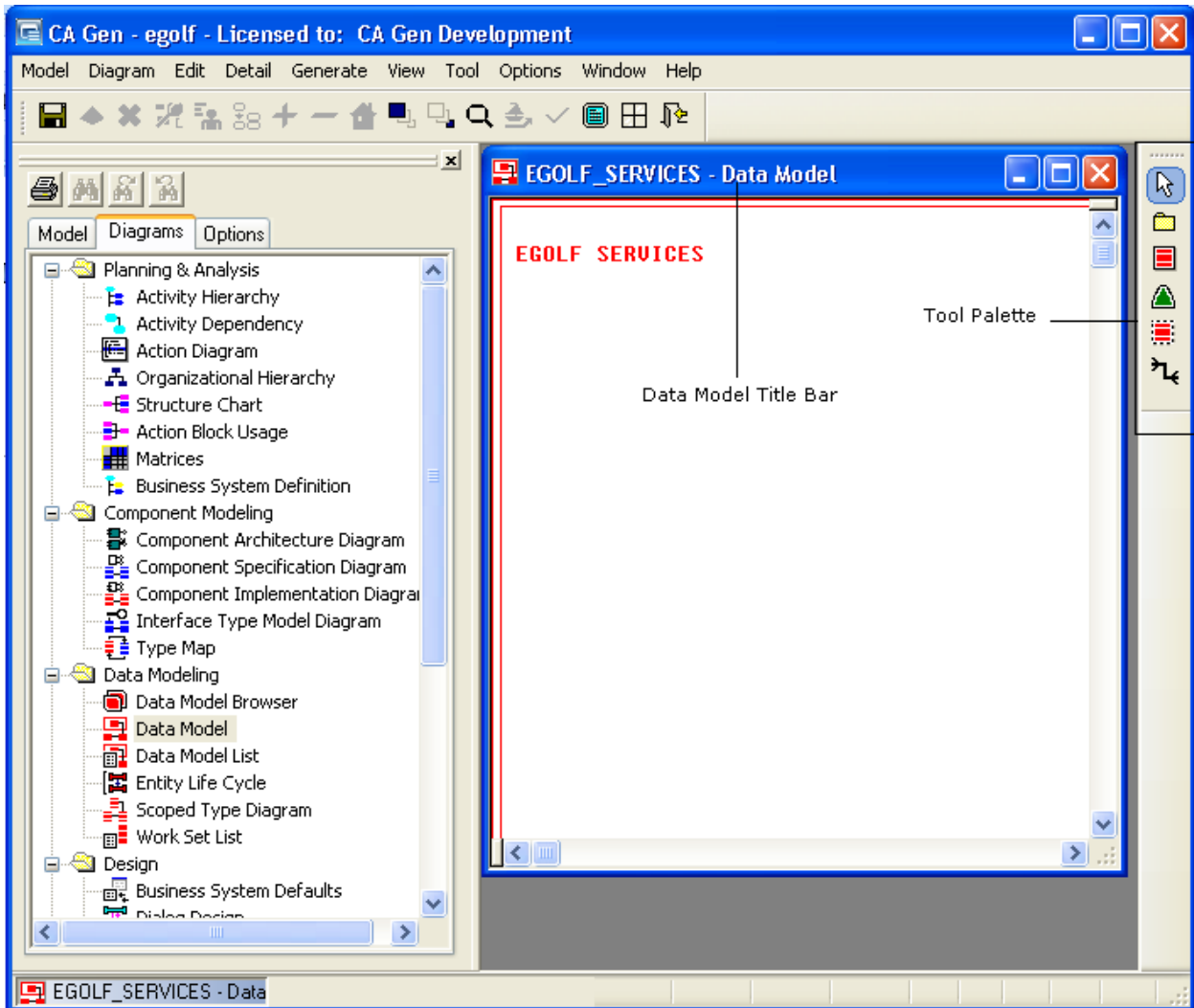


2. In the Model Open dialog, select **egolf.ief** and select **Open**.
3. On the Menu Bar, select **Model, Settings, Single Add**.

Build the eGolf Services Data Model

Follow these steps:

1. Double-click **Data Model** in the tree control.





2. Double-click anywhere in the data model title bar to maximize the data model diagram.

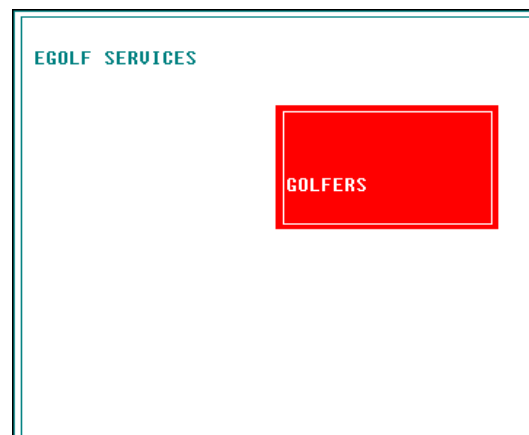
Note: Since we have a diagram open, we also have a tool palette available for that diagram.

Add Golfers Subject Area

Follow these steps:

1. To add the Golfers subject area to the EGOLF SERVICES root subject area, select the **Add Subject Area...**  icon on the tool palette and use the crosshair  to select inside the root subject area which will place the subject area at that location.
2. In the new object's Properties dialog, type **Golfers** in the Subject Area entry field and click **OK**.

Your data model should now look like the following example:




Note: If you mistyped the name of the subject area, double-click the subject area to open its properties panel to enter the correct name.

Add Golfer and Scoring Record Entity Types

To add the golfer and scoring record entity types, you must complete these steps:

- Add the Golfer
- Add the Scoring Record
- Move the entity types
- Create the relationship between the golfer and the scoring record

Follow these steps:

1. To add the Golfer and Scoring Record entity types, use the **Add Entity Type**  icon on the tool palette and use the crosshair to click on the **Golfers** subject area.
2. In the Entity Type (new object) Properties dialog, type **Golfer** in the Type Name field and select **OK**.


3. Do the same to add Scoring Record.

If you make a mistake, here are some tips to correct it:

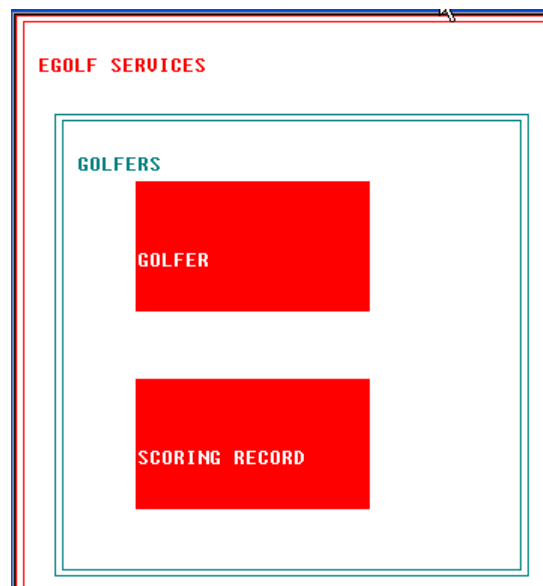
- If you mistyped the name of an entity type, double-click the entity type to open its properties panel and type the correct name.
- If you placed the entity type in the wrong subject area, resize the destination subject area to ensure it is large enough to accept the new entity type, and drag and drop the entity type to the correct subject area.
- To resize the subject area, select the subject area, and move the cursor across its boundaries until it turns into a double-sided arrow. Click-and-drag the boundary to the desired size. When moving an entity type from one subject area to another, the cursor changes to the following icon:



to indicate you are changing the parentage of the entity type.


- You can also delete the entity type by selecting it and clicking the **Delete...**  icon on the Toolbar.

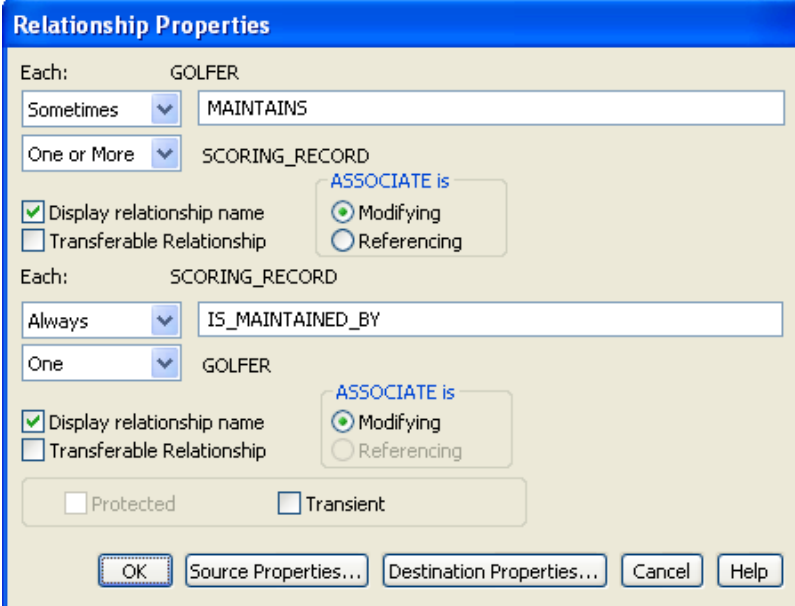
4. Resize the subject area and move the entity types as shown in the following illustration:



To resize the subject area, follow these steps:

- a. Select the subject area
- b. Move the cursor across its boundaries until it turns into a double-sided arrow
- c. Click-and-drag the boundary to the desired size

5. To add the relationship between Golfer and Scoring Record:
 - a. Select the **Join...**  icon on the tool palette
 - b. Select the **Golfer** entity type
 - c. Select the **Scoring Record** entity type
6. Complete the Relationship Properties dialog as in this illustration and click **OK**:



Relationship Properties

Each: GOLFER

Sometimes MAINTAINS

One or More SCORING_RECORD

☒ Display relationship name

☐ Transferable Relationship

ASSOCIATE is

☒ Modifying

☐ Referencing

Each: SCORING_RECORD

Always IS_MAINTAINED_BY

One GOLFER

☒ Display relationship name

☐ Transferable Relationship

ASSOCIATE is

☒ Modifying

☐ Referencing

☐ Protected ☐ Transient

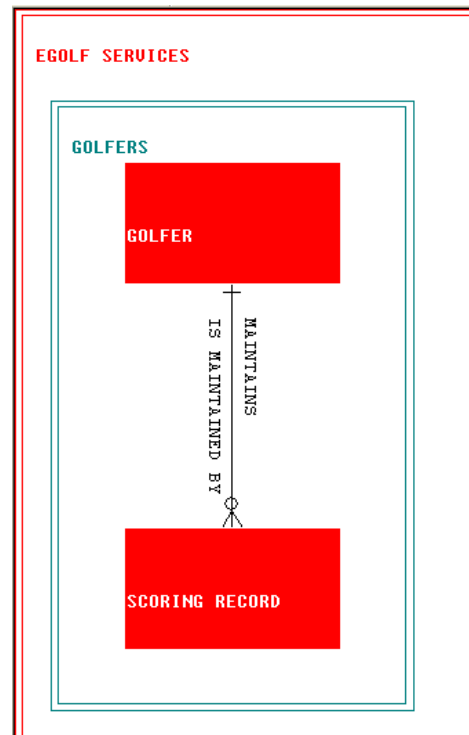
Note: You must type **MAINTAINS** and **IS_MAINTAINED_BY** in the fields.

These settings create the following relationship between the GOLFER and SCORING RECORD:

- Each GOLFER sometimes maintains one or more SCORING RECORD
- Each SCORING RECORD always is maintained by one GOLFER

If you make a mistake, double-click the relationship to open the Relationship Properties dialog to make changes.

Your diagram should look similar to the following illustration:



Add Attributes to Golfer and Scoring Record

Follow these steps:

1. To add attributes more efficiently, turn the Multiple Adds feature on. To do so, on the Menu Bar select **Model, Settings, Multiple Adds**.
2. To add the Golfer attributes, select the **Golfer** entity type, click the **Attributes** icon on the Tool Bar bringing up the Attribute List diagram, and select the **Add...Attribute View** icon on the Tool Bar.


Note: The Add Attribute View icon and the Attributes icon are the same.

- In the (new object) Properties dialog, enter the following information for each attribute.

Note: The (new object) Properties dialog remains open after you click **OK** after adding each attribute because Multiple Adds is turned on.

You should be able to see each attribute added in the panel in the background. After adding the last attribute for Golfer click **Cancel**.

Attribute Name	Category	Domain	Length	Decimal Places	Optional	Case Sensitive	Varying Length
Userid	Basic	Text	8	N/A			
Password	Basic	Text	8	N/A		✓	
FirstName	Basic	Text	30	N/A		✓	
Last Name	Basic	Text	30	N/A		✓	
Handicap Index	Basic	Number	4	1	✓	N/A	N/A
Email Address	Basic	Text	100	N/A	✓	✓	✓

- To close the Attribute List Diagram listing the attributes for Golfer, select the **close** button  in the upper right hand corner of the Toolset, diagram at the end of the Menu Bar (not the one which may be above it which closes the toolset entirely).

If you make a mistake, select the entity type, select the **Attributes** icon on the Menu Bar, and double-click the attribute in the Attribute List Diagram to open the properties dialog for the attribute. To delete an attribute, select the attribute and select the **Delete** icon on the Menu Bar.

- Add the following attributes to Scoring Record.

Attribute Name	Category	Domain	Length	Decimal Places	Optional	Case Sensitive	Varying Length
Date	Basic	Date	N/A	N/A		N/A	N/A
Time	Basic	Time	N/A	N/A		N/A	N/A
Adjusted Gross Score	Basic	Number	3	0		N/A	N/A

Attribute Name	Category	Domain	Length	Decimal Places	Optional	Case Sensitive	Varying Length
Course Rating	Basic	Number	3	1		N/A	N/A
Course Slope Rating	Basic	Number	3	0		N/A	N/A
Note	Basic	Text	100	N/A	✓	✓	✓

6. Two of the ScoringRecords attributes names are longer than 15 characters, and can be a problem when targeting JDBC as we will do for our Web server. In the list of attributes for Scoring Record:
 - a. Double-click **ADJUSTED_GROSS_SCORE**
 - b. Uncheck the 'Set TD name to the attribute name checkbox'
 - c. Change the TD Name to **ADJ_GROSS_SCORE**
 - d. Do the same for **COURSE_SLOPE_RATING**

Review the Data Model Using the Data Model List

Follow these steps:

1. On the Menu Bar, select **Tool, Analysis, Data Model List**.
2. On the Menu Bar, select **View, Expand Diagram**.

Review the objects in the Data Model Diagram. Double-click the objects to change their properties. You can add new data model objects or delete existing objects.

Review the Data Model Using the Data Model Browser






Follow these steps:

1. On the Menu Bar, select **Tool, Analysis, Data Model Browser**.
2. Select **Golfer** from the upper-left panel in the Data Model Browser dialog.

Note: The other panels are populated. Double-click an object to change their properties. You can add new data model objects or delete existing objects.

Close the Data Model Diagrams

Follow these steps:

1. To close the Data Model Browser, select the **Close** button  in the upper right hand corner of the Data Model Browser dialog.
2. To close the Data Model List, select the lower **Close** button  in the upper right hand corner of the diagram at the end of the menu bar.
3. To close the attribute list, select the lower **Close** button  in the upper right hand corner of the diagram at the end of the menu bar.
4. To close the Data Model Diagram, select the lower **Close** button  in the upper right hand corner of the diagram at the end of the menu bar.
5. Save your model using one of the following methods:
 - Select the  icon from the Toolbar.
 - On the Menu Bar, select **Model**, and then select **Save Model**.
 - Press **Ctrl+S**

Develop the Process Model

The following sections describe how to develop a process model.

Lesson Objectives and Time Allotment

After this lesson, you will be familiar with:

- The two basic modeling objects used in an Activity Hierarchy Diagram:
 - Functions
 - Processes
- The Activity Hierarchy for the eGolf Services business

Allow approximately 20 minutes to complete this lesson.

Activity Hierarchy Diagram

While the Data Model Diagram describes the things, which are of interest to a business, the Activity Hierarchy Diagram describes the actions the business takes with those things. For instance, a business can be interested in something called an Order. In fact, the more orders a business has, the better. However, what does the business do with Orders? Typically they Accept Order, Approve Order, Pick Order, Pack Order, Ship Order, and Invoice Order. In addition, only on rare occasions, they have to Cancel Order. These business activities are documented on an Activity Hierarchy Diagram.

The term activity is somewhat generic. It can mean either a function or a process. However, the terms function and process have specific meanings in CA Gen, and we will look at them in a moment.

Activity analysis involves the continued decomposition of activities until the analyst has identified the lowest-level processes of interest to the business. Functions decompose into either lower-level functions or higher-level processes. Higher-level processes decompose into lower-level processes. The decomposition ends when the analyst has uncovered all of the lowest-level processes of interest to the business. These lowest-level processes are known as elementary processes.

For the purpose of this tutorial, we started doing Data Analysis. However, we could have begun doing Activity Analysis. In fact, since one complements or confirms the other, they should be done in parallel.

Functions

Functions are groupings of activities that deal with the major areas of interest to a business. They are broad business activities that take place continuously.

In our XYZ Health Clinic example, there were at least two natural areas of interest to our business, Care Providers and Patients. Therefore, we have two corresponding functions—Care Provider Management and Patient Care—in which we can group all of the business activities pertaining to each area of interest.

In the Entity Subtypes section, we contemplated sub-typing the Care Provider entity type into the sub-types Doctor and Nurse. One of the conditions for sub-typing calls for different business activities for each subtype. If we needed to sub-type Care Provider into Doctor and Nurse, we would want to decompose Care Provider Management into lower-level functions covering each sub-type to contain those specific activities associated with each.



Examining the XYZ Health Clinic Data Model and Activity Model, you should begin to see some parallels between the two. In fact, this concept is known as parallel decomposition. The lowest-level functions should correspond with the lowest level subject areas or, in some cases, subtypes.

There are different techniques for activity decomposition, including:

Value Chain Analysis

Generally used for the first level of decomposition, not demonstrated here

Specialization

Demonstrated with the previous Care Provider specialization

Life Cycle

Used for the first level processes discussed next

Steps to Complete

Lower-level processes used, if necessary, to move an entity from one life cycle state to the next.

Given this brief introduction to Parallel Decomposition, what functions would exist beneath the eGolf Services root function?

eGolf Services Functions

For now, we will only to deal with Golfers. We need a corresponding business function to group all of the activities we are going to have for Golfers. Our eGolf Services root function can decompose into one other function called Golfer Management.



The naming convention for business functions is usually a noun, followed by the noun form of a verb, as in:

- Customer Service
- Order Processing
- Manufacturing
- Human Resources

Activity decomposition follows the rules of structured outlining. Normally, a function would decompose into at least two other functions, or two other processes. We are only modeling part of the business. If we were modeling the entire business, we would likely have other functions dealing with things like Advertising and Courses.

Processes

A Process is a defined business activity whose executions are identified in terms of the input and output of specific entities or data about specific entities.

Processes are lower-level activities. Unlike functions, which are large, broad-based activities that happen continuously, as in the case of the Manufacturing function, processes have a definable start and end.

Processes also accomplish work. From a modeling standpoint, work generally means they must create, update, or delete entities of an entity type. For example, in the real world, your business can have someone who sits at a counter and takes orders. Before they take an order, they have nothing. After they take an order, they have a completed order. In the modeling world, we would have a process called Take Order. After an execution of that process, we would have a new entity added to our Order entity type.

The first level of processes under a primitive function focus on one lifecycle state of the subject of that function. A primitive function corresponds to a primitive subject area, that is, a subject area that only has entity types within it.

Let us look at the lifecycle of a Doctor subtype within the XYZ Health Clinic. In this business, we first interview the doctor to evaluate their credentials. If we like them, we present them with a contract to work at our health clinic. Periodically, we evaluate their performance, and when we are through with them, we terminate them. Sound familiar? Every business object has at least two lifecycle states, a Creation state and a Termination state. Most have more. Think about the activities required in planning for acquiring the objects, usage of the objects, and ultimately disposal of those objects.



Given the brief introduction to previous process modeling, take a moment to determine the processes associated with planning for, acquiring, usage, and disposal of Golfers. When you are finished, move on to the next section.

eGolf Services Processes

While it is not clear from the interview, golfers are acquired when they register for the service (the creation state). The registration information is used when they login to enter their scoring records and we calculate their Handicap Index. While it is not specifically mentioned, we know that sometimes we have to remove golfers (the termination state) as well. Accordingly, our activity hierarchy looks something like this:



The diagramming notation for processes is similar to functions, only that the color of processes is blue, while the color of functions is cyan. Processes are traditionally named with a verb and a noun, with the noun usually being an entity type or attribute from the data model.

Elementary Processes

The purpose of Activity Decomposition is to identify all of the elementary processes associated with a selected portion of the business. Elementary processes are the smallest unit of business activity that has meaning to the business and when complete, leave the business in a consistent state.

In the following example, Contract Doctor is a process that has a start, an end, and does useful work. However, it is not the lowest-level process of meaning to this business. Contract Doctor can be broken down into lower-level processes, each of which have a start, an end, and perform meaningful work. In this case, the steps to complete the higher level of work include:

- Preparing the contract
- Delivering the contract to the doctor
- Waiting for the doctor's acceptance

Each step performs meaningful work for the business, can and do happen at different times and are performed by different people, and leave the business in a consistent state

The phrase "leave the business in a consistent state" has several points to it, but at a minimum, it means that none of the optionality rules as depicted in the data model are violated by an execution of the process. In other words, if an attribute value or relationship membership is mandatory, then that attribute will have its value set or that relationship membership will be created by the end of the execution of the process.

When processes cannot be broken down into lower-level processes without violating some of the guidelines discussed in the previous paragraph, they are considered elementary processes and marked as such in their properties. This causes a dashed line to appear on them as shown in the following example.



The process action diagram (PAD) will eventually contain pseudo-code specifying the business rules associated with that business activity.

Note: Each elementary process has an action diagram associated with it.

eGolf Services Elementary Processes

Are the processes we have identified so far for the eGolf Services business elementary?

To determine whether each process is an elementary process, we would have to examine each one with an eye towards the following guidelines:

- Does it perform meaningful work? Does it create, update, or delete entities of some entity types?
- Does it leave the business in a consistent state? At a minimum, it must not violate the optionality attribute and relationship rules as depicted in our data model.
- From the time it begins execution, to the time it finishes, does it generally happen at a single point in time?
- From the time it begins execution, to the time it finishes, does it generally happen at a single location?
- Can it be broken down further while still meeting all the previous guidelines?

Let's examine the processes we already defined in our eGolf Services model:

Register Golfer

Creates an occurrence of Golfer, and assuming it sets all of the mandatory attributes of Golfer, leaves the business in a consistent state. The entire thing happens at one location and time, and cannot be further broken down without violating one of the previous guidelines. Therefore, it is probably an elementary process.

Login Golfer

This process does not create, update, or delete anything. It only performs a read to see if a Golfer is registered. Therefore, someone can argue it does not do meaningful work and therefore is not even a process. Another argument is that Login Golfer is a security function, and therefore should be dealt with in Design. However, this would be an exception to the rules. Processes that only do reads but are so intrinsic to the business, as security is in a Web application, can be considered processes. In this case, it can be considered elementary since it would meet the remaining guidelines as well.

Enter Scoring Record and Calculate Golfer Handicap Index

One can argue that these should be one process instead of two. After all, the interview says when they enter their scores we will recalculate their handicap index. However, the question is not "Should these be one", but rather, "Can there be two". Can each be executed independently at different points in time and at different locations, - can each perform meaningful work, and can each leave the business in a consistent state? It can be argued that they can. In fact, we cannot even calculate a valid handicap index until a golfer has entered at least five scores. Do we want to make the golfers wait until they have at least five scores to enter? On the contrary, we want them to visit our Website early and often to run up our hit count. Each can be executed independently and meet the remaining guidelines assuming that when we created a Scoring Record we associated it to the Golfer which maintains it. Therefore, each would be an elementary process.

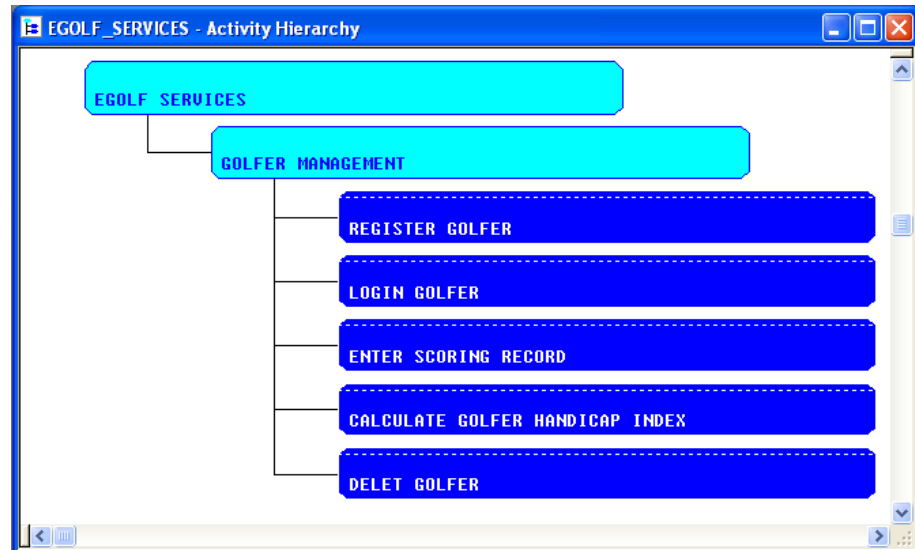
Delete Golfer

Deletes an occurrence of Golfer. Assuming it deleted all of the related occurrences of Scoring Record, then it would be considered an elementary process as well.

Each of these processes should be marked as elementary processes. When marked, each would then show the dashed line across them and each will then have a process action diagram associated with them.

eGolf Services Activity Hierarchy Diagram

The following is the eGolf Services solution.



Lesson Activity

In this exercise, we will build the Activity Hierarchy diagram as specified earlier by following these steps:

- Open the Activity Hierarchy Diagram
- Add the Golfer Management Function
- Add Elementary Processes

Build the eGolf Services Activity Hierarchy

To build the services activity hierarchy, we must open the activity hierarchy, add the golfer management function, and add the elementary processes.


Open the Activity Hierarchy Diagram

Follow these steps:

1. With the `egolf.ief` model open in the Toolset, double click **Activity Hierarchy** in the tree control.
2. Double click anywhere in the Activity Hierarchy Diagram title bar to maximize the Activity Hierarchy Diagram to the allotted space.

Add the Golfer Management Function

Follow these steps:


1. To add the Golfer Management function to the **EGOLF SERVICES** root function, select the EGOLF SERVICES root function and select the **Add Function**  icon from the Tool Bar.
2. In the (new object) Properties dialog, enter **Golfer Management** in the Function entry field and select the **OK** push button. If you still have multiple adds turned on, you will need to select **Cancel** to close the (new object) Properties dialog.

If you mistyped the name of the function, double click the function to its properties panel to enter the correct name.

If you added it as a process (blue) instead of as a function (cyan), select the root function **EGOLF SERVICES**, select **Edit** from the menu bar, and select **Modify**.

Add Elementary Processes

Follow these steps:

1. To add the elementary processes to the GOLFERS MANAGEMENT function, select the **GOLFERS MANAGEMENT** function and then select the **Add Process**  icon from the Tool Bar.
2. In the (new object) Properties dialog, enter **Register Golfer**, select the check box for **Elementary**, and click **OK**.
3. If you have multiple adds still turned on, the (new object) Properties dialog remains open and you can continue to add the remaining processes. If not, select the **Add Process** icon again. Add the remaining elementary processes:

- **Login Golfer**
- **Enter Scoring Record**
- **Calculate Golfer Handicap Index**
- **Delete Golfer**

If you mistyped the name of a process or forgot to select the elementary check box, double click the process to open its properties panel. If you added them all as functions (cyan) instead of processes (blue), select their parent function **Golfer Management**, select **Edit** from the Menu Bar, and select **Modify** to convert them all to processes.

4. While methodologically order is not important, aesthetically it is nice to have the processes in logical sequence. To rearrange the order of the items in the diagram, click and drag them to the desired location. If you move them to an entirely different parent, the cursor switches to the change parent icon, identical to what we saw in the data model diagram.

5. Save your model.
6. Close the Activity Hierarchy diagram.

Confirm the Process and Data Model

Lesson Objectives and Time Allotment

After completing this lesson, you will be familiar with four techniques for confirming the results of your analysis:

- Walk Throughs
- Customer Reviews
- Consistency Checks
- Matrices

Allow approximately 40 minutes to complete this lesson.

Confirmation

Model Confirmation is a series of checks to ensure the model is complete, correct, and stable, terms described in the following list:

Complete

None of the business requirements have been missed or omitted

Correct

The requirements are conforming to fact

Stable

The model is developed to mitigate future changes

These checks consist of manual and automated methods.

Checks used to ensure the model is complete, correct and reasonably stable

- Manual Methods
 - Walk Throughs
 - Customer Reviews
- Automated Methods
 - Consistency Checks
 - Matrices

Manual Methods

Walk Throughs and Customer Reviews are the manual methods:

Walk Throughs

An elementary process has a process action diagram (PAD) associated with it that will eventually contain pseudo-code, documenting specific business rules associated with that business activity. From the pseudo-code, the tool generates source code which will then be compiled into executable code. It is a good structured programming technique to review, or walk through, this pseudo-code with another knowledgeable analyst to identify errors or omissions early in the development lifecycle.

Customer Reviews

The data and activity models should represent actual things that the business is interested in and the tasks that the business accomplishes with the data and activity models. Formal reviews of the model with the user community helps confirm nothing was missed. An analyst should be able to stand up in front of a group of knowledgeable users and adequately explain it to them. If the general reaction is positive, the project is likely on track and can move forward. If the users fail to understand the review, more analysis work is probably required.

Automated Methods

Consistency Checks and Matrices are automated methods:

Consistency Checks

CA Gen includes an automated facility called Consistency Check that applies a set of rules to a model, a subset of a model, or a specific object in the model to evaluate its consistency and completeness. We will look at one use of Consistency Check to evaluate the consistency and completeness of our data model.

Matrices

CA Gen includes a set of more than 35 standard matrices that help record and evaluate numerous facts about the business. These matrices can help during the initial assessment of a project, when defining an information architecture, technical architecture, and business system architecture, when making a current environment assessment, and when defining business systems. CA Gen supports creating your own custom matrices as well. We will look at one use of the Matrix Processor to examine the interaction between elementary processes and entity types, looking for gaps in our analysis.

Lesson Activity

In this exercise, we are going to:

- Perform a consistency check on the data model
- Correct any problems found by the consistency check
- Examine the entity type versus elementary process matrix and make improvements to the model if necessary

Perform a Consistency Check

With Consistency Check you can check an entire model or specific objects in the model. Before performing the consistency check, you must set an appropriate consistency check level.

As a model progresses from Analysis, to Design, to Construction and Test, it becomes increasingly detailed. Performing a comprehensive test on the data model in the Analysis stage identifies more problems than we need to worry about at this point in the development lifecycle. To keep this from happening, we want to set a specific consistency check level.


Set Consistency Check Level

Follow these steps:

1. Open the egolf.ief model.
2. From the main Menu Bar, select **Model, Settings, Consistency Check Options**.
3. In the Consistency Check Options dialog:
 - a. Select the **Level** tab and click the **Analysis** radio button.
 - b. Select the **Severity Reported** tab and the **All Warnings and Errors** radio button.
 - c. Click **OK**.

Perform the Consistency Check

Follow these steps:

1. Open the Data Model diagram and select the **EGOLF SERVICES** root subject area.
2. On the Tool Bar, select the **Check**  icon.

3. CA Gen issues a report similar to the following illustration:

Consistency Check

Model Name: EGOLF SERVICES

Subset: ALL

Mar. 23, 2006 12:15

Level: Analysis

Reporting: Warnings, Severe Warnings, Errors, Fatal Errors

Subject Area: GOLFERS

Entity	Type	SCORING_RECORD
ERROR:	ICCHL03E	Each persistent entity type with occurrences must have an identifier.
WARNING:	ICCHL12W	A persistent entity type's maximum number of occurrences should be defined.
WARNING:	ICCHL13W	A persistent entity type's average number of occurrences should be greater than 0.
ERROR:	ICCHL18E	A PERSISTENT entity type must have either a BASIC or DESIGNED attribute or two

Entity	Type	GOLFER
ERROR:	ICCHL03E	Each persistent entity type with occurrences must have an identifier.
WARNING:	ICCHL12W	A persistent entity type's maximum number of occurrences should be defined.
WARNING:	ICCHL13W	A persistent entity type's average number of occurrences should be greater than 0.
ERROR:	ICCHL18E	A PERSISTENT entity type must have either a BASIC or DESIGNED attribute or two

Relationship GOLFER MAINTAINS SCORING_RECORD

WARNING:	ICCRE02W	A relationship should have estimated cardinality defined.
WARNING:	ICCRE03W	A relationship should have expected optionality defined.

number of ERRORS: 4 , **number of WARNINGS:** 6

*** End of report ***

Examine the Consistency Check Report

This report has two errors and six warnings.

Errors

Indicate conditions you must correct before the objects in the model can move forward to the next stage of development

Warnings

Indicate conditions that may cause problems later, but do not prevent those objects from progressing to the next stage of development

Severe Warning

Indicate conditions that are more severe than warnings in that they should be addressed before continuing to the next stage of development but that do not deny you the ability to proceed anyway. These types of warnings may cause other problems that are difficult to diagnose from the information given at the time (such as during code generation or compilation) so they should be addressed as soon as possible.

Examining our Consistency Check Report, we see that at the top left it shows the model name, the subset name, and the date and time the report was run. Level indicates the consistency check level we selected under the model settings. In this case, BAA represents the Analysis level we selected earlier. BAA stands for Business Area Analysis. Reporting indicates the severity of messages to report, also selected under the model settings. After that, it shows the object that was selected for the consistency check, in our case the EGOLF SERVICES root subject area. Finally it shows the objects within that subject area that had problems. We are going to address each of the problems identified in this report.

Looking at the entity type GOLFER, we have an error and two warnings. The error states that each entity type must have an identifier defined for it. An *identifier* is a way to uniquely identify each entity of an entity type. For example, your business employs many employees. How does your business uniquely identify each employee? Your company probably identifies employees with something like a unique employee id. A secondary identifier can be their social security number. Likewise, we need a way to uniquely identify every GOLFER. In our case, each GOLFER is going to have a unique User ID. Therefore, we will declare the User ID as the identifier for GOLFER, which will eliminate the error.

GOLFER also has two warnings. These warnings refer to the minimum, maximum, and average number of GOLFERS the business expects to have. The minimum number of golfers would be zero; since this is the default value, we receive no warning. However, the maximum and average values default to zero as well, and the eGolf Services business hopes to have more golfers than that. So the question is, what is the maximum number of golfers the business expects at any point in time and what's the average number? We will assume that marketing research has indicated that we will have a maximum number of 100,000 golfers with an average of 75,000. We will enter these values in after examining the rest of our errors and warnings.

Looking at the relationship membership GOLFER MAINTAINS SCORING RECORD, or more specifically EACH GOLFER SOMETIMES MAINTAINS ONE OR MORE SCORING RECORD, we have two warnings. The warnings state that the estimated cardinality and optionality of this relationship should be defined.

In this case, cardinality deals with the minimum, maximum, and average number of scoring records each golfer will maintain. We know that a golfer needs at least five and preferably 20 scoring records to determine a handicap index. However, we also know that golfers can register without entering any scoring records. Thus, the minimum number of scoring records can be zero, the maximum number 20, and the average five. However, this would only be true if as they entered their 21st scoring record, we deleted their 1st scoring record, maintaining 20 as the maximum number of scoring records maintained by each golfer. However, what if we wanted to keep all of their scores for historical purposes? You need to ask these questions to the users. We will assume the minimum number of scoring records maintained by a single golfer will be zero, the maximum will be 100, and the average will be 20.

Optionality deals with the SOMETIMES aspect of the relationship membership. Essentially, what percentages of golfers sometimes maintain a scoring record? While we know that a golfer can register without ever entering any scoring records, we are anticipating that most will enter at least one scoring record. Therefore, something like 99 percent of the golfers will maintain at least one scoring record.

Entering these values, which we will do shortly, will resolve the relationship membership problems identified previously. However, that relationship also has another relationship membership, EACH SCORING RECORD ALWAYS IS MAINTAINED BY ONE GOLFER that we did not receive any errors or warnings for. What is the cardinality and optionality of this relationship membership? The cardinality in this case is one: each scoring record is maintained by one golfer. In addition, there is no optionality since one golfer always maintains each scoring record.

Finally, SCORING RECORD had similar problems to GOLFER. A golfer will have many scoring records. How can we uniquely identify each scoring record? Each scoring record has the date and time that the game was played. It is unlikely that a golfer can play two games on the same date at the same time, so date and time can be used to uniquely identify each scoring record. However, we can still have two scoring records for the same date and same time played by two different golfers. So to uniquely identify every scoring record, we not only need to know the date and time it was played, but the golfer who maintains it.


What about the average and maximum number of SCORING RECORDS? If we know that we have a maximum number of 100,000 golfers and each will have a maximum number of 100 games, then the maximum number of scoring records would be 10,000,000. The average number of scoring records would be the average number of golfers (75,000) times the average number of games per golfer (5) which is 375,000.

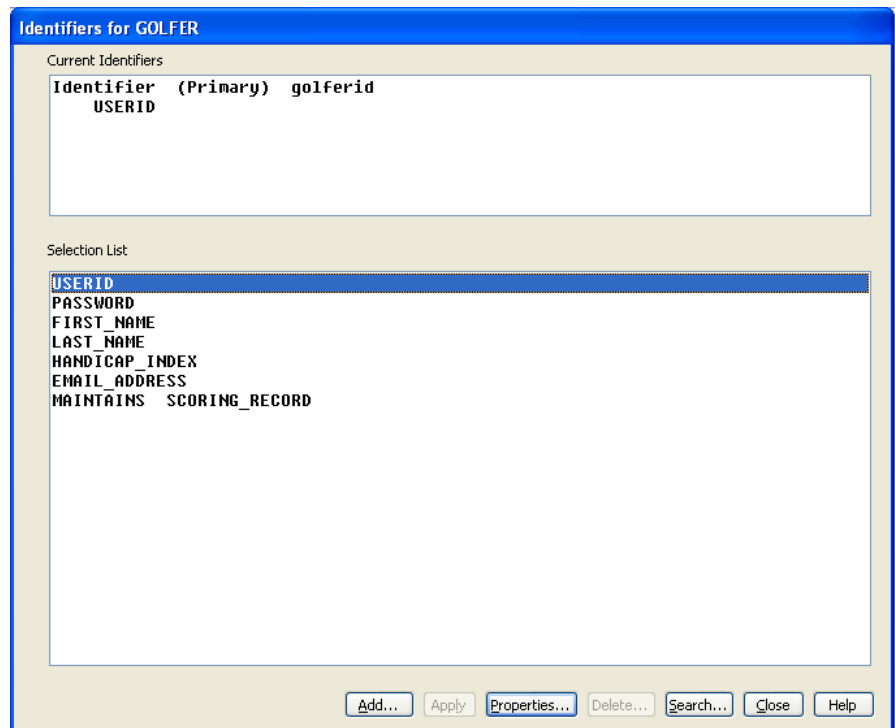
While the previous discussion on entity type and relationship cardinality is accurate, the downstream effect of all this is that eventually we are going to create a database to store all of this data. In addition, the database will be sized based on these numbers. To keep the size of our test database manageable, we will enter significantly smaller numbers into the Toolset.

Correct the Problems

Follow these steps:

1. **Close** the Consistency Check Report, but not the model.
2. In the data model, doubleclick the **GOLFER** entity type.
3. In the Entity Type GOLFER Properties dialog, enter **75** for the Average Expected Number of Occurrences and **100** for the Maximum, and click **OK**.

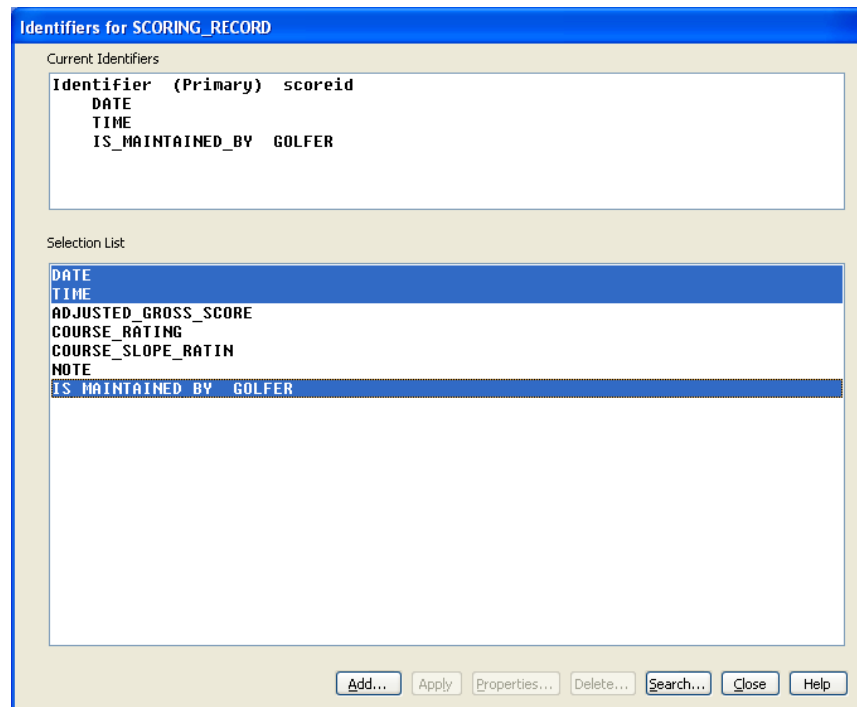
4. While the GOLFER entity type still has focus, select the **Identifiers**  icon on the Tool Bar.
5. In the Identifiers for GOLFER dialog, select **USERID** from the Selection List, and select the **Add** button.
6. In the (new object) Properties dialog, type **golferid** in the Identifier: entry field and click **OK**. If you have multiple adds turned on, select **Cancel** to close the (new object) Properties dialog. The Identifiers for GOLFER dialog should now look like the following illustration:



If you added more than one primary identifier, select the secondary identifier in the upper panel, unselect any selected items in the Selection List by selecting them again, and click **Delete**. If you mistyped the name of the identifier, select the identifier in the upper panel; unselect selected items in the Selection List by selecting them again, and select **Properties** button.

7. Click **Close** to exit the Identifiers for GOLFER dialog.

8. Repeat these steps for the SCORING RECORD entity type. The minimum, average, and maximum expected number of occurrences should be **0**, **375**, and **1000**, respectively. The identifier consists of the attributes **DATE** and **TIME**, and the relationship **IS MAINTAINED BY GOLFER**. Select all three as in the following example, and click **Add**. Name the identifier **scoreid**.



Record the cardinality and optionality of the relationship memberships.

Follow these steps:

1. Double click the relationship line in the data model.
2. In the Relationship Properties dialog, click **Source Properties**.
3. In the Source Properties of MAINTAINS dialog, type **99** for the percentage of time, **1** for at least, **5** for on average, and **40** for at most.
4. Click **OK**.
5. Click **Destination Properties** to review the Destination Properties.
6. **Cancel** out of the Destination Properties of IS MAINTAINED BY dialog, and click **OK** on the Relationship Properties dialog.

Perform Another Consistency Check

Follow these steps:

1. Select the root subject area **EGOLF SERVICES**, and click the **Check** icon on the toolbar.
CA Gen displays the message:
Consistency Check has completed successfully.
2. Click **OK** to continue.
3. Save the model.

Open the Elementary Process uses Entity Type Matrix

Follow these steps:


1. In the tree control, double click **Matrices** under Planning & Analysis.
2. In the Select First Axis panel in the Matrix Selection dialog, select **Elementary Process**.
3. In the Select Second Axis panel, select **uses Entity Type** and click **OK**.

Examine Expected Effects

It can be useful to examine the expected effects of an execution of an elementary process against objects in the data model. This can assist you in finding gaps in your analysis. Expected effects are defined in terms of creating entities of an entity type, reading entities of an entity type, updating entities of an entity type, or deleting entities of an entity type. Collectively, these are often known as CRUD (Create, Read, Update, Delete) actions.

What are the expected effects of an execution of REGISTER GOLFER on the entity type GOLFER? This process is expected to Create an occurrence (entity) of the GOLFER entity type.

Follow these steps:

1. To record this in the matrix, highlight the empty cell at the intersection of **REGISTER GOLFER** and **GOLFER** by selecting it.
2. Select the Cell Values: **C = Create** in the upper-left corner of the matrix, and select the **Add object...**  icon from the Tool Bar.

Note: If you made a mistake adding a value, remove the cell value by selecting the cell, selecting = **Not Referenced** from the upper-left corner of the matrix, and selecting the **Add object...** icon.

What are the expected effects of LOGIN GOLFER? It is expected to Read an occurrence of GOLFER to verify the User ID and Password.

1. To record this in the matrix, highlight the empty cell at the intersection of **LOGIN GOLFER** and **GOLFER**.
2. Select **R = Read Only** from the Cell Values: in the upper-left corner, and select the **Add object** icon on the Tool Bar.

What are the expected effects of ENTER SCORING RECORD? It is expected to Create an occurrence of SCORING RECORD and -, to associate that SCORING RECORD to the GOLFER that maintains it. It must get currency on the GOLFER, and to do that, it has to Read GOLFER.

1. To record this in the matrix, select the empty cell value at the intersection of **ENTER SCORING RECORD** and **GOLFER**.
2. Select **R = Read Only** from the Cell Values: in the upper-left corner, and select the **Add object** icon on the Tool Bar.
3. Select the empty cell value at the intersection of **ENTER SCORING RECORD** and **SCORING RECORD**.
4. Select **C = Create** in the upper-left corner, and select the **Add object** icon on the Tool Bar.


What are the expected effects of CALCULATE GOLFER HANDICAP? It is expected to Read SCORING RECORDs, calculate the handicap index, and Update the GOLFER that maintains them. To update the GOLFER, it must Read the GOLFER first.

1. To record this in the matrix, select the empty cell at the intersection of **CALCULATE GOLFER HANDICAP** and **SCORING RECORD**.
2. Select the **R = Read Only** cell value, and select the **Add object** icon.
3. Select the intersection of **CALCULATE GOLFER HANDICAP** and **GOLFER**. Since R = Read Only is still selected, select the **Add object** icon.
4. To add the update cell value to the same cell, select the **U = Update** cell value and select the **Add object** object icon.

Note: Only the update is indicated in the cell value, not the read. If there are multiple CRUD actions for a single cell, it shows the value representing the greatest degree of involvement.

In order from lowest to highest that would be R, U, D, and C. To verify that both values exist, use the following steps:

5. In the Activity Hierarchy diagram, select the **GOLFER MANAGEMENT** function.

6. Select the **Expand**  icon on the Tool Bar
7. Select the elementary process **CALCULATE GOLFER HANDICAP**
8. Select **Detail** from the Menu Bar
9. Select **Expected Effects** on from the drop down menu.

You can also maintain the expected effects from here.

What are the expected effects for DELETE GOLFER? It is expected to Delete the GOLFER. To delete the GOLFER, you first must read GOLFER to verify that it exists.

To record this in the matrix, follow these steps:

1. Select the empty cell at the intersection of **DELETE GOLFER** and **GOLFER**.
2. Select the **R = Read Only** cell value, and select the **Add object** icon.
3. To Add the delete cell value to the same cell, select the **D = Delete** cell value and select the **Add object** icon.

How can this help you discover gaps in your analysis? We know that every object of interest to a business has at least two lifecycle states. A creation (create) state represents that point when an object becomes of interest to a business, and the termination (delete) state represents when the object is no longer of interest to the business. However, somewhere between these states a business typically needs to use, that is read and update, these objects. Examining the CRUD actions can help us identify missing processes.



1. Use the **Rotate** icon on the Tool Bar to rotate the matrix, and select **GOLFER** in the Entity Type axis. Your matrix should look like the following illustration:

Cell values: = Not referenced C = Create D = Delete U = Update R = Read only		Elementary Process						
Entity Type		REGISTER GOLFER	LOGIN GOLFER	ENTER SCORING RECORD	CALCULATE GOLFER HANDICAP	DELETE GOLFER		
GOLFER		C	R	R	U	D		
SCORING RECORD				C	R			

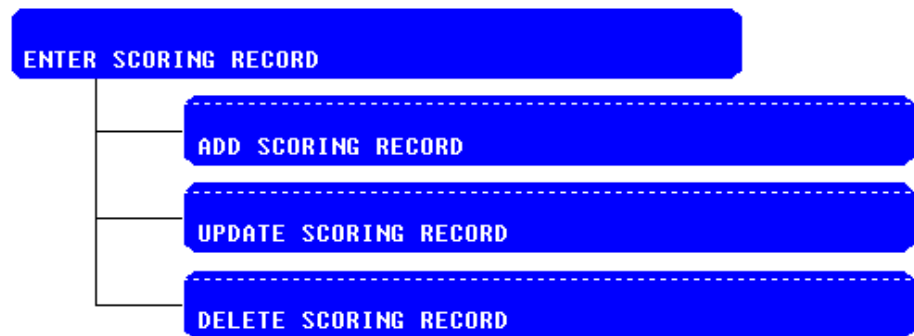
A GOLFER becomes of interest to the business when they register and they no longer are of interest when they remove themselves. Between these events, a GOLFER gets used when they login, enter their scoring records, and when we calculate their handicap index. It appears that a GOLFER receives the full treatment. However, only the golfer's handicap index is updated. What if the GOLFER changes their name or incorrectly enters their name? There is no provision to handle this change, or a change in the SCORING RECORD. Scoring records are created and read, but no provision exists for updating them if they were incorrectly entered, or for removing them.

Some development methodologies suggest that common action blocks (CAB) can be created to handle these error correction routines. Most people prefer to see them as elementary processes on the Activity Hierarchy diagram. One technique is to take the elementary process where these error correction routines are most likely needed, mark them as non-elementary, and add them as new elementary processes following the original process.

For example, REGISTER GOLFER can be marked as non-elementary, with ADD GOLFER and UPDATE GOLFER added as elementary processes underneath it.

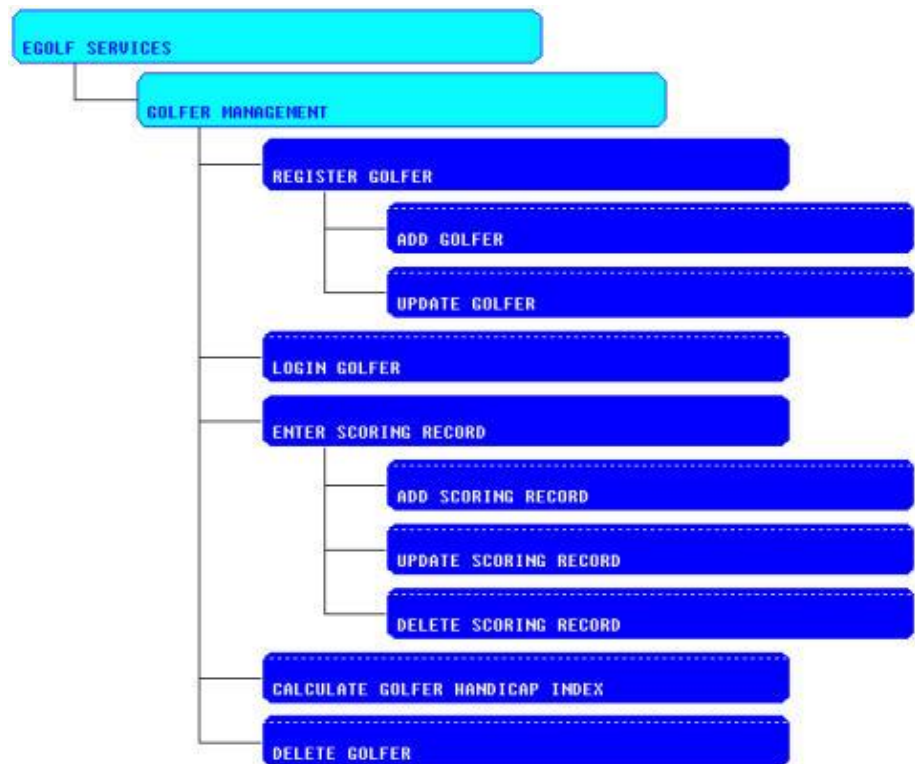


ENTER SCORING RECORD can be marked as non-elementary with ADD SCORING RECORD, UPDATE SCORING RECORD, and DELETE SCORING RECORD added following it.





2. Make these changes to the Activity Hierarchy diagram. Uncheck the **Elementary** checkbox from your processes before adding elementary processes to them.

The improved activity hierarchy should look like the following illustration:



Our first level processes under **GOLFER MANAGEMENT** still represent the lifecycle of a golfer, and we have incorporated all of the activities to fully support the object. Each activity marked as an elementary process meets the guidelines set forth earlier for an elementary process.

To contract and expand portions of the diagram or the entire diagram, select an activity that has lower-level activities and select the **Expand**  or **Contract**  icons on the Menu Bar. When an object is contracted it shows **...**. This is true for many other diagrams as well, including the Data Model diagram.

The updated Elementary Process uses Entity Type matrix that look similar to the following example. You can sort or move columns around as necessary.

Cell Values:									
= Not referenced									
C = Create									
D = Delete									
U = Update									
R = Read only									
Entity Type									
GOLFER		C	U	R	R			U	D
SCORING RECORD					C	U	D	R	

Each entity type now has a full complement of CRUD actions.

Detailing the Business Rules

The following sections deal with detailing the business rules.

Lesson Objectives and Time Allotment

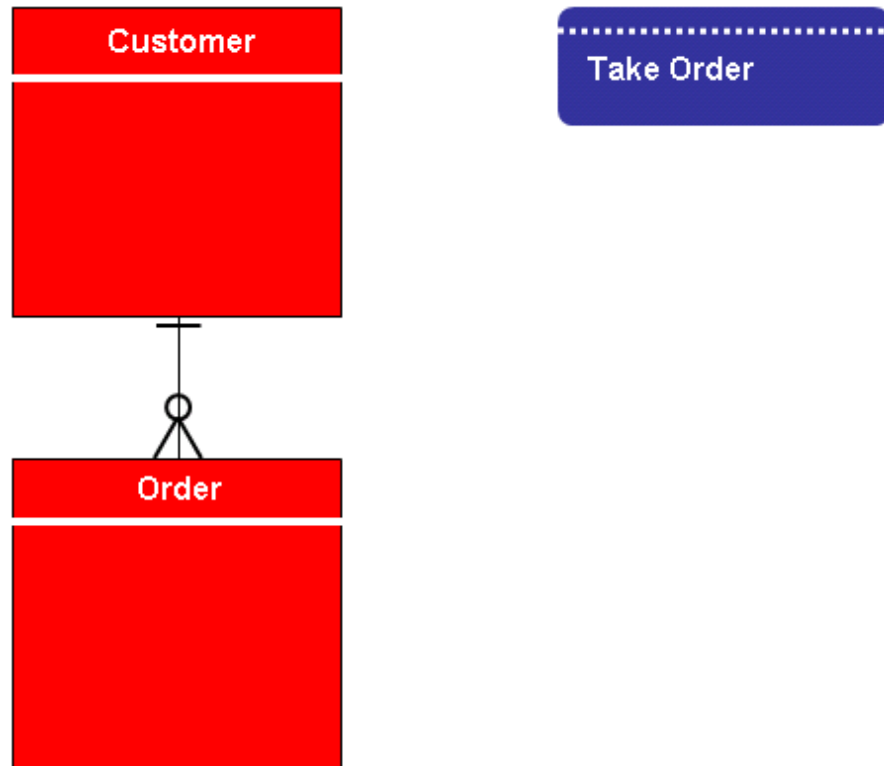
After this lesson, you will be familiar with:

- The basics of action diagramming
- The five steps of Process Logic Analysis
- Action Block Synthesis

Allow approximately 2 hours and 40 minutes to complete this lesson.

Action Diagramming Basics

As a review, in our business model, entity types represent things of interest to our business. Processes represent the business activities we perform with those things. The downstream effects are that the data model forms the basis for our database design, and the elementary processes we defined become subroutines called by batch, online, client/server, or Web Procedures, that we write in the Design phase.



To eventually turn elementary processes into subroutines, we need to continue detailing them in terms of the specific business rules associated with each one. For example, a business can have a process called Take Order. One of the business rules associated with this business can be that any Customer in good standing automatically gets a five percent discount applied to their Order. Detailing the business rules for this process involve documenting, in our action diagram notation, how we determine if a customer is in good standing and applying the five percent discount to their order.

These business rules are documented in the elementary process' Process Action Diagram (PAD). When we refer to processes, we are referring to elementary processes. Only elementary processes have Process Action Diagrams, and only elementary processes are carried forward into Design.

Other types of action diagrams are the Procedure Step Action Diagrams (PrAD) and the Common Action Diagrams (CAD). The type of action diagram, process, procedure step, or common, signifies the source of the action diagram:

- Process Action Diagrams are associated with elementary processes
- Procedure Step Action Diagrams are associated with Procedures, essentially programs, that we cover in Design
- Common Action Diagrams are not associated with processes or procedures. Common Action Diagrams represent some common code that can be called by processes, procedures, or other Common Action Diagrams. Common Action Diagrams are often known as Common Action Blocks, and sometimes as Process Action Blocks, depending on if they were discovered in the Analysis or the Design phase.

All action diagrams have some common characteristics, as you can see by examining the following example:

```
XYZ_ACTION_DIAGRAM
IMPORTS:
EXPORTS:
LOCALS:
ENTITY ACTIONS:
```

At the top of the diagram is its name, XYZ ACTION DIAGRAM. If this were a PAD, the name would be the same as the process' name. Changing the name here would change it on the Activity Hierarchy and vice versa. If this were a PrAD, it would be the same name as the Procedure Step, and if it were a CAD, it would not be the same name with anything but itself.

At the top and bottom of the action diagram name are its place holders for Information Views. There are four types of information views:

- Import
- Export
- Local
- Entity Action

The action diagram logic manipulates the information in these views.

The import views and export views are the linkage to the action diagram. Any information, arguments passed into the diagram are passed through the import views. Any information passed out of the diagram is passed through the export views. While the import and export views can be configured to be bi-directional, as the linkage is in most programming languages, that is not the normal case in CA Gen. Separating the views in such a way helps the analyst think through the action diagram logic. Import views generally contain the information the action diagram needs to begin execution, and the export views contain the information the action diagram needs to provide to other action diagrams, screens, windows, or reports.

Local views represent information that the action diagram needs to maintain temporarily; that is, for the length of its execution. After the diagram completes execution, the information in the local, import, and export views is lost. For that reason, the import, export, and local views are collectively known as transient views.

Entity Action views represent the data in the database that the action diagram manipulates. If the action diagram reads, updates, deletes, or creates an occurrence of an entity type, it requires an entity action view. Since this view is actually manipulating the data in the database, when the program completes execution, the data remains in the database, assuming it was not being deleted. The entity action views are known as persistent views.

In most programming languages, the programmer must define the arguments in terms of length and domain, a process that is error-prone. How many times have we seen data truncated as it was passed from program to program? Worse yet, how many times have we seen programs abend due to a mismatch in arguments? With CA Gen, your data model contains the majority of your data definitions. To define an argument to contain a Golfer's name, for instance, add an entity view of a Golfer containing the Name attribute. We will see an example of this shortly. By definition, this view would be the correct length and domain. If you realized you made a mistake, updating the data model updates the argument's properties in every action diagram in which it was used.

Finally, at the bottom of the views, but within the bracket that runs down the left side of the action diagram, we would see the action diagram statements required to represent the business rules.

Action diagram statements can broadly be categorized as follows:

- Entity Actions such as Create, Read, Read Each, Update, and Delete
- Relationship Actions such as Associate, Disassociate, and Transfer
- Assignment Actions such as Set, Remove, Summarize, Move, and Exit State
- Conditional Actions such as If / Else, and Case Of

- Control Actions such as Next, Escape, and Use
- Repeating Actions such as Read Each, For Each, Repeat Until, For, and While
- Miscellaneous Actions such as Note

Although this list is not meant to be all-inclusive, it represents many of the action diagram statements available.

eGolf Services Action Diagramming

As an example, let's look at the Login Golfer process. What are its inputs, outputs, and what work is it going to do?

If we had been good analysts and taken the time to enter a description for this process, it could read something like the following:

With a golfer's user ID and password, this process verifies that the golfer has registered to use our system, and that they are who they say they are by verifying that the password matches the one on file for them. If so, we welcome them back using their name.

Here is how the business rules discussed would look in our action diagram notation:

```
LOGIN GOLFER
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
  userid (mandatory)
  password (mandatory)
EXPORTS:
  Entity View export golfer (transient,export only)
  last_name
  first_name
  handicap_index
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
  userid
  password
  last_name
  first_name
  handicap_index

  READ golfer
    WHERE DESIRED golfer userid IS EQUAL TO import golfer userid
  WHEN successful
    IF golfer password IS EQUAL TO import golfer password
    MOVE golfer TO export golfer
    ELSE
    EXIT STATE IS invalid_password
  WHEN not found
  EXIT STATE IS invalid_userid
```

The information to begin executing this process, is a golfer's user ID and password. This information comes from the import view. Since this information is required for this process to begin execution, they are marked as mandatory.

Information needed because of the successful execution of this process, is the golfer's name, so that we can welcome them back by name. This information comes from the export view.

Since we do not need any temporary use of information, we do not need any local views.

We are going to access our database of golfers, to see that they exist and to check their password. The entity action view will be used for this purpose.

The action-diagramming notation appears in brackets, following the information views.

Looking at the action-diagramming notation, we see that we are going to read a golfer in our database, based on the value of import golfer user ID. There are two possible outcomes of this read, success, or there will not be a golfer in our database with a user ID matching the one given to us.

If the golfer is not found, we are going to set a special system attribute, the exit state, to a value indicating that the golfer was not found. Later in the Design phase, we will decide how we want to handle that condition. We might want to return a message, print a report, highlight some field on a screen or window, do all three, do something else entirely, or do nothing.

If the read was successful, we want to verify that the password we have on file matches the one given to us in the import view. If the passwords match, we want to move the golfer name we read from our database to the export view.

Note: There are more attributes in the entity action view than there are in the export view. On a move statement, only corresponding attributes move.

If the passwords fail to match, we set the exit state to a value indicating an invalid password. We determine the appropriate action for that condition in the Design phase.

Process Logic Analysis

A five-step technique exists, the Process Logic Analysis, to help think through the main processing logic, identifying entity, and relationship actions. The steps are:

Identify the primary entity type

Identify the primary entity type that is the focus of the process. Most processes deal with more than one entity type, but only one entity type is the true focus of that process. In the Login Golfer example, Golfer is the entity type that is the primary focus of that process.

Determine the action to take on the primary entity type

Determine the entity action (create, read, update, or delete) to be taken on the primary entity type. In the Login Golfer example, it is to read it.

"Walk the neighborhood" looking for other actions and entity types.

Walk the neighborhood means to look at all of the directly related entity types and determine if they are involved in this process. In our data model, Golfer lives in a small neighborhood. Golfer has one optional relationship to Scoring Record. Since the process, Login Golfer, is not involved with Scoring Record, there are no other actions or entity types required.

Sequence the actions.

Things should happen in a logical sequence. For instance, you generally must read something first to update or delete it. Since Login Golfer only has one action, read, it is easy to sequence them. We read the golfer first.

Determine selection criteria.

Of all of the Golfers in our database, how are we going to choose the one we want? In many cases, it will be by an attribute of the desired entity type. In other cases, it will be by the entity type's relationship to another entity type, and then by an attribute of the related entity type. In the Login Golfer example, it is by the Golfer's User ID.

Lesson Activity

In this exercise, we are going to:

- Perform Process Logic Analysis on each of the processes in our model.
- Manually add the process logic to:
 - ADD GOLFER
 - UPDATE GOLFER
 - LOGIN GOLFER
 - DELETE GOLFER
- Use Action Block Synthesis to automate adding the process logic to:
 - ADD SCORING RECORD
 - UPDATE SCORING RECORD
 - DELETE SCORING RECORD
- Use a component to CALCULATE GOLFER HANDICAP INDEX.

Add the Process Logic to ADD GOLFER

Before entering the process logic into the model, perform the Process Logic Analysis (PLA). For relatively simple action diagrams, perform PLA mentally. For complex diagrams, perform PLA by writing on printout of the relevant portion of your data model. Try performing the Process Logic Analysis for this process yourself, using the following outline before comparing your answers to the ones we suggested.

1. What is the primary entity type that is the focus of the ADD GOLFER process ?

2. What is the entity action to take on this entity type?

3. Walking the neighborhood, are there any other entity types or actions required of this process?

4. Sequence the actions.

5. Determine the selection criteria.

Now, look at our suggested answers:

1. What is the primary entity type that is the focus of the process ADD GOLFER?
GOLFER
2. What is the entity action to take on this entity type?
CREATE
3. Walking the neighborhood, are there any other entity types or actions required of this process?

GOLFER has an optional relationship to SCORING RECORD. In other words, according to the data model, golfers can exist without any scoring records. Scoring Records are important, but they are not important to this process. So there are no other actions or entity types required.
4. Sequence the actions.

Since there is only one action, sequencing it is easy. The first (and last) thing we will do is create an occurrence of GOLFER.


5. Determine the selection criteria

In this particular instance, we do not need to select a golfer; we are creating a golfer. So there are no selection criteria.

Now we can add the process logic to the ADD GOLFER process action diagram.

Open the ADD GOLFER Action Diagram

Follow these steps:

1. If it is not already open, open and expand the Activity Hierarchy diagram.
2. Select the **ADD GOLFER** process.
3. From the Tool Bar, select the **Action Diagram**  icon. An empty ADD GOLFER action diagram should open.
4. From our Process Logic Analysis, we know that this diagram has only one action: to create an occurrence of a golfer. If we are going to create, read, update, or delete entities of an entity type, we need an entity action view of that entity type. Likewise, when we create the occurrence of the golfer, we need to set each of its mandatory attributes to some value. In most cases, we cannot simply make up some value; the value must be given to us in order for us to begin. Information that is given to an action diagram in order for it to begin execution is supplied to the action diagram through its import views. Therefore, we need an import view of a golfer as well.

It is usually easier to add the views before adding the action diagram statements, but for this example, we will add them as we need them.


Begin Adding Action Diagram Statements

Follow these steps:

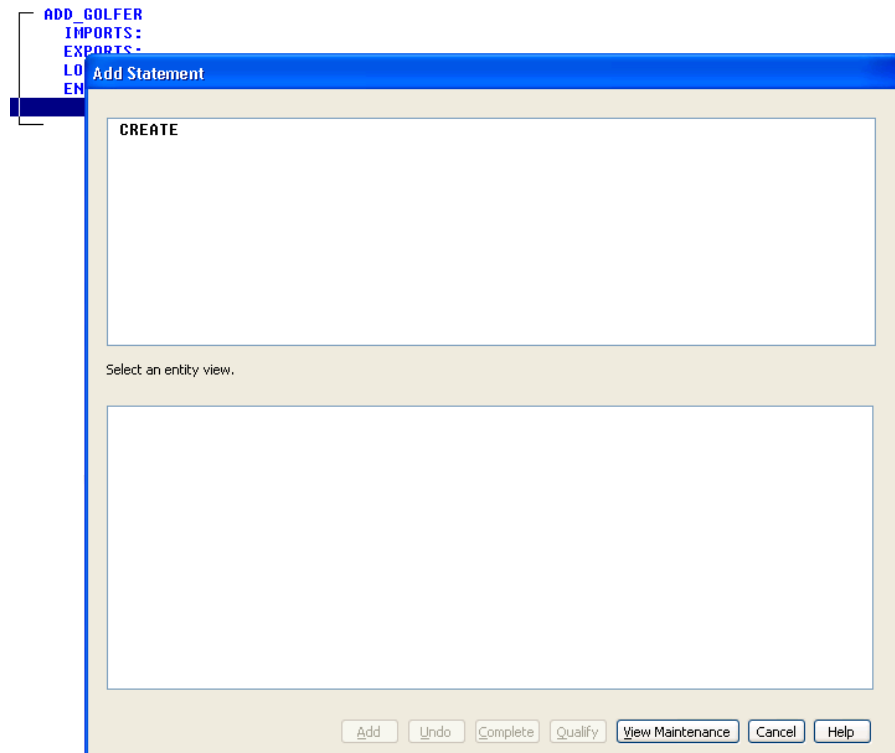
1. In the ADD GOLFER action diagram, select the blank line directly beneath ENTITY ACTIONS: as indicated in the following diagram:



Here are some tips on adding action diagram statements:

- When adding an action diagram statement, you always select the blank line or existing action diagram statement that you want to add the new statement being added. If it is directly above a statement between two existing statements, select the statement that you want to add the new statement after. The Toolset will then insert the new statement directly below of the highlighted statement and above any subsequent statements.
- The Toolset makes different options available to you depending on what you have selected. In the previous example, had we not selected the blank line and instead selected ENTITY ACTIONS: at the top of the blank line, the Toolset would not allow us to add any statements. It would, however, allow us to add an entity view to our entity action views. If we had selected the bottom of the bracket running down the far left side of the action diagram that indicates the end of the action diagram, the Toolset would also not have allowed us to add any statements. You cannot add statements outside of the action diagram.
- To add a new action diagram statement, with a few exceptions, the entire statement that you are trying to add the new statement beneath has to be highlighted. If only a portion of the statement is highlighted, the Toolset thinks you are trying to change only the highlighted portion, instead of adding a new statement altogether. The easiest way to highlight the entire statement is to select the first word in the statement.
- Finally, if you want to delete a statement, select the statement and then press the Delete key on your keyboard or select the Delete  icon on the Tool Bar. You can delete several statements at the same time. If the statements are all together, you can click-and-drag the cursor down across the first word in each statement, highlighting all of them at once, and then select the Delete key or the Delete icon. If the statements are scattered throughout the diagram, you can select the first one, and then hold down the Ctrl key while selecting the others. Once they are all highlighted, select the Delete key or the Delete icon.

2. With the blank line highlighted, select **Edit** from the Menu Bar, then **Add Statement** from the drop-down menu, and then select **Create...** from the choice of statements. This opens the Add Statement dialog as shown in the following example:




When adding statements, you build the statement in the upper panel by selecting options presented to you in the lower panel. When you are finished building the statement, you select the **Add** push button to add it to the action diagram. The **Add** push button will be disabled (as in the previous example) until you have a syntactically correct statement.

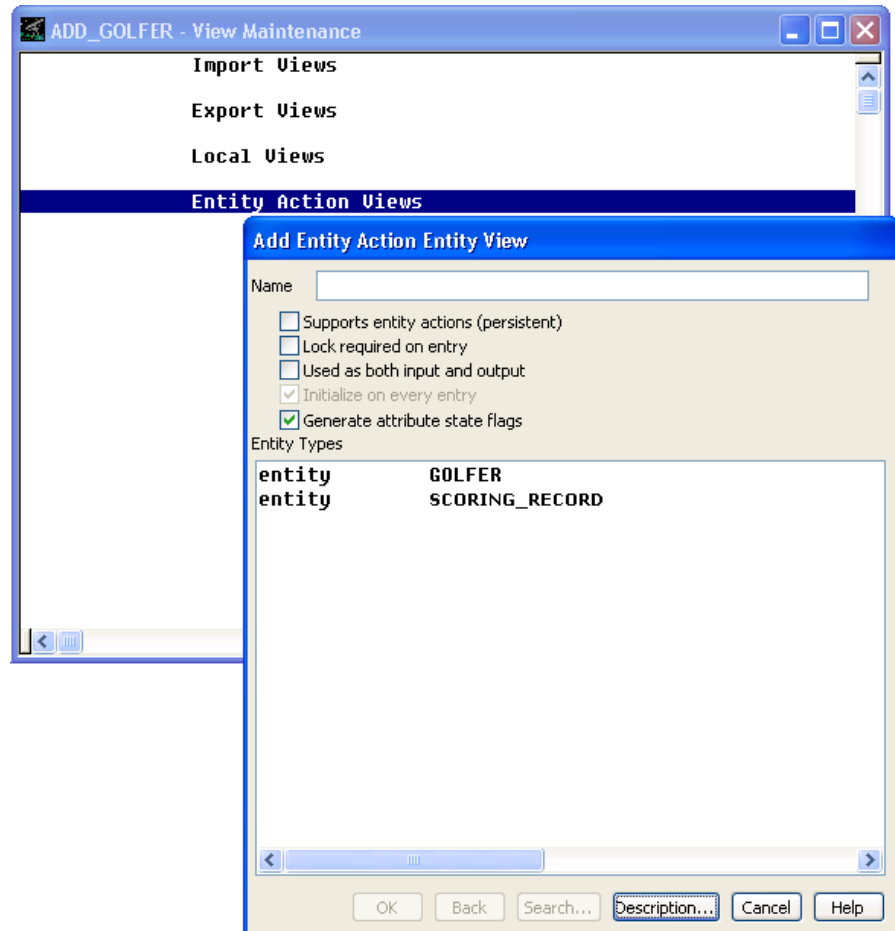
When adding Create, Read, Update, Delete, or Read Each statements, the options it would normally show you in the bottom panel are your Entity Action views. Entity Action views are used to manipulate the information in the database. Since we have not yet created any views, it is not showing us any views. However, we can perform view maintenance from here.

Perform View Maintenance

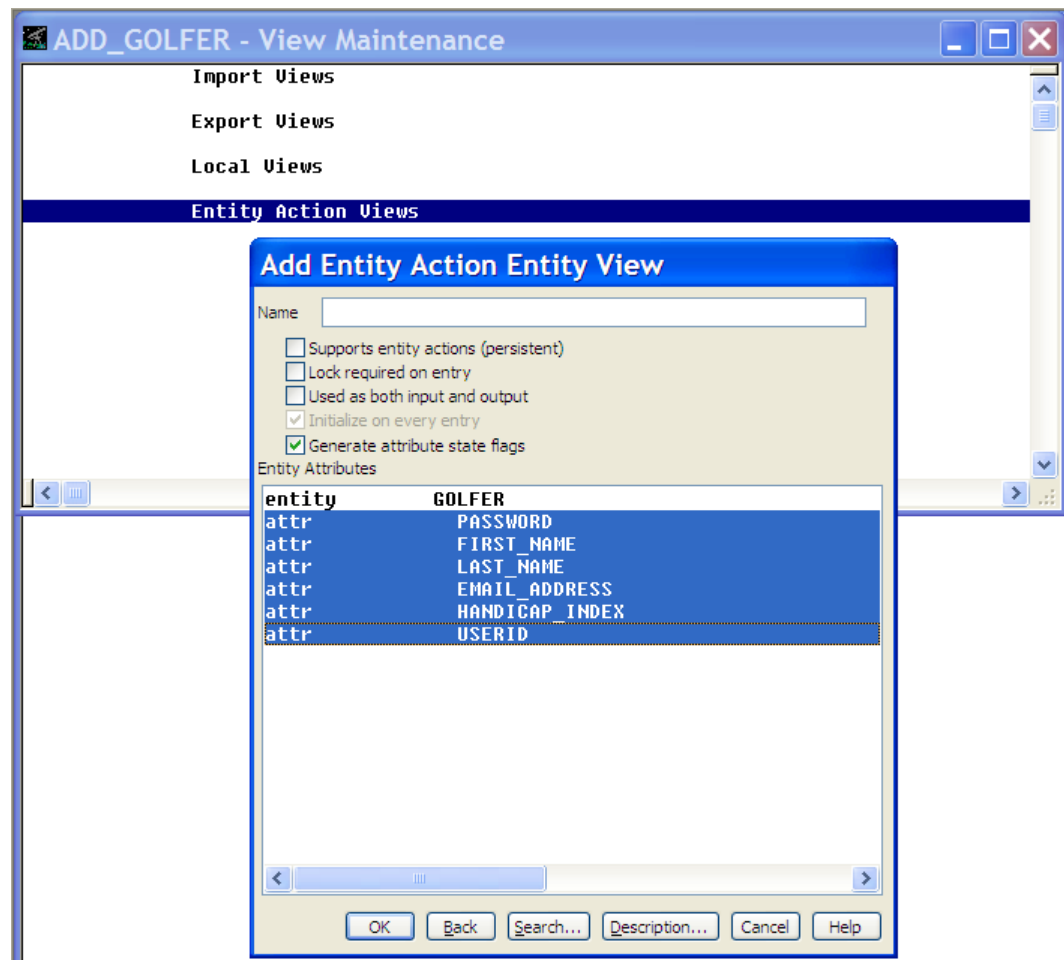
Follow these steps:

1. Select the **View Maintenance** push button at the bottom of the Add Statement dialog.
2. In the ADD GOLFER – View Maintenance diagram, select **Entity Action Views**.

3. From the Tool Bar, select the **Add Entity View...**  icon. You should receive the Add Entity Action Entity View dialog as shown in the following example:



4. In the Add Entity Action Entity View dialog, select **GOLFER** from the list of Entity Types in the model. You will be presented with a list of GOLFER attributes from which to choose. To include all the attributes for Golfer in the view, select entity **GOLFER** in the list of Entity Attributes.



5. All of the attributes should be highlighted. Select the **OK** push button to add the view to the diagrams Entity Action Views. If you have multiple adds turned on, the Add Entity Action Entity View dialog remains open, allowing you to add additional views. We do not need any additional Entity Action Views, so select the **Cancel** push button.

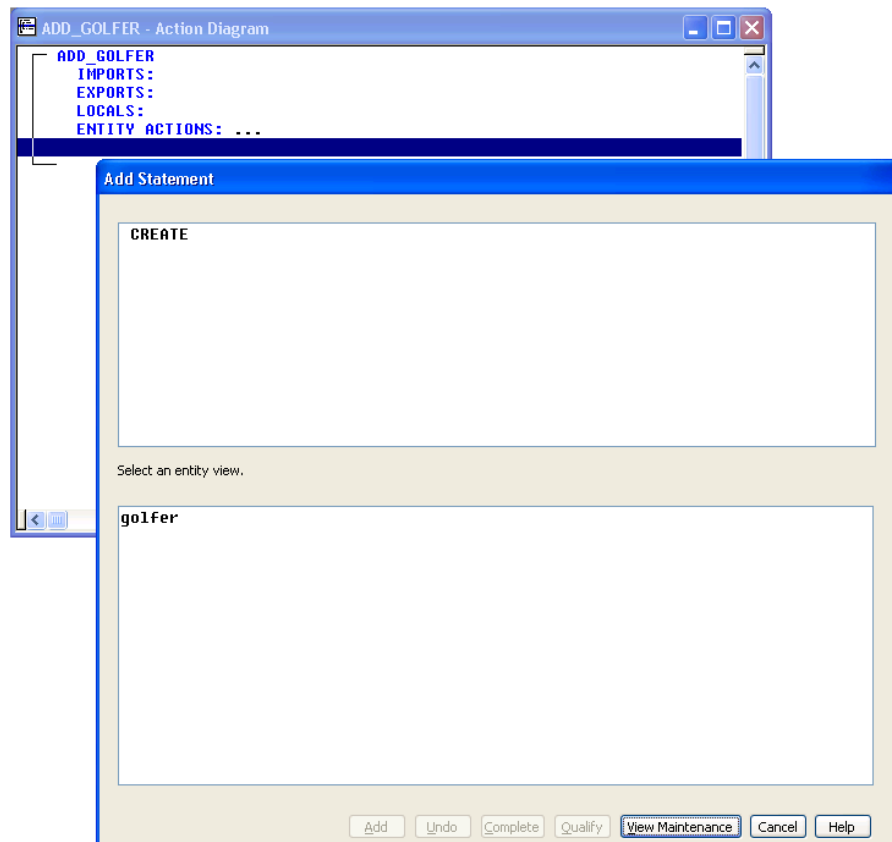
6. If your ADD GOLFER – View Maintenance diagram looks like the following example, you can close the diagram. Remember that the order of the attributes in the view is not important. If it does not look the same (except for the order of the attributes, which can be different), delete the view by selecting it and selecting the **Delete** icon, and then try adding it again. You will have to reselect the **Entity Action Views** label again.



Finish Adding Action Diagram Statements

The Add Statement dialog now gives us the choice of selecting the golfer Entity Action view that we added to the diagram.

Note: In the background the ADD GOLFER – Action Diagram has an ellipsis (...) next to ENTITY ACTIONS: indicating that there is now a contracted view (of GOLFER) there.



Follow these steps:

1. In the Add Statement dialog, select **golfer** from the bottom panel.

Note: A golfer is added to the statement we are creating in the upper panel and we are presented with new choices in the bottom panel. This will continue until we finish building the statement and then add it to the action diagram. If we make the wrong choice along the way, we can select the **Undo** push button to back up one or more selections.

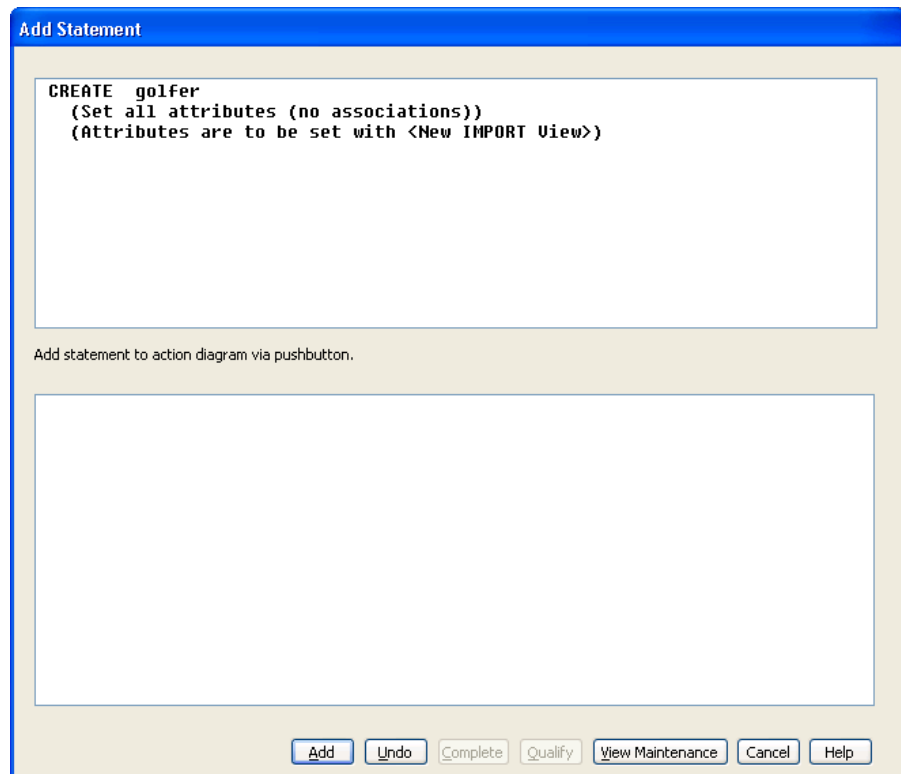
In many cases, the Toolset tries to help you out by automating some tasks and selections. For example, when we create an occurrence of golfer, we have to set its first name, last name, and so on, to a specific value. Usually these values will be passed to us through the import views. We can add the Set statements manually, much like we are adding this Create statement, or we can allow the Toolset to add them for us. Sometimes it is easier to add them manually; sometimes it is easier to let the Toolset add them. In this case, it is going to be easier to allow the Toolset to add them for us, since all of our attribute values will be set from the import view, which we have yet to define.

2. Select **Set all attributes (no associations)** from the choices in the bottom panel. If we wanted to add the attributes manually, we would have selected **Add no SET or ASSOCIATE** statements. If we only wanted mandatory attributes to be set or had relationships to other entity types to be associated, we would have selected one of the other options.

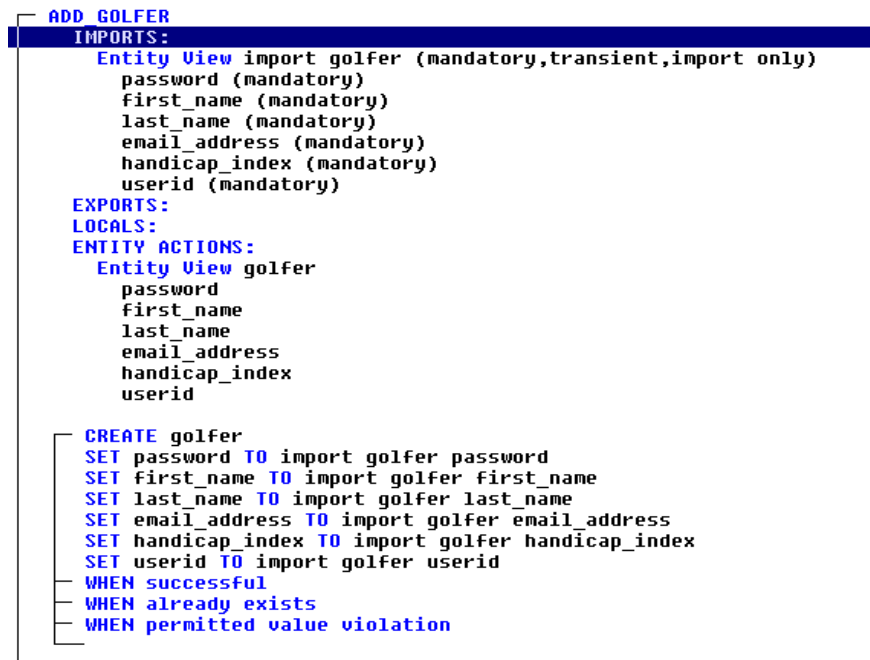
The Toolset now knows that we want it to set all of the attributes for us, but it does not know from which view to get the values to set the attributes. Normally, the bottom panel would show us a listing of the transient views in our diagram to choose from, but since we have no other views defined, there is nothing to list. However, we know that we want the attributes set from an import view, and we can let the Toolset create one for us.

Select **<New IMPORT View>** from the bottom panel.

3. If your Add Statement dialog looks like the one in the following example, select the **Add** push button to add the statement to the action diagram. If it looks different, select the **Undo** button (repeatedly if necessary) and try to correct it. If necessary, you can add the statement to the diagram, delete it, delete any views created for you, and start again.



4. Your ADD GOLFER action diagram should look like the following diagram. The order in which the attributes appear in each view is not important, as long as each view contains the attributes indicated. To expand the views as shown, from the Menu Bar select **View**, and then select **Expand All Views** from the drop-down menu. Alternatively, you can select each contracted view separately, and then select the **Expand** icon on the Tool Bar.



There are only a few things left to do to complete the ADD GOLFER action diagram.

5. According to the data model, the attributes password, first name, last name, and user ID are all mandatory, while handicap index and email address are optional. Since they are optional, they should not be marked as required (mandatory) in the import views of this process. Furthermore, the golfer will not provide us with their handicap index; we are going to calculate it for them based on the scoring records they enter. So it should not even be an import to this process, and we should not be setting it in the create statement. Therefore, we need to remove the views and the set statement associated with the handicap index from the action diagram. To delete any views, we must first remove the references to them in the action diagram statements.

Highlight the entire statement **SET handicap index TO import golfer handicap index** by selecting the word **SET** in the statement. Select the **Delete** icon from the Tool Bar and select the **Yes** push button when prompted for confirmation.

6. Select **handicap index** in both the import view and the entity action view by selecting it first in one view and then, while holding the Ctrl key down, selecting it in the other view. Your action diagram should look like the following example:

```

ADD_GOLFER
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
    password (mandatory)
    first_name (mandatory)
    last_name (mandatory)
    email_address (mandatory)
    handicap_index (mandatory)
    userid (mandatory)
EXPORTS:
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
    password
    first_name
    last_name
    email_address
    handicap_index
    userid

CREATE golfer
SET password TO import golfer password
SET first_name TO import golfer first_name
SET last_name TO import golfer last_name
SET email_address TO import golfer email_address
SET handicap_index TO import golfer handicap_index
SET userid TO import golfer userid
WHEN successful
WHEN already exists
WHEN permitted value violation
  
```

7. From the Main Menu select **Edit**, and then select **Delete View(s)...** toward the bottom of the drop-down menu. Select **Yes** when prompted for confirmation.

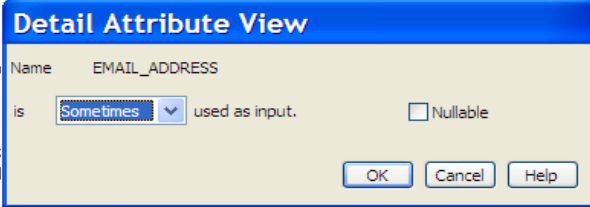
8. Double-click **mandatory** next to email address in the import view and change the properties drop-down menu to **Sometimes**. Select the **OK** push button. This will change it to optional in the action diagram.

```

ADD_GOLFER
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
    password (mandatory)
    first_name (mandatory)
    last_name (mandatory)
    email_address (mandatory)
    handicap_index (mandatory)
    userid (mandatory)
EXPORTS:
LOCALS:
ENTITY ACTIONS:
  Entity View go
    password
    first_name
    last_name
    email_address
    handicap_index
    userid

CREATE golfer
SET password TO import golfer password
SET first_name TO import golfer first_name
SET last_name TO import golfer last_name
SET email_address TO import golfer email_address
SET handicap_index TO import golfer handicap_index
SET userid TO import golfer userid
WHEN successful
WHEN already exists
WHEN permitted value violation

```




Set Exit States

There is nothing to do when this process executes successfully. We can export the golfer information, but there is really no point in that since everything we know about the golfer was passed to us in the import view in the first place. Passing it back out through the export view would be simply handing the same information back to whomever or whatever provided us with the information in the first place, and they already have it. Remember, elementary processes ultimately are simply subroutines called by another program. If we had created some new information internally, like we had randomly generated a golfer id or something, then we would have wanted to export that. So for the WHEN successful condition of the Create statement, we do not need to do anything. We do not even need to set any kind of a message or flag, since elementary processes are expected to work successfully.

However, when processes do not execute successfully, they are expected to indicate that somehow, usually by setting a special system attribute called the exit state to some value. Later on in the Design phase, we will decide how we want to handle these conditions. Therefore, we need to set the exit state in two places, once for when the golfer we tried to create already exists, and once for when we have a permitted value violation. A golfer would already exist if we already had a golfer in the database with the same value for the user ID (identifier) as the one we are trying to create. A permitted value violation would exist if we had defined specific values for an attribute (like employee gender can be either M or F) and we tried to set it to something else. We have not defined any permitted values for any attributes in our data model, but it is good practice to set an exit state for the condition anyway.

Follow these steps:

1. From the action diagram Tool Palette, select the **Exit State**  push button, and then (with the crosshair) select the **WHEN** in the statement **WHEN already exists**. An Add Statement dialog will open, and an Exit State Selection dialog will open in front of the Add Statement dialog.

Note: If the action diagramming Tool Palette is not visible on the right side of the Toolset, you can open it by selecting Options from the Main Menu and then selecting Tool Palette.

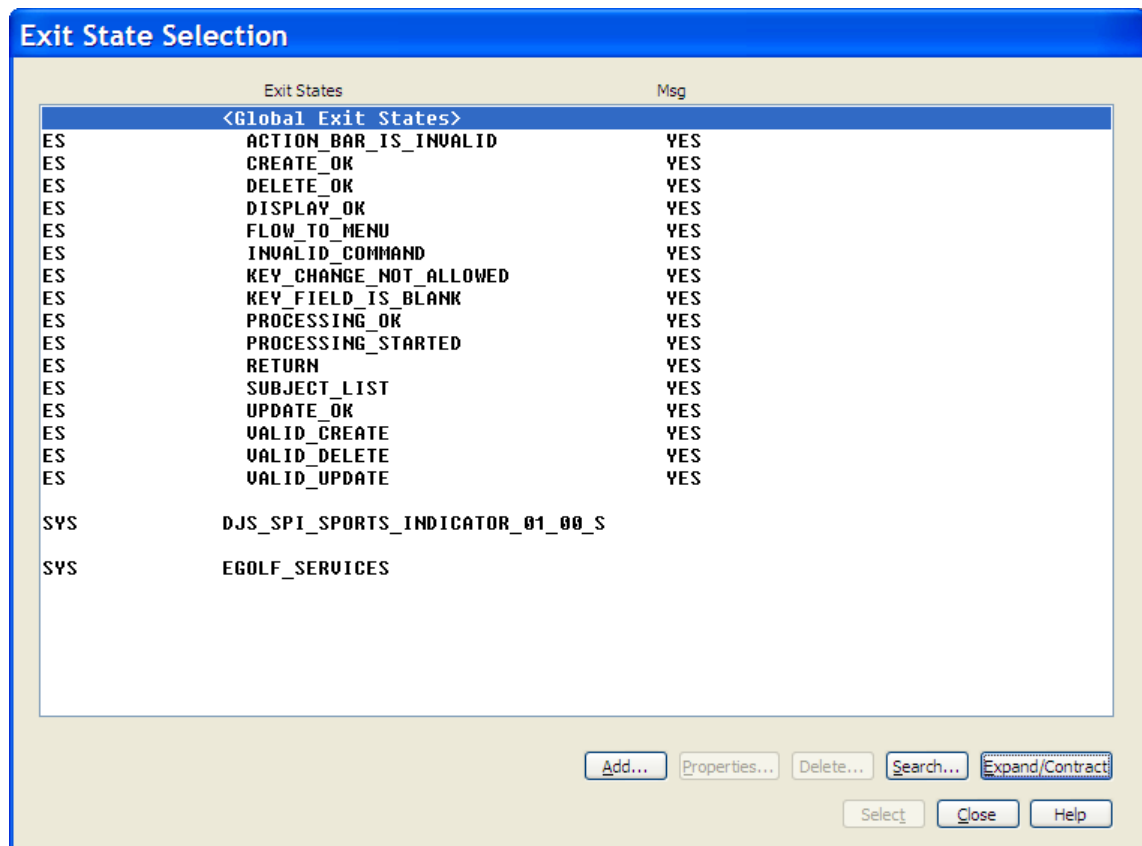
2. The special system attribute Exit State has four properties associated with it. These properties are:

- A name
- A termination action
- A message type
- A message

Rather than set each of these properties individually each time the Exit State needs to be set, pre-defined exit-state-values are created, and then used to set all four properties of the exit state at once. It becomes a little confusing to the new user, however, in that when referring to the special system attribute Exit State, and the pre-defined exit-state-values with which you can set it to, the term Exit State is used somewhat interchangeably.

We are going to pre-define four exit-state-values that will be used in almost all of our process action diagrams. Since they will be used in most of the PADs, we will add them to the list of Global Exit State values. In Design, when we are working more with PrADs, we will also be defining some exit-state-values, but they will be added to the EGOLF SERVICES business system.

In the Exit State Selection dialog, select **<Global Exit States>**, and then select the **Expand/Contract** push button at the bottom of the dialog to expand the list of global exit states.



- On the Exit State Selection dialog, select the **Add...** push button. You will receive the following (new object) Properties dialog.

The '(new object) Properties' dialog box contains the following fields and controls:

- Exit State Name:** A text input field.
- Termination Action:** A dropdown menu with 'Normal' selected.
- Message Type:** A dropdown menu with 'None' selected.
- Message Text:** A text input field.


At the bottom right, there are buttons for 'OK', 'Cancel', and 'Help'.

Referring to the following table, enter the following exit-state-values. Pay particular attention to entering the exact exit state name as shown in the table. Later on when we do action block synthesis, the tool will generate new exit-state-values if the ones that it wants do not exist. We want it to reuse these. If you have multiple adds turned on, the (new objects) Properties dialog will remain open after you select the **OK** push button. Select the **Cancel** push button after you have entered the last exit-state-value. If you do not have multiple adds turned on, you will have to select the **Add...** push button on the Exit State Selection dialog to reopen the (new object) Properties dialog each time.

Note: If you think you may have made a mistake after adding the new exit state, you can select it from the Exit State Selection list and then select the **Properties...** push button to review or change its properties.

Exit State Name	Termination Action	Message Type	Message Text
GOLFER AE	Normal	Error	User ID entered is already in use. Try another User ID.
GOLFER NF	Normal	Error	GOLFER not found.
GOLFER NU	Normal	Error	GOLFER not unique.
GOLFER PV	Normal	Error	GOLFER permitted value violation.

- After adding the last exit-state-value and canceling out of the (new object) Properties dialog, select GOLFER AE from the Exit State Selection list, select the Select push button, and then select the Add push button on the Add Statement dialog to add the statement to the action diagram.

5. From the action diagram Tool Palette, select the **Exit State**  push button, and then (with the crosshair) select the **WHEN** in the statement **WHEN permitted value violation**. An Add Statement dialog will open, and an Exit State Selection dialog will open in front of the Add Statement dialog. If necessary, expand the global exit states and select **GOLFER PV**, then select the **Select** push button followed by the **Add** push button on the Add Statement dialog.

The completed action diagram should look like the following example:

```

ADD_GOLFER
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
  password (mandatory)
  first_name (mandatory)
  last_name (mandatory)
  email_address (optional)
  userid (mandatory)
EXPORTS:
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
  password
  first_name
  last_name
  email_address
  userid

  CREATE golfer
  SET password TO import golfer password
  SET first_name TO import golfer first_name
  SET last_name TO import golfer last_name
  SET email_address TO import golfer email_address
  SET userid TO import golfer userid
  WHEN successful
  WHEN already exists
  EXIT STATE IS golfer_ae
  WHEN permitted value violation
  EXIT STATE IS golfer_pv
  
```

Perform Consistency Check

The final thing we want to do is perform a consistency check. Select the **Check** icon from the Tool Bar. You should receive the following consistency check report:

```

Consistency Check

Model: EGOLF SERVICES
Subset: (complete model)
Mar. 23, 2006 12:15

Level: BAA
Reporting: Warnings, Severe Warnings, Errors, Fatal Errors.
-----
Elementary Process Action Block ADD_GOLFER
-----

Process ADD_GOLFER
WARNING      : ICCUD01W Each non-root activity should satisfy or require at least one
dependency
WARNING      : ICCPF111W For each elementary process, expected frequency estimates should
be defined.

Number of ERRORS: 0. number of WARNINGS: 2.

- End of Report -

```

The first warning suggests that we should do dependency analysis with this process. Dependency analysis is another tool that can be used to verify the activity decomposition. In addition, it can help identify and document the sources and destinations of process information and the events that trigger the process. This tutorial will forego a discussion on dependency analysis.

The second warning indicates that we did not document the processes' expected frequencies. The expected frequencies indicate how often the process will be performed; for example, ten times a month, once a week, twice an hour, and so on. This can be useful in predicting system performance. We will forego this discussion as well.

Save your model, then move on to the next section.

Add the Process Logic to UPDATE GOLFER

Try performing the Process Logic Analysis for this process using the following outline. Then turn the page and compare your answers to the ones we have suggested.

What is the primary entity type that is the focus of the process UPDATE GOLFER?

What is the entity action to be taken on this entity type?

Walking the neighborhood, are there any other entity types or actions required of this process?

Sequence the actions.

Determine the selection criteria.

Now, look at our suggested answers:

1. What is the primary entity type that is the focus of the process UPDATE GOLFER?
GOLFER
2. What is the entity action to be taken on this entity type?
UPDATE
3. Walking the neighborhood, are there any other entity types or actions required of this process?

If we are going to update the GOLFER, the Toolset requires us to first establish currency on GOLFER. There are two ways to establish currency:

- Create an occurrence of the thing that you want to establish currency with
- Read an existing occurrence of the thing

In this case, we want to update an existing golfer, so we want to read an existing occurrence of golfer. Then, walking the neighborhood, we see that golfer has an optional relationship to SCORING RECORD. The scoring record has no bearing on this process. So there are no other actions or entity types required.

4. Sequence the actions.

We have identified two actions, an update, and a read. Prior to updating the golfer, we must first establish currency. Currency is established by reading the golfer. Therefore, the correct sequence is:

- a. Read the golfer
- b. Update the golfer

5. Determine the selection criteria.

In most cases, you will read the desired object by its identifier, or by the identifier of some related object. This usually begs the question, however, what if all we have is their name? In Design, you make provisions for looking up things by other attributes or relationships, and then once identified, you would call the update process with their unique identifier.

Now we can add the process logic to the UPDATE GOLFER process action diagram.

Open the UPDATE GOLFER Action Diagram

Follow these steps:

1. If necessary, close the **ADD GOLFER** process. Then, if it is not already open, open and expand the **Activity Hierarchy diagram**.
2. Select the **UPDATE GOLFER** process and open its action diagram. If you need assistance, see the previous section.


Perform View Maintenance

In the ADD GOLFER action diagram, we created the views as we needed them. For this diagram, we are going to create the views we need in advance.

Since we are going to both read and update a golfer, we need an entity action view. To read the correct golfer, we need to know the user ID of the golfer we plan to update. The user ID will be passed to this diagram through its import view. Likewise, after we read the golfer, we are going to update it with new values. The new values will be passed to this diagram through its import views.

Therefore, we need both an entity action and an import view of a golfer. In addition, like the ADD GOLFER action diagram, we do not need the handicap index; since that will be calculated and updated by the process CALCULATE GOLFER HANDICAP INDEX.

Follow these steps:

1. From the Tool Bar, select the **View Maintenance**  icon.
2. In the UPDATE GOLFER – View Maintenance panel, select **Entity Action Views**.

3. Select the **Add Entity View** icon from the Menu Bar.
4. In the Add Entity Action Entity View dialog, select entity **GOLFER**.
5. In the Add Entity Action Entity View dialog, select entity **GOLFER** to select all of the attributes, and then select HANDICAP INDEX to un-highlight it. Alternatively, select each attribute except **HANDICAP INDEX** individually. The dialog should look like the following example:

Add Entity Action Entity View

Name

☐ Supports entity actions (persistent)
☐ Lock required on entry
☐ Used as both input and output
☒ Initialize on every entry
☒ Generate attribute state flags

Entity Attributes

entity	GOLFER
attr	PASSWORD
attr	FIRST_NAME
attr	LAST_NAME
attr	EMAIL_ADDRESS
attr	HANDICAP INDEX
attr	USERID

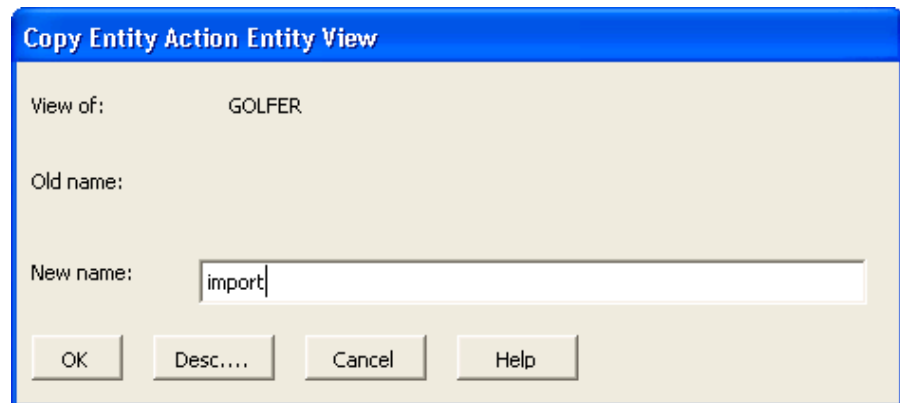
< >

OK Back Search... Description... Cancel Help

6. Select the **OK** push button to add the view to the entity action views. If you have multiple adds turned on, select the **Cancel** push button to close the Add Entity Action Entity View dialog and return to the UPDATE GOLFER – View Maintenance panel.

7. The import view is going to look exactly like the entity action view, so we can repeat the previous steps for the import view, or we can copy the entity action view to the import view. We are going to copy the entity action view to the import view.

Select **view of <unnamed>** under the Entity Action Views. From the Menu Bar, select **Edit**, then **Copy**, and with the cursor (which now looks like a hand), select **Import Views**. In the Copy Entity Action Entity View dialog, enter **import** in the New name: entry field and select the **OK** push button.

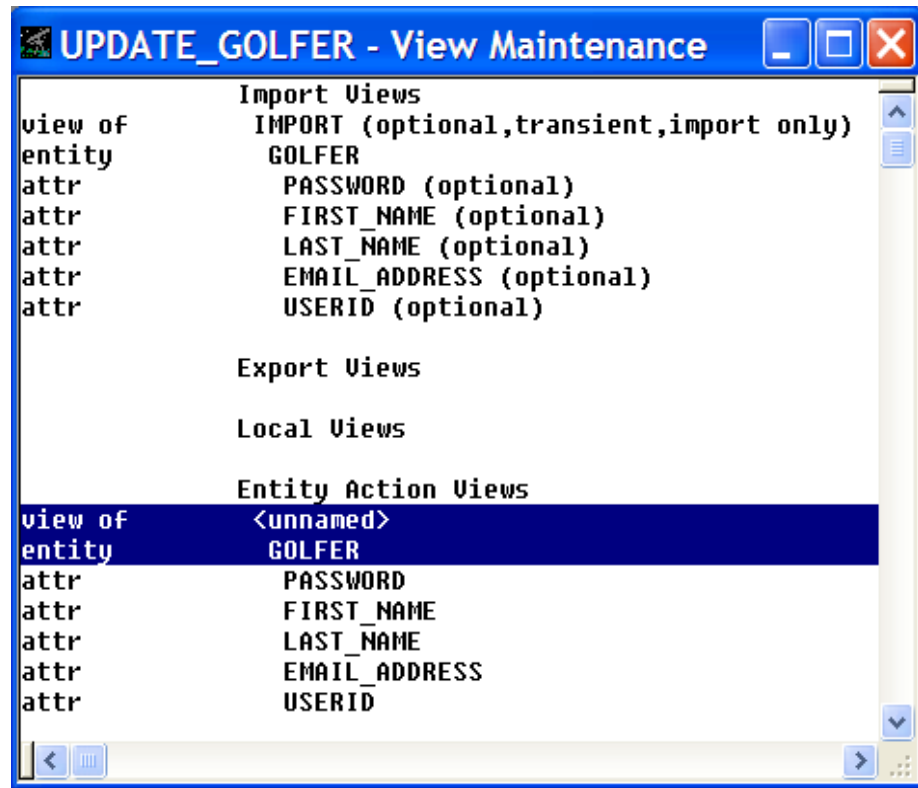


Note: If you select OK without naming the view, you will receive this error message: The name matches one already in the Import Subset.

8. Within an action diagram, you will likely have several views of the same thing, as we do of a GOLFER. The action diagram statements act on these views, and therefore need to be able to distinguish one from the other. If we said, SET golfer name TO golfer name, it is not clear which attribute value is being replaced by which attribute value. However, if we said, SET golfer name TO import golfer name, it is now clear: the value of golfer name is being replaced with the value of import golfer name. For these reasons, you must name your views.

Throughout this tutorial, import views will be named IMPORT. Export views will be named EXPORT. Local views will be named LOCAL. Entity Action views will be left unnamed. If all of the other views have names, and entity action views do not have names, we can still tell each of the views apart.

The UPDATE GOLFER – View Maintenance diagram should look as follows:



9. Close the UPDATE GOLFER – View Maintenance diagram. Expand the views in the UPDATE GOLFER – Action Diagram
10. The last thing we want to do with the views is determine which should be marked optional or mandatory. Since we are going to update the golfer by setting each of its attributes to the values supplied in the import views, we want to require values for each mandatory attribute as defined in the data model. All of the attributes were mandatory except email address, so we want to double-click the **optional** property for each view except the email address and mark it as always used as input.

The action diagram should now look like the following example:

```

UPDATE_GOLFER
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
    password (mandatory)
    first_name (mandatory)
    last_name (mandatory)
    email_address (optional)
    userid (mandatory)
EXPORTS:
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
    password
    first_name
    last_name
    email_address
    userid
  
```

Add Action Diagram Statements

Follow these steps:

1. Select the blank line in the action diagram.
2. When we performed the Process Logic Analysis, we determined that we first wanted to read the golfer, and then we would update the golfer. So, with the blank line highlighted, select **Edit** from the Menu Bar, select **Add Statement** from the drop-down menu, and then select **Read...** from the choice of statements.

There are no choices in the bottom panel of the Add Statement dialog. This is because the Toolset has already made the choice for us. Looking at the statement we are trying to build in the upper panel, we can see that it has already selected the golfer for us to read. As mentioned before, the Toolset tries to assist us by automating some tasks and selections. In this particular case, we only had one entity action view defined. Entity action views are the only views that can be read, updated, created, or deleted. Since we were trying to add a read statement, the tool knows we want to use an entity action view, and since we only had one entity action view defined, it selected it for us. If we had two or more entity action views, it would have waited for us to select one.

As such, the Add push button is enabled. If we wanted to, we can add the statement as is. And we would read some golfer, but the golfer we read would not likely be the golfer we wanted to update, unless we only had one golfer in the database. Therefore, in virtually every case, when performing a read, you will want to qualify the read, to get a particular occurrence of the thing being read.

As mentioned before, the Toolset tries to assist you in making your selections. So when qualifying the read statement, the Toolset will present you with what essentially amounts to a series of filters. There are many ways to construct a read statement; providing only an all-inclusive listing of choices would be daunting. However, if you select the wrong filter up front, you likely will not see what you are looking for, as you get further down constructing the statement. Therefore, if you have decided you have taken the wrong path, you can simply select the **Undo** push button to back up one or more choices.

In the vast majority of cases, you will qualify the thing being read either by an attribute of itself, or by its relationship to something else. In addition, if you qualify it by its relationship to something else, then you generally need to qualify the something else with an attribute of it.

For example, in this particular case, we want to read the golfer by an attribute of the golfer. We want to read the golfer whose (attribute) user ID is equal to the user ID passed to us in the import golfer user ID. Another example of an attribute qualification would be if we wanted to read all of the golfers whose (attribute) last name is equal to the last name provided to us in the import golfer last name field.

An example of a relationship qualifier would be if we wanted to read all of the scoring records for a golfer. We do not want to read the golfer, only the golfer's scoring records. So we would read each of the scoring records, where the desired scoring records (relationship) were maintained by some golfer, [and then we would use an attribute qualifier to qualify which particular golfer] and that golfer (attribute) user ID is equal to the import golfer user ID.

Therefore, the next step is to qualify the golfer being read.

3. On the Add Statement dialog, select the **Qualify** push button.
4. From the bottom panel, select the expression **attribute view**.
5. While the more common situation is to qualify something by an attribute or a relationship in which it participates, you can qualify it based upon other attributes or relationships in the model. To qualify it by an attribute of the thing you are trying to read, you would select **DESIRED persistent view**, and the Toolset will automatically fill in the thing that you are reading (for example, the thing you desire). If you wanted to read it based on some attribute of something you have already read, then selecting **CURRENT persistent view** would provide you with a list of entity action views you have already read. If you wanted to read it based on an attribute of an entity action view that you have not yet read and do not want to read, then you would select **SOME persistent view**. If none of those applies, select transient view. In this case, we want to read it based on an attribute of the thing we desire. From the bottom panel, select the occurrence phrase **DESIRED persistent view**.

6. Since in our read statement we only desired to read one thing—the golfer—the Toolset automatically selected golfer for us. However, it is possible to write a single read statement to read several things. For example, the read list in our read statement could have contained both golfer and scoring record as:

```
READ golfer
    scoring record
WHERE ...
```

If this had been the case, then after selecting DESIRED persistent view, the Toolset would have shown us both golfer and scoring record (since we desire them both), and then asked us to select the one which we want to qualify first. However, since we only had one entity action view in our read list, it selected it for us. Since (from number step four) we specified attribute (qualifier) view, it is now asking us to select the attribute of golfer that we want to use as the qualifier. Select **userid**.

7. From the bottom panel select the relational operator **IS EQUAL TO**.
8. Since golfer user ID has the domain of text, the Toolset will only allow us to compare it to other text-based (character-based) views. Since we want the golfer with a user ID equal to the import golfer user ID (a transient, text-based view), from the bottom panel we want to select the expression **character view**, and then **transient view**.
9. Since we only have one transient view—import golfer—the Toolset selected it for us. Now it is showing us a list of the character (text) attribute views in that transient view. We want the user ID, so select **userid**.
10. If the read needed further qualification, we can select **AND** or **OR**, but in our case we are finished. Select the Add push button at the bottom of the **Add Statement** dialog.
11. From the Tool Palette, select **Exit State** and then (with the crosshair) select the **WHEN not found** clause of the read statement. In the Exit State Selection dialog, expand **Global Exit States** if necessary and select **GOLFER NF**, then select the **Select** push button. In the Add Statement dialog, select the **Add** push button to add the statement.
12. From our Process Logic Analysis, the second action we wanted to do was an update. Select the **WHEN successful** clause of the read statement. From the Menu Bar, select **Edit**, **Add Statement**, and then the **Update...** statement.

13. In the Add Statement dialog, the Toolset has already selected golfer, since that was the only entity action view we had. In the bottom panel of choices, select **Set All Attributes**, and then select import golfer as the view from which to set the attributes. Select the **Add** push button to add the statement to the action diagram.
14. When the update is successful, we do not need to do anything. However, we need to set exit states for the other two conditions. From the Tool Palette, select **Exit State** and then select the **WHEN not unique** clause of the update statement. If necessary, expand the **Global Exit States** and select **GOLFER NU**, select the **Select** push button, and then select the **Add** push button to add the statement to the action diagram.
15. Do the same to add the GOLFER PV exit-state-value to the WHEN permitted value violation clause of the update statement.

The completed UPDATE GOLFER action diagram should look as follows. Remember that the order of the attribute views within the entity views is not important.




Save your model and close the UPDATE GOLFER action diagram.

Add the Process Logic to LOGIN GOLFER

We have already reviewed the process logic for Login Golfer. Given a golfer's user ID and password, we are going to read the golfer based on the user ID passed to us, and check to see if the password given was the same as the password we have stored in the database. If so, we will export the golfer's name and handicap index so that we can generate an appropriate welcome message for them. The actual message will be designed when we get into the Design stage of the development process.

Perform View Maintenance

Follow these steps:

1. Start by opening the LOGIN GOLFER action diagram. See the previous examples if you need help.
2. Select **ENTITY ACTIONS:** in the action diagram. From the Menu Bar, select **Edit**, select **Add View**, and then select **Add Entity View....**
3. In the Add Entity Action Entity View dialog, select **GOLFER**. Then, in the list of attributes, select all of the attributes except email address. Select the **OK** push button to add the view to the entity action views within the diagram. If you have multiple adds turned on, select the **Cancel** push button to close the Add Entity Action Entity View dialog. Expand the entity action views by selecting the **Expand**  icon on the Tool Bar.
4. We can copy this view to the import and export views, and then delete the unwanted attributes.

Select Entity View **golfer**. Press **F8** (copy) on your keyboard and (with the hand as the cursor) select **IMPORTS:**. Enter **import** in the New name: field on the Copy Entity Action Entity View dialog. Select the **OK** push button to complete copying the view.
5. Press **F8** again and copy it to the export views naming it **export**.
6. From the Menu Bar, select **View**, and then select **Expand All Views**.
7. All that is required in the import view of golfer is the user ID and password. Delete the other attributes by selecting them (use the Ctrl key to make multiple selections) and then from the Menu Bar, select **Edit**, and then **Delete View(s)....** Select **Yes** when asked to confirm.
8. All of the remaining import views are required, so double-click the view property **optional** in each view and change it to **always used as input**.

9. Delete the attribute views password and userid from the export view of golfer.

The action diagram for LOGIN GOLFER should now look as:

```

LOGIN GOLFER
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
    password (mandatory)
    userid (mandatory)
EXPORTS:
  Entity View export golfer (transient,export only)
    first_name
    last_name
    handicap_index
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
    password
    first_name
    last_name
    handicap_index
    userid
  
```

10. Save your model.

Add Action Diagram Statements

Follow these steps:

1. Now, we want to add the read statement. Select the blank line in the action diagram. From the Menu Bar select **Edit**, select **Add Statement**, and then select **Read...**
2. In the Add Statement dialog, the golfer was already selected for us, since that was the only entity action view we had. Select the **Qualify** push button. We want to qualify the golfer based on the attribute user ID.
3. Select **attribute view**, then **DESIRED persistent view** (the golfer gets automatically filled in), then **userid**, then **IS EQUAL TO**, then **character view**, then **transient view**, then **import golfer**, then **userid**, and finally select the **Add** push button to add the statement to the diagram.
4. For the WHEN not found condition, set the exit state to the exit-state-value **GOLFER NF**. See the previous action diagram instructions if you need help.
5. When the read of golfer is successful, we want to check that the password entered is the same as the one we have on file for them. Select the **WHEN successful** clause, from the Menu Bar select **Edit**, then **Add Statement**, then **If...**
6. From the choices provided in the bottom panel of the Add Statement dialog, select **attribute view**, then **golfer**, then **password**, then **IS EQUAL TO**, then **character view**, then **import golfer**, then **password**, and finally select the **Add** push button to add the statement to the action diagram.

7. If the two passwords are equal, then we want to move the golfer's name and handicap index that we read off the database to the export view. From the Menu Bar select **Edit**, then **Add Statement**, and then select **Move...**
8. From the choices provided in the bottom panel of the **Add Statement** dialog, select **golfer**. The Toolset fills in the rest of the statement for us, moving the golfer to the export golfer. This was a correct assumption on the Toolset's part; but if we had something else in mind, we can select the **Undo** push button and make a different selection.
9. Select the **Add** push button to add the statement to the action diagram.
10. If the passwords do not match, then we want to set an appropriate exit-state-value. After the **MOVE** statement (which should still be highlighted), we want to add an **else** statement.

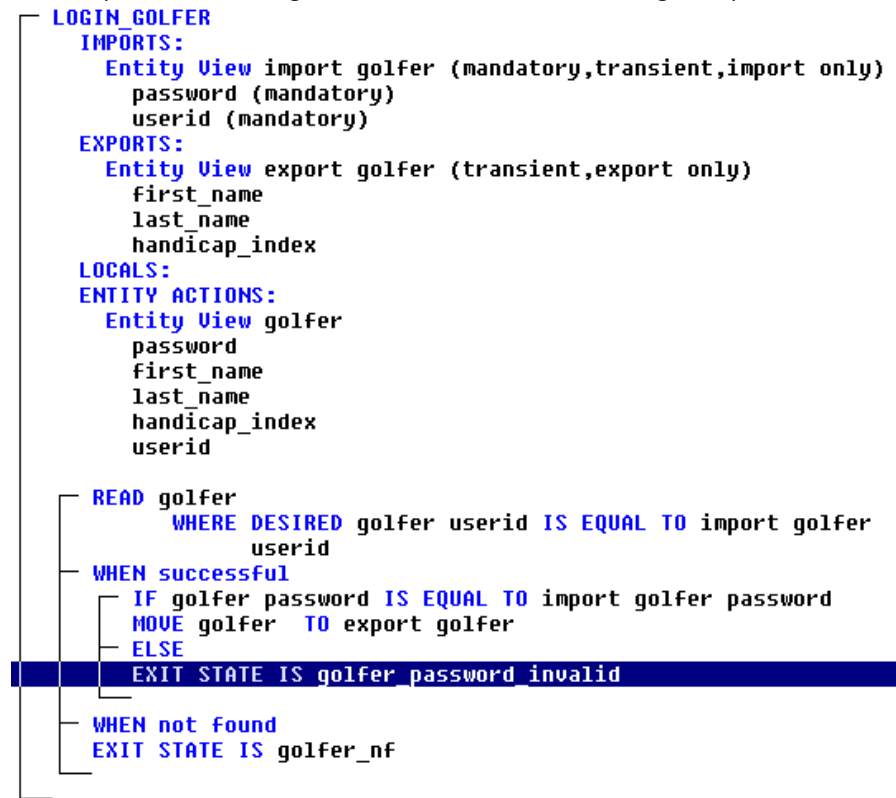
From the Menu Bar, select **Edit**, select **Add Statement**, and then select the **Else** statement.

11. To set the exit state, from the Tool Palette select **Exit State**, and then (with the crosshairs) select the **ELSE** statement. If necessary, in the Exit State Selection list, expand the **<Global Exit States>**.

There is not an appropriate exit-state-value available, so we will need to create a new one.

12. Select **<Global Exit States>**, then select the **Add...** push button. For the (new object) Properties, enter **golfer password invalid** in the Exit State Name: entry field, select **Error** for the Message Type, and enter the Message Text **The password entered is invalid. Re-enter your password**. Then select the **OK** push button to add it to the list. If multiple adds is turned on, select the **Cancel** push button to close the (new object) Properties dialog. The new exit-state-values should be highlighted. Select the **Select** push button to select it and then select the **Add** push button to add the statement to the action diagram.

The completed action diagram should look like the following example:



13. Close the diagram and save your model.

Set Exit States

Follow these steps:

1. For the **WHEN not found** condition, set the exit state to the exit-state-value **GOLFER NF**. Refer to the prior action diagram instructions if you need help.
2. When the read of golfer is successful, we want to check that the password entered is the same as the one we have on file for them. Select the **WHEN successful** clause, from the Menu Bar select **Edit**, then **Add Statement**, then **If....**
3. From the choices provided in the bottom panel of the Add Statement dialog box, select **attribute view**, then **golfer**, then **password**, then **IS EQUAL TO**, then **character view**, then **import golfer**, then **password**, and finally select the **Add** pushbutton to add the statement to the action diagram.
4. If the two passwords are equal, then we want to move the golfer's name and handicap index that we read off the database to the export view. From the **Menu Bar** select **Edit**, then **Add Statement**, and then select **Move....**
5. From the choices provided in the bottom panel of the Add Statement dialog box, select **golfer**. The Toolset fills in the rest of the statement for us, moving the golfer to the export golfer. This was a correct assumption on the Toolset's part; but if we had something else in mind, we could select the Undo pushbutton and make a different selection.

Select the **Add** pushbutton to add the statement to the action diagram.

6. If the passwords do not match, then we want to set an appropriate exit-state-value. Below the MOVE statement (which should still be highlighted), we want to add an "else" statement. From the Menu Bar, select **Edit**, select **Add Statement**, and then select the **Else** statement.
7. To set the exit state, from the Tool Palette select **Exit State Is**, and then (with the crosshairs) select the **ELSE** statement. If necessary, in the Exit State Selection list, expand the **<Global Exit States>**.
8. There isn't an appropriate exit-state-value available, so we will need to create a new one.

Select **<Global Exit States>**, then select the **Add...** pushbutton. For the (new object) Properties, enter **golfer password invalid** in the Exit State Name: entry field, select **Error** for the Message Type, and enter the Message Text **The password entered is invalid. Please re-enter your password.** Then select the **OK** pushbutton to add it to the list. If multiple adds is turned on, select the **Cancel** pushbutton to close the (new object) Properties dialog box. The new exit-state-value should be highlighted. Select the **Select** pushbutton to select it and then select the **Add** pushbutton to add the statement to the action diagram.

9. The completed action diagram should look like the example below:

```

LOGIN GOLFER
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
    password (mandatory)
    userid (mandatory)
EXPORTS:
  Entity View export golfer (transient,export only)
    first_name
    last_name
    handicap_index
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
    password
    first_name
    last_name
    handicap_index
    userid

  READ golfer
    WHERE DESIRED golfer userid IS EQUAL TO import golfer userid
  WHEN successful
    IF golfer password IS EQUAL TO import golfer password
    MOVE golfer TO export golfer
    ELSE
    EXIT STATE IS golfer_password_invalid
  WHEN not found
    EXIT STATE IS golfer_nf
  
```

10. **Close** the diagram and **save** your model.

Add the Process Logic to DELETE GOLFER

Try performing the Process Logic Analysis for this process using the following outline. Then turn the page and compare your answers to the ones we have suggested.

1. What is the primary entity type that is the focus of the process DELETE GOLFER?

2. What is the entity action to be taken on this entity type?

3. Walking the neighborhood, are there any other entity types or actions required of this process?

4. Sequence the actions.

5. Determine the selection criteria.

Now, look at our suggested answers:

1. What is the primary entity type that is the focus of the process DELETE GOLFER?

GOLFER

2. What is the entity action to be taken on this entity type?

DELETE

3. Walking the neighborhood, are there any other entity types or actions required of this process?

If we are going to delete the GOLFER, the Toolset requires us to first establish currency on GOLFER. There are two ways to establish currency:

- Create an occurrence of the thing that you want to establish currency with
- Read an existing occurrence of the thing

In this case, we want to read an existing occurrence of golfer. In walking the neighborhood, golfer has an optional relationship to SCORING RECORD, which means golfers can exist without scoring records. However, SCORING RECORD has a mandatory relationship to GOLFER. If there are scoring records attached to a golfer, and we delete the golfer, what happens to the scoring records? Since scoring records cannot exist without the golfer, deleting the golfer will automatically delete the scoring records. The Toolset provides the capability to modify these referential integrity rules if desired. There are no other actions or entity types required.

4. Sequence the actions.

We have identified two actions, a delete, and a read. Prior to deleting the golfer, we must first establish currency. Currency is established by reading the golfer. Therefore, the sequence of actions is as follows:

- a. Read the golfer.
 - b. Delete the golfer. Deleting the golfer will automatically delete any scoring records maintained by it.
5. Determine the selection criteria.

We will select the desired golfer based on its identifier.

Now we can add the process logic to the DELETE GOLFER process action diagram.

Add Action Diagram Statements

Follow these steps:

1. After opening the DELETE GOLFER action diagram, add the following views and action diagram statements to the action diagram. See the previous section if you need assistance.

Note: There are no exception conditions for a delete statement. It merely deletes whatever is current.

```

DELETE GOLFER
IMPORTS:
  Entity View import golfer (optional,transient,import only)
  userid (optional)
EXPORTS:
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
  userid

  READ golfer
    WHERE DESIRED golfer userid IS EQUAL TO import golfer
    userid
  WHEN successful
  DELETE golfer
  WHEN not found
  EXIT STATE IS golfer_nf
    
```

2. Close the DELETE GOLFER action diagram and save your model.

Add the Process Logic to ADD SCORING RECORD

Try performing the Process Logic Analysis for this process using the following outline. Then turn the page and compare your answers to the ones we have suggested.

1. What is the primary entity type that is the focus of the process ADD SCORING RECORD?

2. What is the entity action to be taken on this entity type?

3. Walking the neighborhood, are there any other entity types or actions required of this process?

4. Sequence the actions.

5. Determine the selection criteria.

Now, look at our suggested answers:

1. What is the primary entity type that is the focus of the process ADD SCORING RECORD?

SCORING RECORD

2. What is the entity action to be taken on this entity type?

CREATE

3. Walking the neighborhood, are there any other entity types or actions required of this process?

SCORING RECORD has a mandatory relationship to golfer. Therefore, they must be ASSOCIATED to a GOLFER. To associate them to a golfer, we have to establish currency on the golfer. There are two ways to establish currency:

- Create a new occurrence of golfer
- Read an existing occurrence of golfer

This process is going to read an existing occurrence. The process ADD GOLFER would create new golfers if necessary. There are no other actions or entity types required.

4. Sequence the actions.

We have identified three actions: a create, an associate, and a read. Prior to creating the scoring record, we must first establish currency. Currency is established by reading the golfer. Therefore, the sequence of actions is as follows:

- a. Read the golfer.
- b. Create the scoring record.
- c. As part of the create, associate the newly created scoring record to the current golfer.

5. Determine the selection criteria.

We will select the desired golfer based on its identifier.

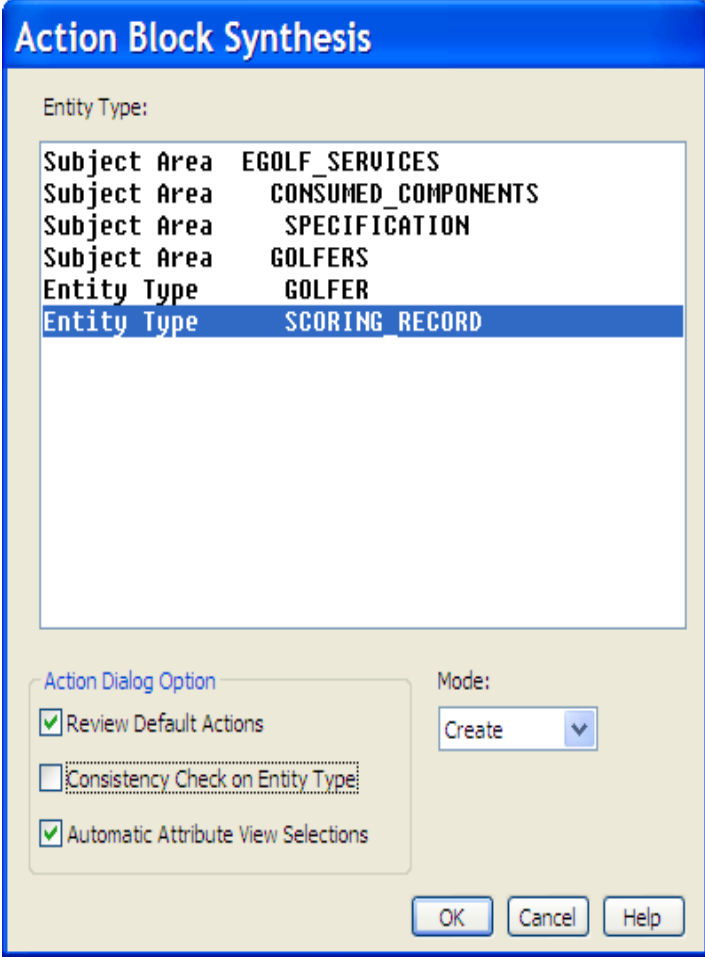
Add the Process Logic Using Action Block Synthesis

Now that we know what we want, we can add the process logic to the ADD SCORING RECORD process action diagram. However, rather than manually adding the logic as we have been doing, we will use action block synthesis to add the logic for us.

Action Block Synthesis is similar to Process Logic Analysis in that, after selecting the primary entity type and action, it walks the neighborhood for us, making assumptions about the other entity types and actions. The difference is that, once it has finished walking the neighborhood, the Toolset will automatically generate the action diagram statements based on its assumptions. Therefore, it is important to know what we want, because while action block synthesis is effective, it does not always synthesize the action diagrams exactly as we need them. Often times, the action diagram will need to be modified.

Follow these steps:

1. Select the **ADD SCORING RECORD** process and open its action diagram.
2. From the Main Menu, select **Generate**, and then **Action Block Synthesis** to open the following Action Block Synthesis dialog:



The dialog box is titled "Action Block Synthesis". It contains a list of entity types under the heading "Entity Type:". The list includes:

Subject Area	EGOLF_SERVICES
Subject Area	CONSUMED_COMPONENTS
Subject Area	SPECIFICATION
Subject Area	GOLFERS
Entity Type	GOLFER
Entity Type	SCORING_RECORD

Below the list, there are three checkboxes under the heading "Action Dialog Option":

- ☒ Review Default Actions
- ☐ Consistency Check on Entity Type
- ☒ Automatic Attribute View Selections

To the right of these checkboxes is a "Mode:" dropdown menu with "Create" selected. At the bottom right are three buttons: "OK", "Cancel", and "Help".

3. The Action Block Synthesis dialog is made up of three parts:
 - Entity Type—In this section, you choose the primary entity type that is the focus of the process
 - Mode—Select the mode, which is the action you want to take on the primary entity type.
 - Action Dialog Option—Select from several options that allow you to customize to a limited degree what is going to happen.

In this case, we know that the scoring record is the primary entity type, and that we want to create an occurrence of it. Select **SCORING_RECORD**.

4. The default action for Mode is a Create. Looking at the options, when it walks the neighborhood, it makes some assumptions regarding the relationships to other entity types. We want to Review these Default Actions and make changes if necessary. We have already done a consistency check on our entire data model and corrected the problems, so it is not necessary to do that again on individual entity types. Un-check **Consistency Check on Entity Type**. Finally, we can let it Automatically Select the Attributes to be set in the Create, or we can opt to be prompted to select them ourselves. We will go with the default, which is to let it select them for us. Select **OK**.
5. In the create of the scoring record, the Toolset recognizes that scoring record has a mandatory relationship to golfer, which means that the scoring record must be associated to a golfer. To associate the two, we must establish currency on the golfer. Currency can be established either by creating a new occurrence of golfer, or by reading an existing occurrence of golfer. Looking at the following dialog, we can see that it is assuming that we are not going to create an occurrence of golfer, but instead read an existing occurrence. The No under the Create column and a YES under the Read column indicate this. Select the **OK** push button to create the action diagram.

Create	Read	Related Entity Types
NO	YES	IS MAINTAINED BY GOLFER

Buttons: OK, Chq Create, Chq Read, Cancel, Help

Edit Action Diagram

While the synthesized action diagram is pretty close to what we wanted, we still need to make some changes to it. First, since all of the attributes are set from import views, and no new values are created from within the action diagram, it is not necessary to export anything. Therefore, we want to delete the export views. However, before we can delete any views, we must first delete the action diagram statements that reference them.

Follow these steps:

1. Under the WHEN successful for the read of golfer, delete the **MOVE golfer TO export golfer** statement.
2. Under the WHEN successful for the create of scoring record, delete the **MOVE scoring record TO export scoring record** statement. Now delete the export views.
3. Mark The attribute view note in the import view as optional.

The completed action diagram should look like the following example. Note that the new exit-state-values were created for Scoring Record.

```

ADD_SCORING_RECORD
IMPORTS:
  Entity View import scoring_record (mandatory,transient,import only)
    date (mandatory)
    time (mandatory)
    adjusted_gross_score (mandatory)
    course_rating (mandatory)
    course_slope_rating (mandatory)
    note (optional)
  Entity View import golfer (mandatory,transient,import only)
    userid (mandatory)
EXPORTS:
LOCALS:
ENTITY ACTIONS:
  Entity View scoring_record
    date
    time
    adjusted_gross_score
    course_rating
    course_slope_rating
    note
  Entity View golfer
    userid

  READ golfer
    WHERE DESIRED golfer userid IS EQUAL TO import golfer userid
  WHEN successful
    CREATE scoring_record
    ASSOCIATE WITH golfer WHICH maintains IT
    SET date TO import scoring_record date
    SET time TO import scoring_record time
    SET adjusted_gross_score TO import scoring_record adjusted_gross_score
    SET course_rating TO import scoring_record course_rating
    SET course_slope_rating TO import scoring_record course_slope_rating
    SET note TO import scoring_record note
  WHEN successful
  WHEN already exists
    EXIT STATE IS scoring_record_ae
  WHEN permitted value violation
    EXIT STATE IS scoring_record_pv
  WHEN not found
    EXIT STATE IS golfer_nf

```

Add the Process Logic to UPDATE SCORING RECORD

Try performing the Process Logic Analysis for this process using the following outline. Then turn the page and compare your answers to the ones we have suggested.

1. What is the primary entity type that is the focus of the process UPDATE SCORING RECORD?

2. What is the entity action to be taken on this entity type?

3. Walking the neighborhood, are there any other entity types or actions required of this process?

4. Sequence the actions.

5. Determine the selection criteria.

Now, look at our suggested answers:

1. What is the primary entity type that is the focus of the process UPDATE SCORING RECORD?

SCORING RECORD

2. What is the entity action to be taken on this entity type?

UPDATE

3. Walking the neighborhood, are there any other entity types or actions required of this process?

Since we are going to update a SCORING RECORD, we first need to establish currency by reading the scoring record. Part of the identifier for scoring record includes its relationship to a golfer. As such, there are at least two ways to read the scoring record:

- Read the golfer first, and then read the scoring record as it relates to the CURRENT golfer.
- Read the scoring record as it relates to SOME golfer, and then qualify that golfer without actually reading it.

Action block synthesis will choose the latter method. There are no other actions or entity types required.

4. Sequence the actions.

We have identified two actions, an update, and a read. Therefore, the sequence of actions is as follows:

- a. Perform the read of the scoring record to establish currency
- b. Perform the update

5. Determine the selection criteria.

We will select the desired scoring record based on its identifier, which includes the attributes date and time, and golfer maintains the relationship.

Add the Process Logic Using Action Block Synthesis

Now that we know what we want, we can add the process logic to the UPDATE SCORING RECORD process action diagram using action block synthesis.

Follow these steps:

1. Open the **UPDATE SCORING RECORD** process action diagram.
2. From the Main Menu, select **Generate** and then **Action Block Synthesis**.
3. In the Action Block Synthesis dialog box, select the **Entity Type SCORING RECORD**. In the Mode: drop down list, select **Update**. This time, un-check **Review Default Actions** and **Consistency Check on Entity Type**. Check **Automatic Attribute View Selections**. Select the **OK** pushbutton.

Edit the Action Diagram

Editing the synthesized action diagram includes deleting the export views. Since all of the attributes are set from import views, and no new values are created from within the action diagram, it's not necessary to export anything. But once again, in order to delete the export views, any references to them must first be removed from the action diagram.

Follow these steps:

1. In the WHEN successful condition for the UPDATE, delete the **MOVE scoring record TO export scoring record** statement, and then delete the **export views**.
2. Change import scoring record note to **optional**.
3. The completed action diagram should look as follows:

```

UPDATE_SCORING_RECORD
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
    userid (mandatory)
  Entity View import scoring_record (mandatory,transient,import only)
    time (mandatory)
    date (mandatory)
    adjusted_gross_score (mandatory)
    course_rating (mandatory)
    course_slope_rating (mandatory)
    note (optional)
EXPORTS:
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
    userid
  Entity View scoring_record
    date
    time
    adjusted_gross_score
    course_rating
    course_slope_rating
    note

  READ scoring_record
    WHERE DESIRED scoring_record time IS EQUAL TO import scoring_record time
    AND DESIRED scoring_record date IS EQUAL TO import scoring_record date
    AND DESIRED scoring_record is_maintained_by SOME golfer
    AND THAT golfer userid IS EQUAL TO import golfer userid

  WHEN successful
    UPDATE scoring_record
    SET adjusted_gross_score TO import scoring_record adjusted_gross_score
    SET course_rating TO import scoring_record course_rating
    SET course_slope_rating TO import scoring_record course_slope_rating
    SET note TO import scoring_record note
  WHEN successful
  WHEN not unique
    EXIT STATE IS scoring_record_nu
  WHEN permitted value violation
    EXIT STATE IS scoring_record_pu

  WHEN not found
    EXIT STATE IS scoring_record_nf

```


Add the Process Logic to DELETE SCORING RECORD

Try performing the Process Logic Analysis for this process using the following outline. Then turn the page and compare your answers to the ones we have suggested.

1. What is the primary entity type that is the focus of the process DELETE SCORING RECORD?

2. What is the entity action to be taken on this entity type?

3. Walking the neighborhood, are there any other entity types or actions required of this process?

4. Sequence the actions.

5. Determine the selection criteria.

Now, look at our suggested answers:

1. What is the primary entity type that is the focus of the process DELETE SCORING RECORD?
SCORING RECORD
2. What is the entity action to be taken on this entity type?
DELETE

3. Walking the neighborhood, are there any other entity types or actions required of this process?

Since we are going to delete a SCORING RECORD, we first need to establish currency by reading the scoring record. Part of the identifier for scoring record includes its relationship to a golfer. As such, there are at least two ways to read the scoring record:

- Read the golfer first, and then read the scoring record as it relates to the CURRENT golfer
- Read the scoring record as it relates to SOME golfer, and then qualify that golfer without actually reading it

Action block synthesis will choose the latter method. There are no other actions or entity types required.

4. Sequence the actions.

We have identified two actions, a delete, and a read. Therefore, the sequence of actions is as follows:

- a. Perform the read of the scoring record to establish currency
- b. Perform the delete

5. Determine the selection criteria.

We will select the desired scoring record based on its identifier, which includes the attributes date and time, and golfer maintains the relationship.

Add the Process Logic Using Action Block Synthesis

Now that we know what we want, we can add the process logic to the DELETE SCORING RECORD process action diagram using action block synthesis.

Follow these steps:

1. Open the **DELETE SCORING RECORD** process action diagram.
2. From the Main Menu, select **Generate** and then **Action Block Synthesis**.
3. In the Action Block Synthesis dialog box, select the **Entity Type SCORING RECORD**. In the Mode: drop down list, select **Delete**. Un-check **Review Default Actions** and **Consistency Check on Entity Type**. Check **Automatic Attribute View Selections**. Select the **OK** pushbutton.

Edit the Action Diagram

Editing the synthesized action diagram includes deleting the export views. But in order to delete the export views, any references to them must first be removed from the action diagram.

Follow these steps:

1. In the WHEN successful condition for the READ, delete the **MOVE scoring record TO export scoring record** statement.
2. Delete the **export views**.

The completed action diagram should look like the example below:

```

DELETE SCORING_RECORD
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
  userid (mandatory)
  Entity View import scoring_record (mandatory,transient,import only)
  time (mandatory)
  date (mandatory)
EXPORTS:
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
  userid
  Entity View scoring_record
  date
  time
  adjusted_gross_score
  course_rating
  course_slope_rating
  note

  READ scoring_record
    WHERE DESIRED scoring_record time IS EQUAL TO import scoring_record time
    AND DESIRED scoring_record date IS EQUAL TO import scoring_record date
    AND DESIRED scoring_record is maintained by SOME golfer
    AND THAT golfer userid IS EQUAL TO import golfer userid
  WHEN successful
    DELETE scoring_record
  WHEN not Found
    EXIT STATE IS scoring_record_nf

```

Add the Process Logic to CALCULATE GOLFER HANDICAP INDEX

Try performing the Process Logic Analysis for this process using the following outline. Then turn the page and compare your answers to the ones we have suggested.

1. What is the primary entity type that is the focus of the process CALCULATE GOLFER HANDICAP INDEX?

2. What is the entity action to be taken on this entity type?

3. Walking the neighborhood, are there any other entity types or actions required of this process?

4. Sequence the actions.

5. Determine the selection criteria.

Now, look at our suggested answers:

1. What is the primary entity type that is the focus of the process CALCULATE GOLFER HANDICAP INDEX?

GOLFER

2. What is the entity action to be taken on this entity type?

UPDATE

3. Walking the neighborhood, are there any other entity types or actions required of this process?

If we are going to update the golfer, we first need to establish currency on the golfer by reading the golfer to be updated. Additionally, to calculate the golfer's handicap index, we need to read each of their last 20 scoring records. The actual calculation is somewhat complicated. Fortunately, we have found a component that will perform this calculation for us, but we still need to read the scoring records and pass them to the component.

4. Sequence the actions.

We have identified three actions: an update, a read, and a read each. Therefore, the sequence of actions is as follows:

- a. Perform the read to establish currency on the golfer
- b. Read each of the golfer's last 20 scoring records
- c. Use the component and update the golfer

Unfortunately, action block synthesis will not help us out much with this.

5. Determine the selection criteria.

We will select the desired golfer based upon its identifier, and then read each of its scoring records based on their relationship to the current golfer.

Components

As mentioned previously, we were fortunate enough to have found a component that will perform the complicated handicap index calculation for us. A *component* is an independently deliverable package of software operations that can be used to build applications or larger components. The particular component we have found is a component for calculating various sports performance indices. Among its software operations is one that calculates a bowler's average and another that calculates a golfer's handicap index. Component Based Development allows you to build applications faster, cheaper, and more reliably by combining and integrating pre-built, pre-tested software components.

Components are made up of three parts:

Specification

Describes the consumer's view of what the component does.

Implementation

Describes how the component will do what it says it does.

Executables

The pieces of compiled code that will execute the component's behavior on a computer system.

A component can be delivered as either a black box component or a white box component. A *black box component* is delivered with the Specification and the Executables. A *white box component* is normally delivered with simply the Specification and the Implementation, and you generate your own Executables from the implementation. The component we have found is a black box component, consisting of only the Specification and its Executables.

Copy the Component Specification

To use a component, we have to copy its Specification, or parts of its Specification, into our model. A component Specification consists of one or more Interfaces, and each interface consists of one or more Public Operations. In CA Gen, public operations are just another form of an action diagram.

If we were going to use just one Public Operation of the component, we can just copy that one operation into our model. If we were going to use several Public Operations of the same interface, we could copy the interface into our model and that would include all of that interface's Public Operations. If we needed many operations from each of the component's interfaces, we can copy the entire specification, which would include all the interfaces and all the operations associated with each interface. In our case, we just need the one operation that will calculate the handicap index.

The version control features of an CA Gen encyclopedia can be used to copy the component's specification into your model. To simplify the Tutorial somewhat, we started with a model in which the desired Public Operation (IGLF1011_GOLFER_COMPUTEINDEX_S) has already been copied into it.

Add the Process Logic Using Action Block Synthesis

Follow these steps:

1. Open the CALCULATE GOLFER HANDICAP INDEX process action diagram.
2. We can use action block synthesis to create the basics of the action diagram. From the Menu Bar select Generate, and then select Action Block Synthesis. In the Action Block Synthesis dialog box, select Entity Type **GOLFER**, select mode **Update**, un-check all of the **Action Dialog Options**. Select the **OK** push button. In the Attribute dialog box, select **HANDICAP INDEX**, and then select the **OK** push button.

Match Views

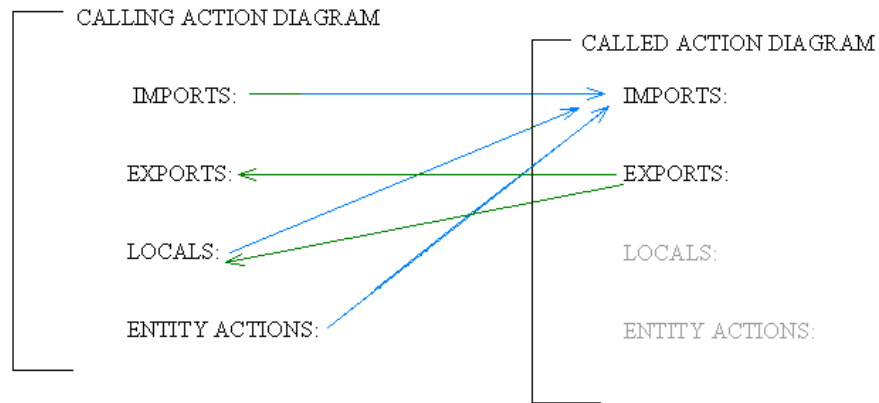
There are several changes that need to be made to the synthesized action diagram, the most important of which is reading the golfer's scoring records and passing them to the component to calculate the handicap index. To pass information from one action diagram to another, you have to match the views in one diagram to the desired views in the other diagram.

The called action diagram always receives the information being passed to it in one or more of its import views. The calling action diagram can match views in its imports, locals, or entity actions to the imports of the called action diagram.

Once called, the called action diagram does whatever it is that it is supposed to do and puts any information that it needs to pass back to the calling action diagram in its export views.

The export views of the called action diagram can then be matched to the export views of the calling action diagram or a local view of the calling action diagram.

The following picture illustrates these options. Keep in mind that these are the standard view matching rules or guidelines. There can be exceptions to these rules.



In addition, only like views can be matched. In other words, a CUSTOMER can be matched to a CUSTOMER, an ORDER to an ORDER, and a PRODUCT to a PRODUCT, but you cannot match a CUSTOMER to a PRODUCT.

In our action diagram, we want to add the call to the component operation `iglf1011_golfer_computeindex_s`. The statement you use to call another action diagram is the USE statement.

Follow these steps:

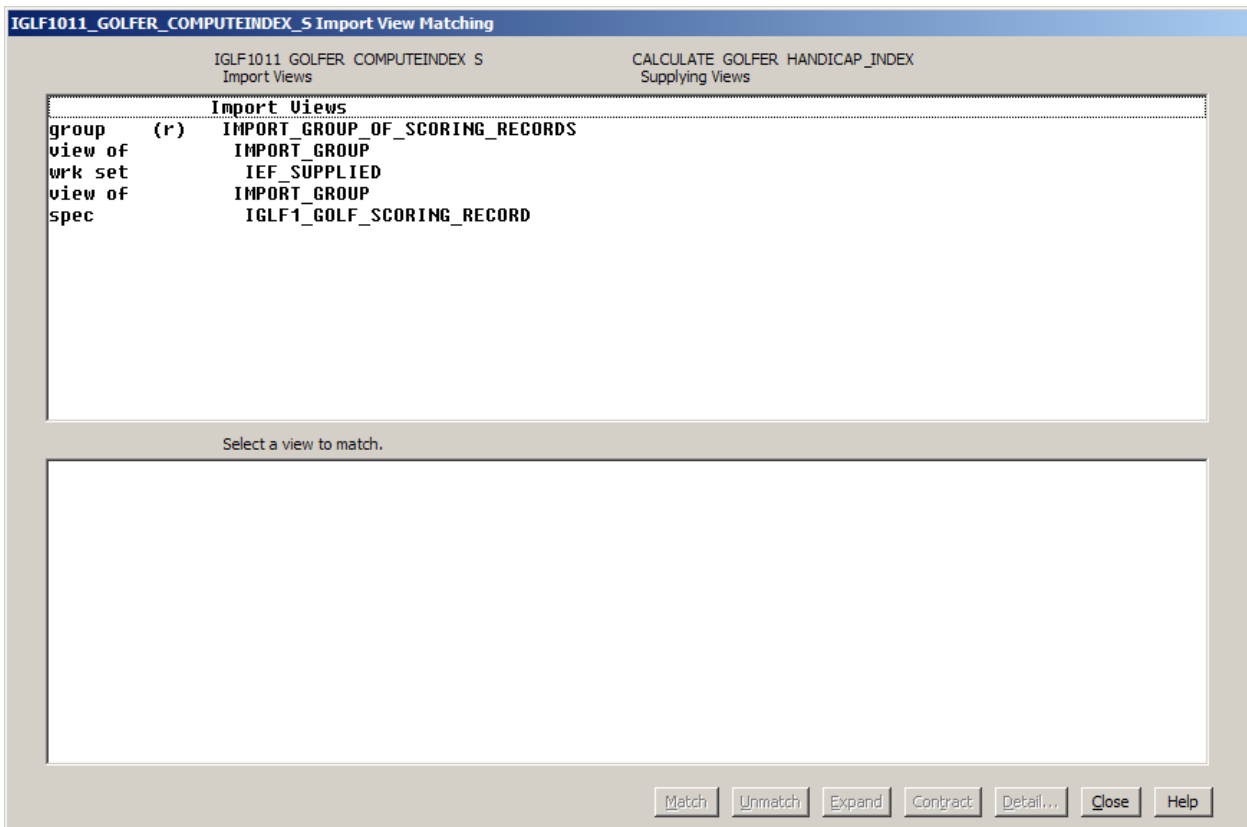
1. Select the **WHEN successful** clause of the READ golfer statement. From the Menu Bar, then select **Edit, Add Statement**, and then **Use....** In the Add Statement dialog, the Public Operation has already been selected for us. That is because it is the only action diagram available to be called. So far, the only other action diagrams in our model are the elementary process action diagrams and, by definition, an elementary process cannot call another elementary process when doing action diagramming during analysis (when doing action diagramming during design, this restriction is lifted). However, elementary processes can call common action blocks.
2. When we select the **Add** push button, the Toolset is going to take us through a dialogue in which we are prompted to match the views. It will first prompt for the import views, and then for the export views. Since only like views can be matched, we will copy the views from the import and export views of the called action diagram into our calling action diagrams local views and match them at the same time. Select the **Add** push button.

3. To the left side of the dotted line in the following picture of the Import View Matching dialog, it shows the import views of the called action diagram; in this case, the import views of the iglf1001_golfer_computeindex_s Public Operation. To the right side of the dotted line is the view from the calling action diagram that has been matched to the import view shown on the left side. Presently there are none matched. As you select an import view of the called action block on the left side to match to some view from the calling action block, you will be presented with a list of possible supplying views from the calling action block to choose from in the bottom panel.

This is the general outline for performing this process:

- a. Choose one import view from the called action block shown to the left side of the top panel.
- b. Choose the appropriate view from the calling action block that you want to match to the selected import view from the choices given in the bottom panel.
- c. Select the Match push button to match them together.

When matched, you would see the two views side by side in the top panel. You would then repeat this process for all of the required import views in the called action block. Once you have completed matching all of the import views that are necessary, you would select the **Close** push button and then go through the same process for the export views of the called action diagram.



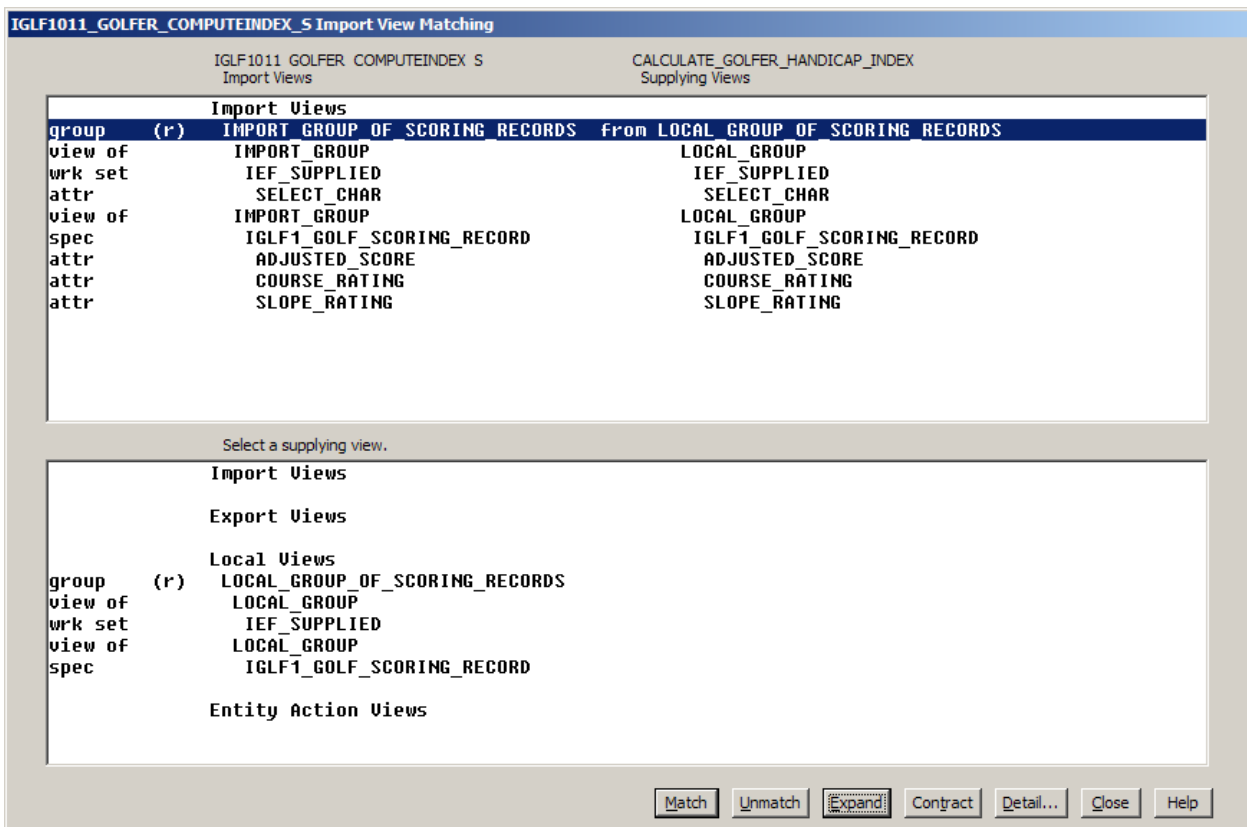
4. In the top panel, select the import view group (r)
IMPORT_GROUP_OF_SCORING_RECORDS.

Note: In the bottom panel we are shown all of the views in the calling action diagram that can be matched to the selected import view of the called action diagram.

Right now, there are none. All we see are the placeholders for the Import Views, Export Views, Local Views, and Entity Action Views. Therefore, we want to create appropriate views and match them to this import view.

By selecting the **Match** push button without selecting anything in the lower panel, the Toolset knows that we want to copy the selected view into the calling action diagram and match the two views together. Select the **Match** push button. In the View Match and Copy dialog, it assumes we want to copy this view into the import view. Most of the time that is a good assumption, but in this case we actually want to copy the view into the Local views. Select the **Local** radio button and then select the **Copy** push button.

5. When copying views, the Toolset shows you the name of the view that you are copying and asks you for the name you want to use for the new view. In the Copy Local Group View dialog, enter **local group of scoring records** and select the **OK** push button. In the Copy Import Work View dialogs, enter **local group** twice (once for each of two dialogs) and select the **OK** push button.



6. Now we can see that the import group view in `iglf1001_golfer_computeindex_s` has been copied into and matched to a local view in `calculate_golfer_handicap_index`. To see the previously shown expanded views, select **IMPORT_GROUP_OF_SCORING_RECORDS** and then select the **Expand** push button. We have no other import views that need to be matched, so we can select the **Close** push button.
7. In the Export View Matching dialog, we have two exports of the called action block that need to be matched. Select the first view of **EXPORT** in the top panel. Then, without selecting anything in the bottom panel, select the **Match** push button. This indicates to the Toolset that we want to copy this view into the calling action diagram and match the two together. It assumes that we want to copy the view into the calling action diagram's export view. Most of the time that can be correct but in this particular case we want to copy it into a local view. Select the **Local** radio button and then select the **Copy** push button. When prompted for a view name, enter **local** and then select the **OK** push button. Do the exact same thing for the **EXPORT ERROR HANDLING** view. Expand the two views and review them if desired, then select the **Close** push button.
8. Save your model. At this point, the action diagram should look like the following example:

```

CALCULATE_GOLFER_HANDICAP_INDEX
IMPORTS: ...
EXPORTS: ...
LOCALS:
  Spec View local error_handling
    severity_code
    rollback_indicator
    origin_servid
    return_code
    reason_code
  Spec View local iglf1_golf_scoring_record
    handicap_index
  Group View local_group_of_scoring_records (20,explicit)
  Work View local_group_ief_supplied
    select_char
  Spec View local_group_iglf1_golf_scoring_record
    adjusted_score
    course_rating
    slope_rating
ENTITY ACTIONS: ...

  READ golfer
    WHERE DESIRED golfer userid IS EQUAL TO import golfer userid
  WHEN successful
    USE iglf1011_golfer_computeindex_s
    WHICH IMPORTS: Group View local_group_of_scoring_records TO Group View import_group_of_scoring_records
    WHICH EXPORTS: Spec View local_iglf1_golf_scoring_record FROM Spec View export_iglf1_golf_scoring_record
    Spec View local_error_handling FROM Spec View export_error_handling
  UPDATE golfer
    SET handicap_index TO import golfer handicap_index
  WHEN successful
    MOVE golfer TO export golfer
  WHEN not unique
    EXIT STATE IS golfer_nu
  WHEN permitted value violation
    EXIT STATE IS golfer_pv
  WHEN not found
    EXIT STATE IS golfer_nf

```

In our action diagram, we can see the new USE statement that invokes the `iglf1011_golfer_computeindex_s` public operation. We can also see that the public operation imports from our local group of scoring records and exports to our local views of `iglf1011_golf_scoring_record` and error handling, all of which were copied from the public operation's import and export views.

Edit Action Diagram

We still need to read the golfer's scoring records and move them to the local group view. However, prior to doing that we can clean up the existing diagram a little bit.

Follow these steps:

1. In the update of the golfer, the handicap index is being set to the import golfer handicap index. We want to set it from the local `iglf1_golf_scoring_record` handicap index, which was returned from the component. Double-click **import golfer handicap index** in the SET statement. In the Change dialog, select **numeric view**, select **local iglf1 golf scoring record**, and then select the **Change** push button. Note that since the handicap index was the only numeric attribute in that view, the Toolset selected it for us automatically.
2. Now we can delete handicap index from the import view, since it is no longer being used. In addition, while we are at it, the value of user ID in the export view will be the same as the value in the import view, so it is not necessary to export it again. Delete **userid** from the export view.
3. Now we can read each of the golfer's last 20 scoring records and populate the local group view with them. Since we need more than one occurrence of a scoring record, we use a group view. A group view is used to establish an array. You add the group view, set its occurrence properties, and then add any other views required to the group view. All of the views within the group view will occur however many times specified in the occurrence property.

Double-click **Group View** in the Locals: view to open the group view's properties dialog (the Detail Local Group View dialog).

From here, you can change the name of the group view.

Note: This group view occurs One or More times. Virtually all group views do. However, you can have a group view that occurs only once, which is the default when adding a new group view. If they occur one or more times, then you can specify how many more times. Since we need the last 20 scoring records, we have made this group view at most 20.

Group views can be explicitly or implicitly indexed. If they are explicitly indexed, then you need to maintain a subscript to the appropriate position in the array. The first position in the array is at location 1. If a group view is implicitly indexed, then CA Gen will control the positioning in the array starting from position 1. After you have done something with the attribute values in the first position, CA Gen will automatically increment the subscript to 2, and so on.

4. Change the type of indexing used for this group view to **Implicit**. Select the **OK** push button.
5. To read each of the scoring records, we need an entity action view of scoring record. Select **ENTITY ACTIONS:**. Then, from the Menu Bar, select **Edit**, then select **Add View**, and then select **Add Entity View**. In the Add Entity Action Entity View dialog, select **entity SCORING RECORD**, select all of the attributes except NOTE, and then select the **OK** push button.
6. Now we can write the Read Each statement. Select the **WHEN successful** condition of the read of golfer. Then, from the Menu Bar, select **Edit**, select **Add Statement**, and then select **Read Each...**

Note: This time the Toolset does not automatically fill in the entity view for us. That is because we have two entity action views that can be read.

7. Select scoring record. Select the **Qualify** push button. Select **Targeting group view**. Targeting statements are used to initialize the internal subscript of implicitly indexed group views. We want the most recent 20 scores, so select **Sorted Descending**, select **date**, select **Sorted Descending** again, select **time**, select **Where expression**, select **relationship view**, select **DESIRED persistent view**, select **is maintained by CURRENT golfer**, and then select the **Add** push button.
8. With the READ EACH statement still highlighted, from the Menu Bar select **Edit**, select **Add Statement**, select **Set...**, select **attribute view**, select **local group iglf1 golf scoring record**, select **adjusted score**, select **NOT ROUNDED**, select **TO expression**, select **numeric view**, select **scoring record**, select **adjusted gross score**, and then select the **Add** push button.

9. With the new SET statement still highlighted, from the Menu Bar select **Edit**, select **Add Statement**, select **Set...**, select **attribute view**, select **local group iglf1 golf scoring record**, select **course rating**, select **NOT ROUNDED**, select **TO expression**, select **numeric view**, select **scoring record**, select **course rating**, and then select the **Add** push button.
10. With the new SET statement still highlighted, from the Menu Bar select **Edit**, select **Add Statement**, select **Set...**, select **attribute view**, select **local group iglf1 golf scoring record**, select **slope rating**, select **NOT ROUNDED**, select **TO expression**, select **numeric view**, select **scoring record**, select **course slope rating**, and then select the **Add** push button.
11. Save the model.

The completed action diagram should look as follows:

```

CALCULATE_GOLFER_HANDICAP_INDEX
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
  userid (mandatory)
EXPORTS:
  Entity View export golfer (transient,export only)
  handicap_index
LOCALS:
  Spec View local error_handling
    severity_code
    rollback_indicator
    origin_servid
    return_code
    reason_code
  Spec View local iglf1_golf_scoring_record
    handicap_index
  Group View local_group_of_scoring_records (20,implicit)
  Work View local_group_ief_supplied
    select_char
  Spec View local_group_iglf1_golf_scoring_record
    adjusted_score
    course_rating
    slope_rating
ENTITY ACTIONS:
  Entity View scoring_record
    date
    time
    adjusted_gross_score
    course_rating
    course_slope_rating
  Entity View golfer
    handicap_index
    userid

  READ golfer
    WHERE DESIRED golfer userid IS EQUAL TO import golfer userid
  WHEN successful
    READ EACH scoring_record
      TARGETING local_group_of_scoring_records FROM THE BEGINNING UNTIL FULL
      SORTED BY DESCENDING scoring_record date
      SORTED BY DESCENDING scoring_record time
      WHERE DESIRED scoring_record is maintained_by CURRENT golfer
    SET local_group_iglf1_golf_scoring_record
      adjusted_score TO scoring_record
      adjusted_gross_score
    SET local_group_iglf1_golf_scoring_record course_rating
      TO scoring_record course_rating
    SET local_group_iglf1_golf_scoring_record slope_rating
      TO scoring_record course_slope_rating

  USE iglf1011_golfer_computeindex_s
    WHICH IMPORTS: Group View local_group_of_scoring_records TO Group View import_group_of_scoring_records
    WHICH EXPORTS: Spec View local_iglf1_golf_scoring_record FROM Spec View export_iglf1_golf_scoring_record
    Spec View local_error_handling FROM Spec View export_error_handling

  UPDATE golfer
    SET handicap_index TO local_iglf1_golf_scoring_record handicap_index
  WHEN successful
  MOVE golfer TO export golfer
  WHEN not unique
  EXIT STATE IS golfer_nu
  WHEN permitted value violation
  EXIT STATE IS golfer_pv
  WHEN not found
  EXIT STATE IS golfer_nf

```

12. Close all of the diagrams but not the Toolset.

Defining Business Systems

The following sections describe defining business systems.

Lesson Objectives and Time Allotment

After this lesson, you will be familiar with the steps necessary to define Business Systems.

Allow approximately 3 minutes to complete this lesson.

Business System Definition

One of the final steps in Analysis is to take the results of Analysis and divide it up into smaller and more manageable pieces that will be carried forward into Design. These smaller and more manageable pieces are called Business Systems. A *business system* is a set of Procedures (programs) that are primarily responsible for implementing one or more of the Processes identified and documented in Analysis. Each business system can have its own set of standards for things like fonts, colors, and error messages seen on screens or windows.

We are going to group all of our elementary processes into simply one business system.

Lesson Activity

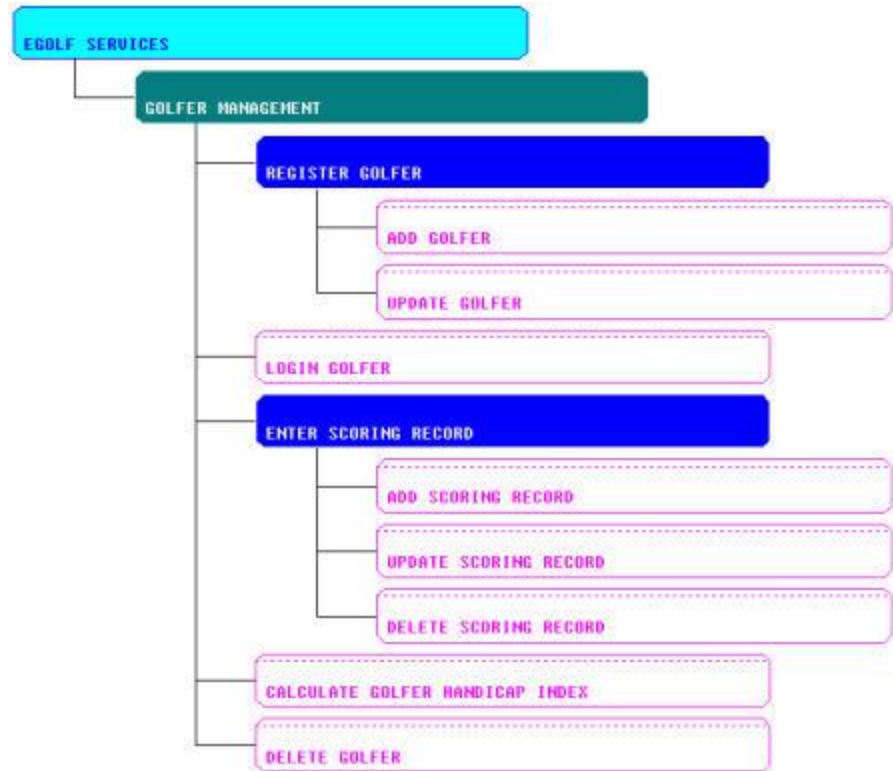
In this exercise, we are going to define a Business System.

Define the eGolf Services Business System

Follow these steps:

1. From the Tree Control, under Planning & Analysis, double-click **Business System Definition**. The business system definition diagram (which looks like an Activity Hierarchy Diagram) opens for the default business system, EGOLF SERVICES. Every model has a default business system with the same name as the model name.
2. Expand all of the processes under **GOLFER MANAGEMENT**.
3. Select the function **GOLFER MANAGEMENT**.

4. From the Tool Bar, select the **Include**  icon. The diagram should look as follows, indicating that all of the elementary processes were included into the eGolf Services business system:



5. Save the model.
6. Close the diagram

Chapter 4: Design

This section contains the following topics:

[Objectives and Time Allotment](#) (see page 149)

[Purpose of Design](#) (see page 149)

[Application Types](#) (see page 150)

[Web Client/Server Applications](#) (see page 153)

[Designing Server Procedures](#) (see page 154)

[Designing Client Procedures](#) (see page 178)

Objectives and Time Allotment

After this module, you will be familiar with:

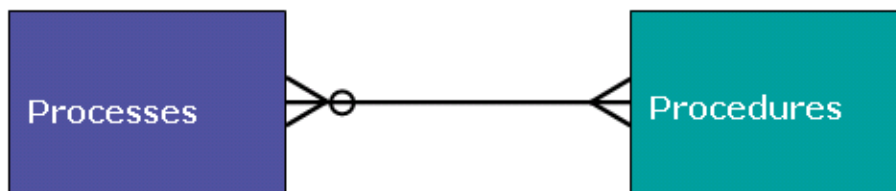
- The Purpose of Design
- The types of applications you can build with CA Gen
- Client/Server Web applications
- Development Wizards and Templates

Allow yourself approximately 8 hours and 35 minutes to complete this module.

If you would like to reference a CA Gen model reflecting the completed objectives of this chapter, you can find the model with the CA Gen installation in the \Samples\Models\Tutorial Models\Completed Models directory corresponding with this chapter.

Purpose of Design

The purpose of Design is to implement the elementary processes (for example, the Requirements) that we documented in Analysis effectively and efficiently.



In Analysis, we dealt primarily with processes. In Design, we deal primarily with Procedures. As processes have Process Action Diagrams (PADs), procedures have Procedure Step Action Diagrams (PrADs). The term procedure is synonymous with the term program. As you can have batch or client/server programs, you can have batch or client/server procedures, and these procedures will eventually be generated into executable programs.

Each elementary process will be implemented (used) in one or more procedures. Each procedure can implement zero, one or more processes. For example, the single procedure Maintain Patient can implement the three elementary processes Add Patient, Change Patient Information, and Remove Patient. In addition, we can have procedures that do not implement any processes. For example, we can have a procedure that just lists patients by their last name so that we can look up their patient identifier. Typically, we would not have an elementary process that merely produces a list of patients, since that process would not do work as we defined it in Analysis. However, all of the processes included in a Business System would eventually be implemented in one or more procedures that will be added to that business system.

There are many factors involved in how you determine which processes are implemented in which procedures. Some of these factors include:

- Performance considerations
- Corporate design standards
- User characteristics (for example, frequent versus infrequent users or high user turnover)
- Complexity and type of procedures themselves

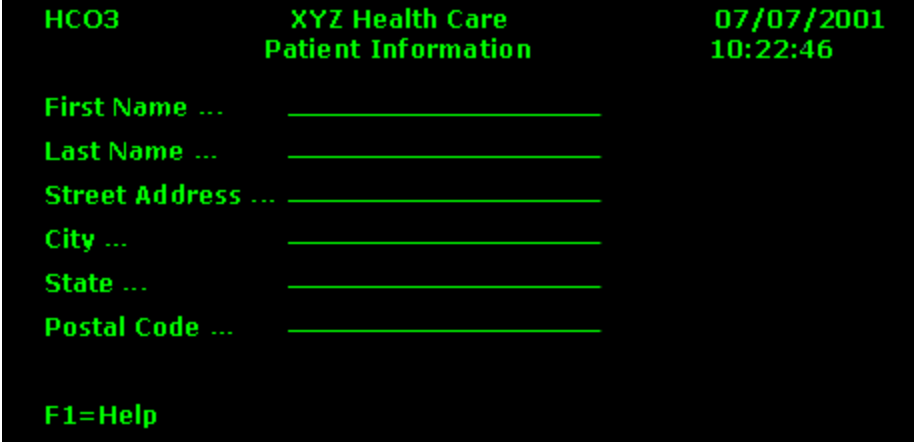
Application Types

CA Gen supports six different types of applications listed earlier, plus several more variations on these types. These application types are explained in detail in the following sections.

- Block Mode
- Batch
- Window
- Client/Server
- Proxies
- Web

Block Mode

One type of application CA Gen allows you to build is called a Block Mode application.



The screenshot displays a green-on-black terminal-style interface. At the top, the text 'HCO3' is on the left, 'XYZ Health Care' is in the center, and '07/07/2001' is on the right. Below 'XYZ Health Care' is the title 'Patient Information'. Underneath the title, there are six input fields, each with a label followed by '...' and a horizontal line for text entry. The labels are 'First Name', 'Last Name', 'Street Address', 'City', 'State', and 'Postal Code'. At the bottom left of the screen, the text 'F1=Help' is displayed.

Block Mode applications are sometimes called Green Screen applications, Mainframe applications, and IBM 3270-like applications.

Batch

Another type of application you can build with CA Gen is a Batch application.

Batch applications are characterized as processing large amounts of data at one time, typically on a mainframe. As such, they run in a background mode and have no user interface.

Window

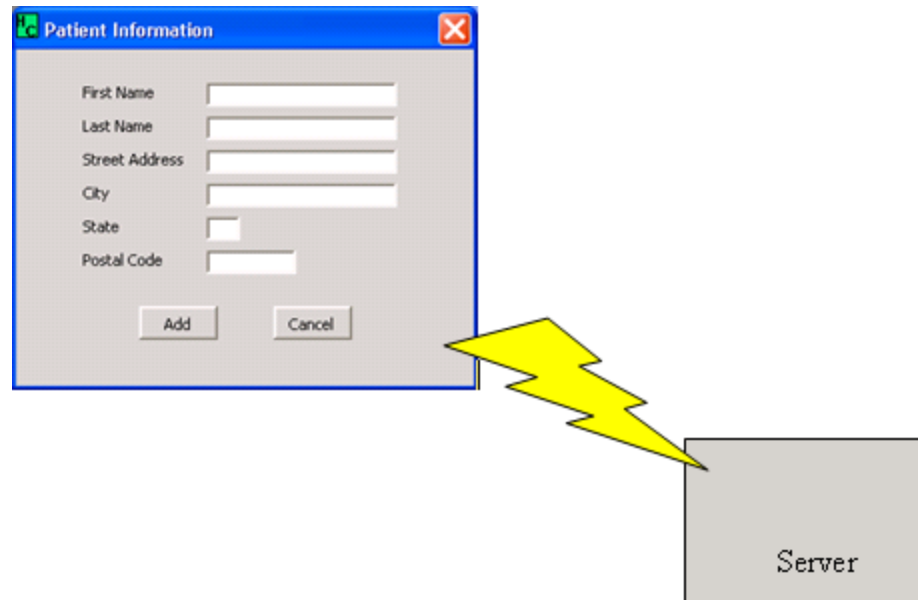
Window applications run under Microsoft Windows operating systems and have a Graphical User Interface (GUI). They can access databases located locally on the same machine, remotely or not at all.

Client/Server

Another type of application supported by CA Gen is the Client/Server type application.

Client/Server applications are made up of two parts:

- The Client procedure—Initiates a request
- The Server procedure—Responds to the request



The processing logic is split between the two programs, which are normally deployed on separate machines.

Proxies

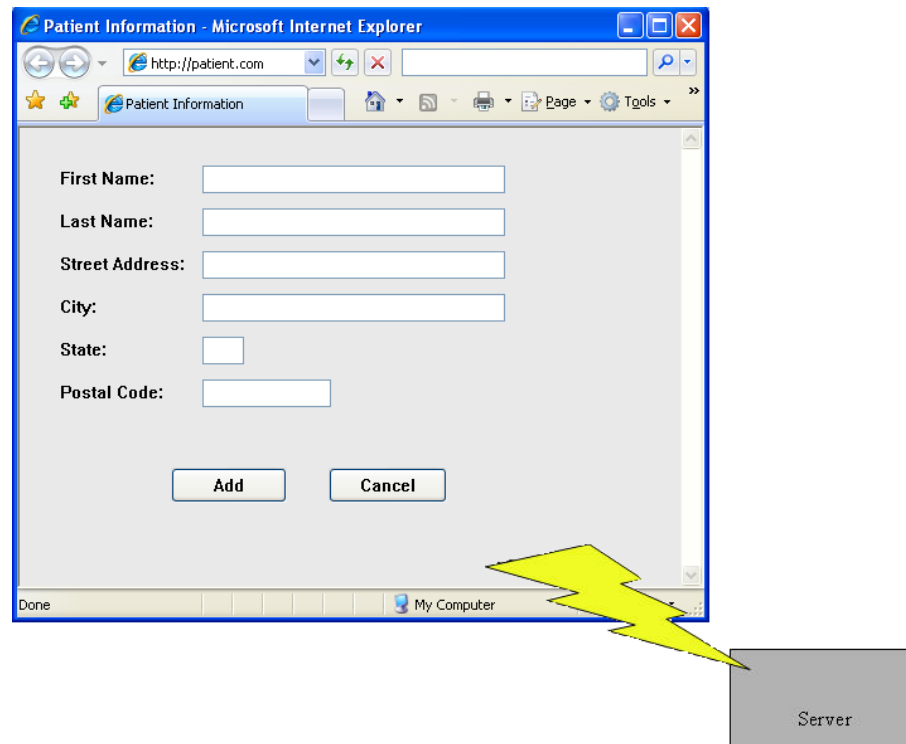
CA Gen also supports Proxies. A proxy is a piece of code generated by CA Gen that acts as an interface to our generated Server procedures. Generated proxies can be called from within a COM, C, Java, or .NET environment.

Proxies would typically be used if you were manually writing or using some other tool to develop your own clients who needed to connect to our generated Servers.

Web

Finally, we have Web applications.

Web applications are similar to window and client/server applications. However, the web Clients User Interface is hosted from a Web Server and accessed through a Web Browser across the World Wide Web. For window applications, all logic runs on the same machine as the Web Server. For client/server applications, the CA Gen Servers can be deployed on the same machine as the Web Server or on a different machine.



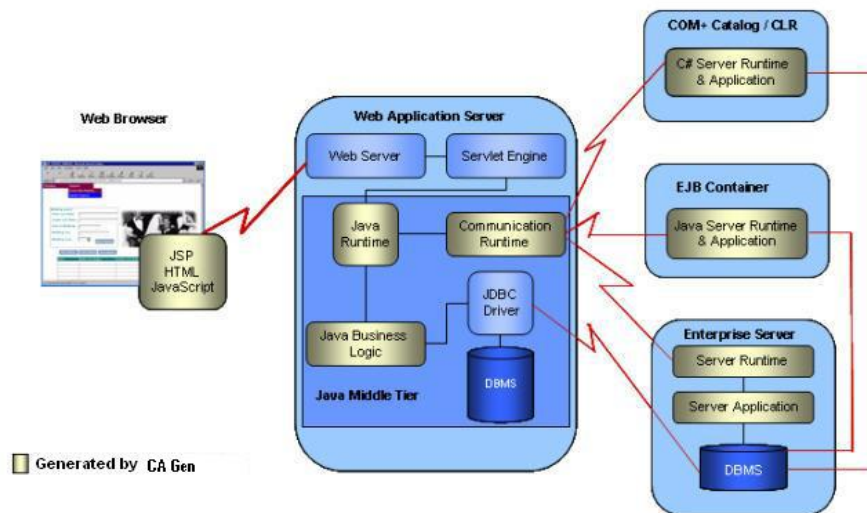
CA Gen supports both J2EE and .NET web technologies. The type of application we are going to build is a Java (J2EE) Web application.

Web Client/Server Applications

In building client/server applications with CA Gen, we generally adhere to a design structure known as Distributed Processing. With Distributed Processing, you have a Client procedure and a Server procedure, which are typically deployed on different machines. In actuality, you will likely have many client procedures and many server procedures, with the clients being deployed as a group on one or more client machines, and the servers being deployed separately or as a group across one or more server machines. CA Gen generates all the code for both programs, as well as any code necessary to support the communications between the two programs.

For Java (J2EE) web applications, Client procedures are generated as a combination of Java Server Pages (JSPs), Hyper Text Markup Language (HTML), JavaScript, and Cascading Style Sheets (CSS), and are deployed to a Web server. Server procedures can be generated as C, COBOL, or Java (EJBs) depending on the environment.

The following example illustrates these and other possible relationships:



Another characteristic of a distributed process system is that the DBMS typically resides on the same machine as the CA Gen Server. Because the elementary processes contain the business rules and most of the data access, performance is generally improved when the Server procedures implement the elementary processes. The Client procedure's logic can then concentrate on presentation, workflow, and some data validation.

Designing Server Procedures

The following sections deal with designing server procedures.

Lesson Objectives and Time Allotment

After this lesson you will understand:

- How Server Procedures are defined
- How Server Procedures are designed
- How to use both manual and automated methods to add procedure step logic

Allow yourself approximately 2 hours and 15 minutes to complete this lesson.

Server Procedures

As mentioned previously, Server Procedures usually implement all of the elementary processes. There can be many factors involved in deciding on how many server procedures are required and what processes are implemented in each procedure. As a starting point for this discussion, you can usually expect there to be two server procedures for each entity type in your model (although we will not be doing that for all entity types in this tutorial). One procedure will contain all of the processes dealing with a single entity of that entity type, and the other procedure will contain the logic necessary to provide a listing of entities for that entity type.

Two factors influence this decision. One is that you would normally create procedures for processes based on the processes having similar view structures. The other is that in a typical client/server application, you first select a single object to be edited from a list of those objects, and then you would edit that single object.

With this as a starting point, we would expect our application to have four server procedures:

- List Golfer
- Maintain Golfer, which implements:
 - Add Golfer
 - Login Golfer
 - Update Golfer
 - Calculate Golfer Handicap Index
 - Delete Golfer
- List Scoring Record
- Maintain Scoring Record, which implements:
 - Add Scoring Record
 - Update Scoring Record
 - Delete Scoring Record

Thinking forward to the workflow and presentation for this application, we can probably dispense with the List Golfer procedure step. It would probably be required from a Customer Service standpoint, that is, a customer calls in and cannot remember their password, so our Web site's Technical Support personnel would have to look the customer up from some sort of a list. Initially, we will not provide this functionality. We will create just three server procedures. However, if we eliminate this functionality, we introduce another small problem. Typically, we get all the attributes of the object we want to display to the user for updating from the list procedure. If we eliminate the list of golfers, then we have to get the attributes to display for the update of golfer from somewhere else. We can easily solve this problem by slightly modifying the Login Golfer elementary process.

Lesson Activity

In this exercise, we are going to:

- Slightly modify the LOGIN GOLFER elementary process to support the display of additional golfer's attributes.
- Create the three new server procedures.
- Manually add the procedure step action diagram logic to MAINTAIN GOLFER
- Use Procedure Synthesis to automate adding the procedure step action diagram logic to MAINTAIN SCORING RECORD
- Manually add the procedure step action diagram logic to LIST SCORING RECORD

Modify the LOGIN GOLFER Process Action Diagram Logic

We need to slightly modify the logic in the LOGIN GOLFER elementary process to return the email address of the golfer. This will then make all of those attributes available to us if the golfer decides they want to update any of them.

Follow these steps:

1. If necessary, open the egolf.ief model in the Toolset. If you need help, see prior lesson.
2. Open the process action diagram for LOGIN GOLFER and expand all of the views. If you need help, see a prior lesson.
3. Under ENTITY ACTIONS, select Entity View **golfer**. From the Main Menu, select **Edit**, then **Add View**, and then **Add Attribute View....** Then, from the list of attributes, select **EMAIL_ADDRESS** and then select the **OK** push button.
4. Under EXPORTS, select Entity View export **golfer**. From the Main Menu, select **Edit**, then **Add View**, and then **Add Attribute View....** Then, from the list of attributes, select **EMAIL_ADDRESS** and then select the **OK** push button.

The completed action diagram should now look like the following example:

```

LOGIN_GOLFER
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
  password (mandatory)
  userid (mandatory)
EXPORTS:
  Entity View export golfer (transient,export only)
  first_name
  last_name
  handicap_index
  email_address
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
  password
  first_name
  last_name
  handicap_index
  userid
  email_address

  READ golfer
    WHERE DESIRED golfer userid IS EQUAL TO import golfer userid
  WHEN successful
    IF golfer password IS EQUAL TO import golfer password
    MOVE golfer TO export golfer
    ELSE
    EXIT STATE IS golfer_password_invalid
  WHEN not found
    EXIT STATE IS golfer_nf

```

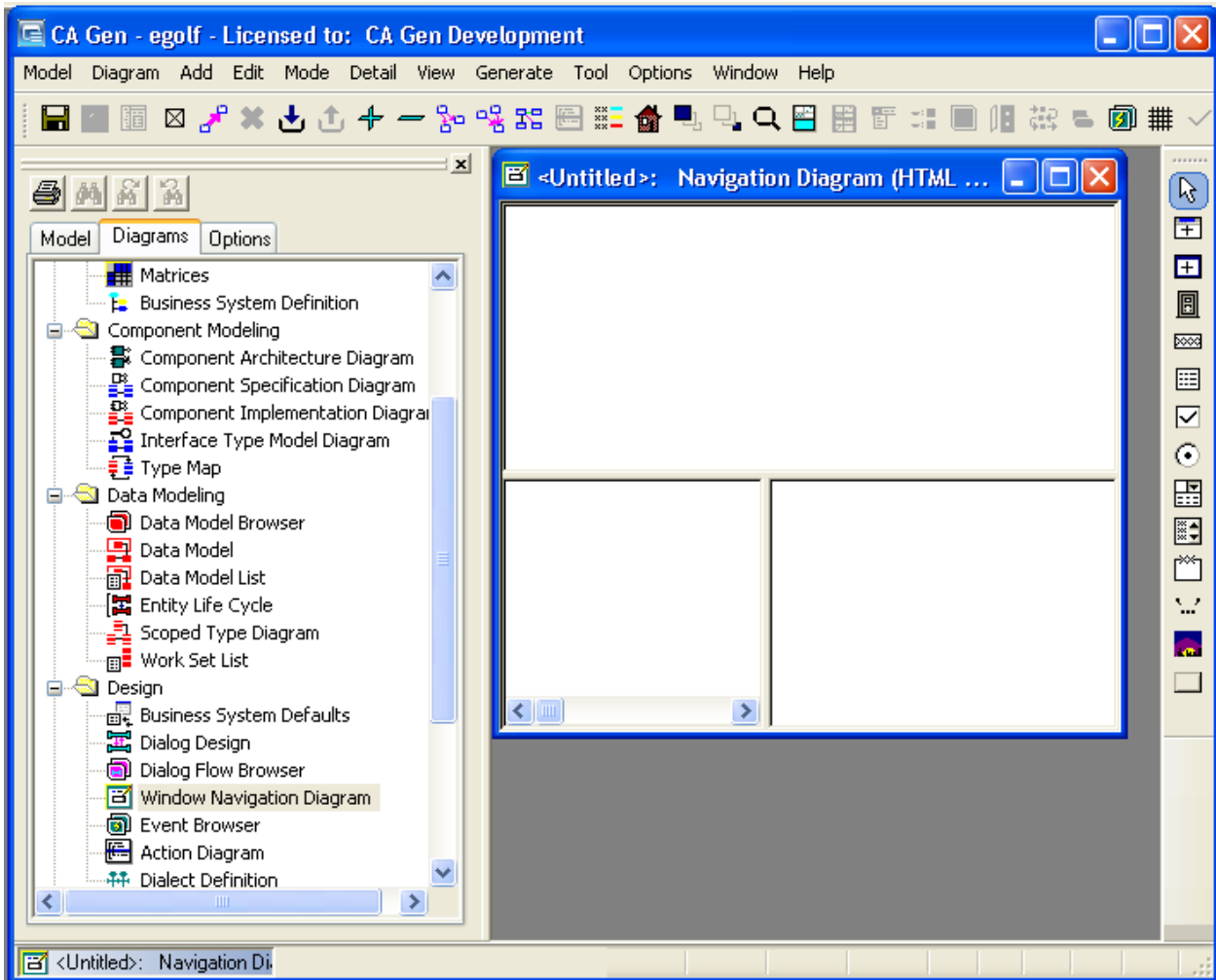
5. Save the model.

Create the Three New Server Procedures

The primary diagram used in client/server design is the Window Navigation Diagram.

Follow these steps:

1. In the tree control under the Design folder, double-click **Window Navigation Diagram**.



Note: The Window Navigation Diagram is made up of three panes:

Network Pane

Contains a graphical representation of the client/server system structure. This structure includes the client procedure's windows and dialogs, the server procedures, and the flows between them.



Hierarchy Pane



Contains an indented list of essentially the same information that is in the Network Pane.

Controls Pane

Lists the specific GUI controls used on a window or dialog selected from either of the other two panes and the specific actions that can occur against those controls.

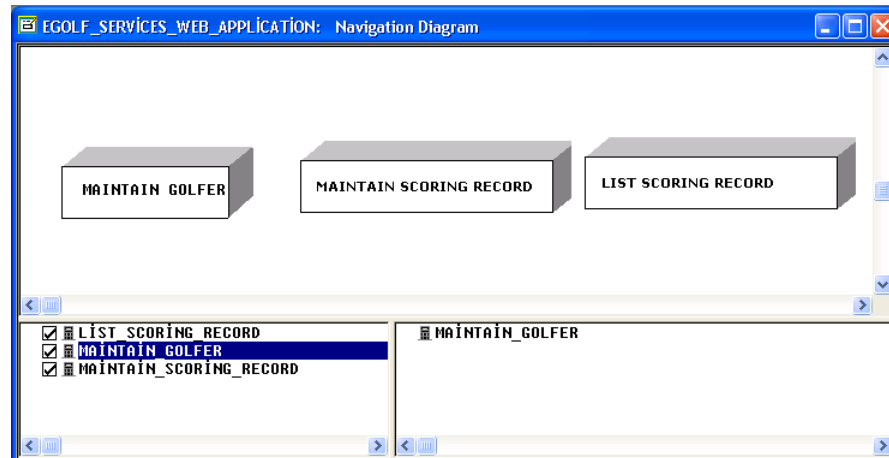
This Navigation Diagram is <Untitled>. You can have multiple Navigation Diagrams; each diagram would then have a name of your choosing assigned to it. We will name and save this navigation diagram now.

2. From the Main Menu, select **Diagram**, then **Create Diagram....**
3. In the subsequent Create Diagram dialog, in the Create Diagram as entry field, enter **egolf services web application**. Select the **Save** push button, and then the **OK** push button when prompted.
4. Save your model.
5. We now want to add the three server procedures. On the Toolbar, select the **Add Server**  icon and then (with the crosshairs ) select anywhere in the Network pane. In the Server/Procedure Step Properties dialog, enter **maintain golfer** in the Server Name entry field and then select the **OK** push button.

The Toolset adds a three-dimensional box representing the server in the Network pane, as well as an entry in the Hierarchy pane. You will probably find it useful to **zoom out**  or **zoom in**  on the diagram by using the appropriate Toolbar options. You can also resize the panes in the Window Navigation Diagram by moving your cursor over the edges of the pane until it turns into double-sided arrow and then dragging the edge to a different position.

Note: If you have made a mistake in the name, you can select the server, select **Detail** from the Main Menu, select **Properties**, and then enter the new name.

6. Zoom out in the diagram and add the maintain scoring record and list scoring record servers. See step number five in the previous section if you need help. Your diagram should look similar to the following example:




You can move objects around in the diagram by selecting the object and then dragging it to where you want it. You can move multiple objects at the same time by selecting each object while holding down the Ctrl key on your keyboard. Then, after selecting the last object, click-and-drag any of the highlighted objects (as a group) to the new location.

7. Save your model.

Add Logic to MAINTAIN GOLFER

Follow these steps:

1. To open the MAINTAIN GOLFER procedure step action diagram, select the MAINTAIN GOLFER procedure in the Network pane or the Hierarchy pane. Then, from the Menu Bar, select the **Action Diagram**  icon.

Notice that Procedure Step Action Diagrams (PrADs) and Process Action Diagrams (PADs) look very similar. They each have Import views, Export views, Local views, and Entity Action view types, each of which are used for the same purposes in each diagram. One of the main differences between the two diagrams, however, is the way in which the views are generally handled. Since process action diagrams (and common action diagrams) usually only perform one activity, we can effectively starve their import and export views, only having in them exactly what is required for that one execution of the process. However, procedures typically support multiple processes. For example, this procedure—Maintain Golfer—is going to support adding new golfers, logging in golfers, updating golfers, calculating their handicap indexes, and deleting golfers.

Individually, each of these processes only uses a small subset of the golfer's attributes. However, collectively, they usually end up using every attribute of the golfer. Therefore, for procedure steps (particularly for those that support multiple processes), the import and export views typically look identical, and typically end up having all of the attributes of the supported entity type in its import and export views. In fact, one of the reasons we chose to put all of the processes in this one procedure was that they had similar view requirements. So let us add the views required for this procedure.

2. In the MAINTAIN GOLFER procedure step action diagram, select **IMPORTS**. Then select **Edit** from the Main Menu, select **Add View**, and then select **Add Entity View....**
3. In the Add Import Entity View dialog, enter **import** in the Name entry field, select entity **GOLFER** in the Entity Types list, select entity **GOLFER** in the Entity Attributes list to select all of the attributes, and then select the **OK** push button. If you have multiple adds turned on, you will then need to select the **Cancel** push button to close the Add Import Entity View dialog.
4. Expand the **IMPORTS**. See a prior section if you need help. Notice that all of the views are marked optional. This is acceptable, since on any one execution of this procedure step, only a subset of the import views will be required.
5. Since the export view is going to look just like the import view, we can copy the import view to the export view. Select Entity View **import golfer** and press **F8** on your keyboard, and with the cursor (which now resembles a hand), select **EXPORTS**. In the Copy Import Entity View dialog, enter **export** for the New name: entry field and then select the **OK** push button. Your export view should now look identical to the import view, notwithstanding the differences in view properties.
6. Save your model.

Diagram Structure

Server Procedure Step Action Diagrams typically follow a certain structure. The following is the structure:

1. Initializing Exit States to a Known Value

Since PADs only set exit state values if they fail, a quick way to determine success or failure when calling an action diagram is to set the special system attribute Exit State to a known value, call the action diagram, and then check to see if the value has been changed by the called action diagram. If you recall, the special system attribute Exit State can only contain one value per user per Business System at any one point in time. Once you have determined that it is failed, and then you can check for individual Exit State values to make appropriate Design processing decisions.

2. Moving the contents of the Import Views to the Export Views

The source of information (the import views) for a procedure step is typically a window (or a screen). The destination of information (the export views) from a procedure step is also typically a window (or a screen). To redisplay the information that was originally entered on the window, along with the new or updated information returned from the PADs, we have to move the import views of the procedure step to the export views of the procedure step. Then, when we call the PADs they will only export those views specific to it, overlaying those views in the export views of the procedure, while leaving all of the other export views unchanged. This updated view would then be redisplayed on the window. This only holds if we properly starve the views of our PADs (and CABs).

3. Invoking the various PADs by way of a Case of Command structure

When a client procedure calls a server procedure, there are often several things the server procedure can do for us. Usually, however, for each call to the server, we only want it to perform one of the many things it may be able to do. To tell the server procedure what we want it to do, we command it. Command is another special system attribute that can only hold one value at any point in time. Similar to the way in which we set the special system attribute Exit State, we typically pre-define a set of Command Values to choose from in setting its value. Then we would set the command to a pre-defined value, call the server procedure, the server procedure would then evaluate the command value, and then perform the appropriate action. The evaluation of the command value is usually performed in a CASE OF statement. A CASE OF statement allows you to evaluate one attribute for a series of values all at once, and then perform just the one set of actions based on the current value.

Initialize the Exit State Value

Follow these steps:

If the Action Diagram Tool Palette or Toolbar is closed, from the Main Menu select Options, and then select (check) each of those options.

1. From the Tool Palette, select the **Exit State** push button and then with the cursor (which has now changed to crosshairs) select the blank line after ENTITY ACTIONS.
2. In the Exit State Selection list dialog, select **<Global Exit States>**, select the **Expand/Contract** push button, and then select the exit state value **PROCESSING OK**. Then, select the **Properties...** push button and change the Message Type: from **Informational** to **None**, and then select the **OK** push button. On the Exit State Selection dialog, select the **Select** push button to add the exit state to the Add Statement dialog, and then select the **Add** push button to add the statement to the diagram.

Move Import Views to Export Views

Follow these steps:

1. From the Tool Palette, select the **Move** push button, and with the cursor (which has now changed to a crosshairs) select the statement **EXIT STATE IS processing ok**, which is the statement we want to add the new statement directly below.
2. In the Add Statement dialog, select **import golfer**. Notice that the Toolset automatically selected the export golfer for the destination. Select the **Add** push button.

Add the CASE OF (COMMAND) Statement

Now we can add the CASE OF (COMMAND) statement. Keep in mind that the CASE OF COMMAND statement is going to evaluate the value of the special system attribute COMMAND to determine which of five different actions we want performed. Those five actions include:

- Add (Register) Golfer
- Login Golfer
- Update Golfer
- Calculate Golfer Handicap Index
- Delete Golfer

Follow these steps:

1. From the Main Menu, select **Edit**, then Add Statement, and then **Case of** In the Add Statement selection list, select **command**, then select the **Complete** push button, and then in the Add Statement selection list select **command value**.

The Toolset then brings up a list of the command values. While it is a complete list (for example, we can use the command CREATE to Add Golfer, UPDATE to Update Golfer, and DELETE to Delete Golfer), there are not any appropriate command values for Login Golfer or Calculate Golfer Handicap Index. Additionally, we can prefer to use a command of REGISTER for Add Golfer. We can add new command values to this default list.

2. Select the **Add Command...** push button. In the Command Properties dialog New Name entry field, enter the new command value **REGISTER** and select the **OK** push button. If you have multiple adds turned on, it will add REGISTER to the list in the background while leaving the Command Properties dialog open. If you do not have multiple adds turned on, select the **Add Command...** push button once again. Either way, add the additional new commands of **LOGIN** and **HANDICAP**.
3. When you are finished, select the **Cancel** push button in the Command Properties dialog if necessary to return to the Command Selection dialog. Then, select **REGISTER** from the list of command values, and then select the **OK** push button. Select **CASE** from the Add Statement selection list, select **command value** from the selection list, select **LOGIN** from the Command Selection list, and then select the OK push button.
4. Repeat this process for the remaining three commands, **UPDATE**, **HANDICAP**, and **DELETE**. When you are finished, select the **Add** push button to add the CASE OF COMMAND statement to the action diagram.

CASE Register

The final step is to add the calls to the appropriate process action diagrams. The import views of each used action diagram will be matched to the import views of the server procedure and the export views of each used action diagram will be matched to the export views of the server procedure.

Follow these steps:

1. In the CASE OF COMMAND statement, select the CASE statement next to the command value register. Then, from the Main Menu select Edit, then Add Statement, and then select Use.... From the list of action blocks in the Add Statement dialog, select add golfer, and then select the Add push button.

Any time you use an action block, the Toolset steps you through the process of matching views. In the top panel of the ADD GOLFER Import View Matching dialog, the Toolset is showing you all of the import views of the ADD GOLFER action diagram on the left side, with the views from the MAINTAIN GOLFER server procedure step action diagram that have been matched to them. Presently none are matched. We need to systematically select one import view in the called action block, match it to the appropriate view from the calling action block, and repeat that at a minimum for each required import view in the called action block.

2. In the ADD GOLFER action diagram, there is only one required import view. Select IMPORT from the left side of the top panel, select IMPORT from the supplying views in the lower panel, and then select the Match push button. Notice that the top panel now shows that IMPORT GOLFER from the called action diagram ADD GOLFER has been matched to the IMPORT GOLFER in the calling action diagram MAINTAIN GOLFER.
3. Select the Close push button to be presented with the export view-matching dialog. In the ADD GOLFER Export View Matching dialog, no export views exist for the ADD GOLFER process action diagram. Therefore, no view matching is necessary. Select the Close push button.

CASE Login

We will perform similar steps for the CASE login.

Follow these steps:

1. Select the **CASE** statement next to the command value login. From the Main Menu select **Edit, Add Statement**, and then **Use....** From the list of action blocks in the Add Statement dialog, select **login golfer**, and then select the **Add** push button.
2. The import view of the LOGIN GOLFER process requires the golfer's userid and password. To verify this, in the LOGIN GOLFER Import View Matching dialog, select the LOGIN GOLFER import view **IMPORT**. Then select the **Expand** push button. You can now see the specific attributes in the import view of LOGIN GOLFER.

3. We want to match the import view of the LOGIN GOLFER process action diagram to the import view of the MAINTAIN GOLFER procedure step. Select **IMPORT** from the supplying views in the bottom panel and then select the **Match** push button.
4. The views are contracted after being matched. To see the specific attributes again, select the **IMPORT** in the top panel again and then select the **Expand** push button. Notice that while the import view of the procedure step has all of the attributes of the golfer, only the userid and password are passed (matched) to the Login Golfer process action diagram.
5. Select the **Close** push button to be presented with the export view-matching dialog. The export view of LOGIN GOLFER returns all of the attributes of the golfer, minus the userid and password. Verify this by expanding the export views.
6. Since we want the server procedure to return this information to the client procedure (which we have not written yet), we need to match this export view of LOGIN GOLFER to the export view of the procedure step MAINTAIN GOLFER. In the top panel select **EXPORT**, in the list of possible receiving views select **EXPORT**, and then select the Match push button. In the top panel, we can now see that the export view from LOGIN GOLFER is matched to the export view in MAINTAIN GOLFER. You can expand the views to see exactly which attributes are matched. Select the **Close** push button.

CASE Update

We will perform similar steps for the CASE update.

Follow these steps:

1. Select the **CASE** statement next to the command value update. Then, from the Main Menu select **Edit**, then **Add Statement**, and then select **Use...** From the list of action blocks in the Add Statement dialog, select **update golfer**, and then select the **Add** push button.
2. The import view of the UPDATE GOLFER process requires all of the golfer's attributes except handicap index. To verify this, in the UPDATE GOLFER Import View Matching dialog, select the UPDATE GOLFER import view **IMPORT**. Then select the **Expand** push button. You can now see the specific attributes in the import view of UPDATE GOLFER.
3. We want to match the import view of the UPDATE GOLFER process action diagram to the import view of the MAINTAIN GOLFER procedure step. Select **IMPORT** from the supplying views in the bottom panel and then select the **Match** push button.
4. The views are contracted after being matched. To see the specific attributes again, you will need to select the **IMPORT** in the top panel again and then select the **Expand** push button. Select the **Close** push button to be presented with the export view-matching dialog.
5. In the UPDATE GOLFER Export View Matching dialog, no export views exist for the UPDATE GOLFER process action diagram. Therefore, no view matching is necessary. Select the **Close** push button.

6. Repeat the previous steps for the **CASE handicap** using the calculate golfer handicap index action diagram.
7. Repeat the previous steps for the **CASE delete** using the delete golfer action diagram.

CASE Handicap

Repeat the previous steps for the **CASE handicap** using the calculate golfer handicap index action diagram.

CASE Delete

Repeat the previous steps for the **CASE delete** using the delete golfer action diagram.

Set Exit States

Follow these steps:

1. Finally, from the Tool Palette, select the **Exit State** push button. With the cursor (which now resembles a crosshairs) select the **OTHERWISE** condition of the CASE OF COMMAND statement, and then select **<Global Exit States>** from the Exit State Selection list
2. Select the **Expand/Contract** push button if necessary, then select the **INVALID COMMAND** exit state value, then the **Properties...** push button. Change the drop-down Message Type: to None. Then select the **OK** push button, select the **Select** push button, and then select the **Add** push button.
3. Save the model.

The completed Procedure Step Action Diagram should look like the following example:

```

MAINTAIN_GOLFER
IMPORTS:
  Entity View import golfer (optional,transient,import only)
  password (optional)
  first_name (optional)
  last_name (optional)
  email_address (optional)
  handicap_index (optional)
  userid (optional)
EXPORTS:
  Entity View export golfer (transient,export only)
  password
  first_name
  last_name
  email_address
  handicap_index
  userid
LOCALS:
ENTITY ACTIONS:

EXIT STATE IS processing_ok
MOVE import golfer TO export golfer
CASE OF COMMAND
CASE register
  USE add golfer
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
CASE login
  USE login_golfer
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
    WHICH EXPORTS: Entity View export golfer FROM Entity View export golfer
CASE update
  USE update_golfer
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
CASE handicap
  USE calculate_golfer_handicap_index
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
    WHICH EXPORTS: Entity View export golfer FROM Entity View export golfer
CASE delete
  USE delete_golfer
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
OTHERWISE
  EXIT STATE IS invalid_command
  
```

Processing Flow

This section describes the processing flow for a typical CAGen scenario. The flow for your project can be different.

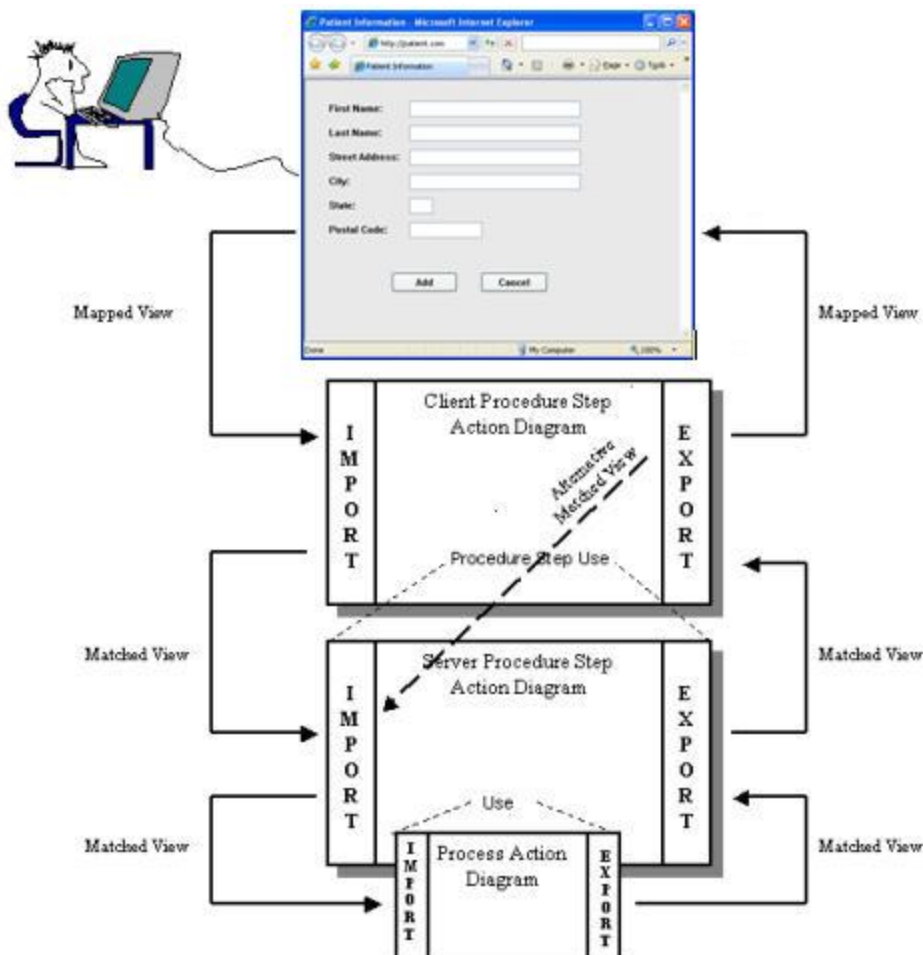
Note: You can follow the picture on the next page as you read this description.

Information entered on a window is mapped to an import view of a client procedure step. The client procedure step will move the import view information to an appropriate export view and do some initial validation of the entered data. If there is a problem, the entered data can now be redisplayed to the user since it is in the export view, and errors can be indicated in some way. If there is no problem, then the appropriate server can be called with a procedure step use statement after the appropriate command value has been set. Information from either the client's import or export views will be matched to the import view of the called server procedure step.

The called server procedure step will move the import views to its export views and then check the command value passed to it. If the command is invalid for that server, an appropriate exit state value is set, and control is returned to the client procedure step along with the information passed to the server unchanged.

If the command is valid, then the appropriate process action diagram is called with a use statement, matching the import view of the server to the import view of the process. The process will take the information given to it in its import views and update the database with it. If the process logic changes the values of the information passed to it (which is not the same thing as simply taking the values as given and updating the database), or creates entirely new information internally within the process logic, then this new or changed information is exported from the process to overlay the original values (which can have been blank) in the export view of the server procedure step. This assumes that the process action diagram views are properly starved. Then, the server's export views are matched back to overlay the export views of the client and the client's export views are mapped back to the window. The client can evaluate the exit state value set by the server to determine if the server call was successful and take appropriate actions if necessary.

The following example depicts this process:



Procedure Synthesis

We now want to add the procedure step action diagram logic to the Maintain Scoring Record server using Procedure Synthesis. Procedure Synthesis is similar to Action Block Synthesis in that it generates a first cut at the action diagram logic. However, Procedure Synthesis attempts to tailor the action diagram logic more for what procedures typically do. This generated logic would then have to be reviewed and potentially modified to ensure it does exactly what you need it to do.

Six stereotypes can be used for Procedure Synthesis. These stereotypes are essentially internal templates or patterns used to generate the action diagram logic. They are particularly well suited to generating blockmode type applications and include the generation of menu type procedures, as well as various types of list and maintenance procedures. In addition, most have several options available to customize the generated logic.

Two of them are especially useful for generating the server procedure logic for client/server type applications. They are the Entity Maintenance stereotype and the Implements Action Blocks stereotype.

We are going to use the Implements Action Blocks stereotype to implement the Add Scoring Record, Update Scoring Record, and Delete Scoring Record processes.

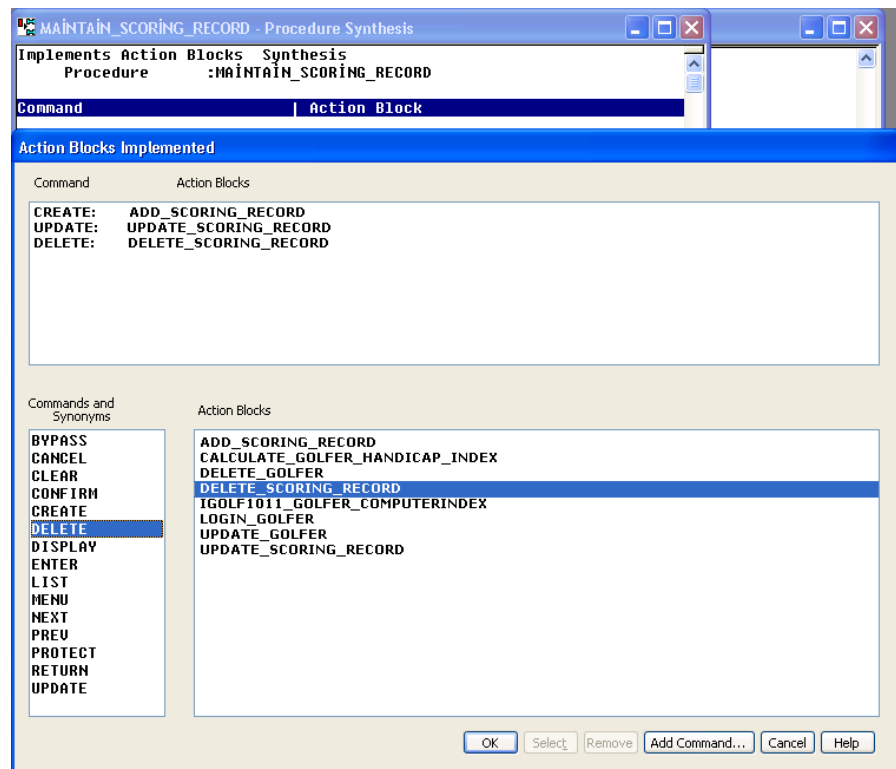
Add Logic to MAINTAIN SCORING RECORD Using Procedure Synthesis


Follow these steps:

1. From the Navigation Diagram, select the **MAINTAIN SCORING RECORD** server procedure.
2. From the Main Menu select **Generate**, and then select **Procedure Synthesis**.
3. Notice in the MAINTAIN SCORING RECORD – Procedure Synthesis panel that the Stereotype is not selected for Synthesis. We now want to select the stereotype. In the MAINTAIN SCORING RECORD – Procedure Synthesis panel select **Procedure :MAINTAIN SCORING RECORD**. Then, from the Main Menu, select **Edit**, then **Stereotype**, and then select **Implements Action Blocks....** Notice that Stereotype not selected for Synthesis has been replaced with Implements Action Blocks Synthesis.
4. Thinking back to the Maintain Golfer server, the invocation of each process action diagram was associated in the CASE OF COMMAND statement with a particular Command value. Thus, for this new server procedure step action diagram, we need to specify which command value will invoke which action diagram. In the MAINTAIN SCORING RECORD – Procedure Synthesis panel, select **Command | Action Block**. Then, from the Main Menu select **Edit**, and then **Select**.

5. In the Action Blocks Implemented dialog, we need to specify which Commands are to be associated with which action diagrams. Select the command value **CREATE** from the listbox on the left and the action diagram **ADD SCORING RECORD** from the listbox on the right, then select the **Select** push button. The command and the action block are added to the panel at the top of the Action Blocks Implemented dialog.
6. Repeat this procedure for the command value **UPDATE**, the action block **UPDATE SCORING RECORD**, the command value **DELETE**, and the action block **DELETE SCORING RECORD**.

Your diagram should now look like the following example:



7. In the Action Blocks Implemented dialog, select the **OK** push button to add these to the MAINTAIN SCORING RECORD – Procedure Synthesis panel.
8. Now, we need to actually apply these choices. From the Main Menu, select **Edit**, and then select **Apply**.
9. Close the MAINTAIN SCORING RECORD – Procedure Synthesis panel by selecting the  to the far right of the Main Menu.

Edit the Action Diagram

eGolfer Home

Place Your Ad Here

Personal Profile

update remove Welcome XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Your Handicap Index is ZZ9.9


Scoring Record

add update remove

Date	Time	Score	Rating	Slope	Note
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX

logout

Follow these steps:

1. In the Navigation Diagram, select the **Action Diagram**  icon from the Tool Bar to open the MAINTAIN SCORING RECORD procedure step action diagram and review the results of our Procedure Synthesis. Expand the Views. We now have the CASE OF COMMAND statement, the action block use statements, and views created to support the view requirements of the called action diagrams. There are no export views because none of the called action blocks have export views.

Are there any changes that need to be made? Thinking back to the basic structure of a server procedure step action diagram, there were three things that were commonly done:

- a. Initializing the Exit State to a known value
- b. Moving the contents of the import views to the export views

c. Invoking the various PADs by way of a Case of Command structure

Thus, we need to make a few changes.

2. At the top of the action diagram, set the exit state value to PROCESSING OK. See a prior section if you need help.
3. Normally, we would have to move the import views to the export views. Since this particular server procedure requires no export views, this will not be necessary. However, it will be necessary to mark some of the import views as optional, since every import view is not required for every invocation of the server. For example, to delete a scoring record, only the golfer's userid and the scoring record's date and time are needed. In fact, since those three attribute views are required for all three processes, they can be left as mandatory while everything else is marked as optional. See a prior section if you need help changing the import view properties to optional.
4. The two notes added during synthesis are not providing us with any value. Delete the two notes.
5. Save the model.

The completed action diagram should look like the following example. The import views have been reordered manually.

```

MAINTAIN_SCORING_RECORD
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
  userid (mandatory)
  Entity View import scoring_record (mandatory,transient,import only)
  date (mandatory)
  time (mandatory)
  adjusted_gross_score (optional)
  course_rating (optional)
  course_slope_rating (optional)
  note (optional)
EXPORTS:
LOCALS:
ENTITY ACTIONS:


EXIT STATE IS processing_ok
CASE OF COMMAND
CASE delete
  USE delete_scoring_record
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
                  Entity View import scoring_record TO Entity View import scoring_record
CASE update
  USE update_scoring_record
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
                  Entity View import scoring_record TO Entity View import scoring_record
CASE create
  USE add_scoring_record
    WHICH IMPORTS: Entity View import scoring_record TO Entity View import scoring_record
                  Entity View import golfer TO Entity View import golfer
OTHERWISE
  EXIT STATE IS invalid_command

```

Add Logic to LIST SCORING RECORD Using Action Block Synthesis

To create the LIST SCORING RECORD action diagram, we will use Action Block Synthesis to give us a quick start, but then we will need to make a fair amount of modifications to finish it.

Follow these steps:

1. In the Navigation Diagram, select the **LIST SCORING RECORD** server, and then select the **Action Diagram**  icon from the Tool Bar.
2. With the procedure step action diagram open, from the Main Menu select **Generate**, and then select **Action Block Synthesis...**
3. In the Action Block Synthesis dialog, select **Entity Type SCORING RECORD**. In the Mode: drop-down list, select **List**. Then, in the Action Dialog Options, remove the checkmark next to Consistency Check on Entity Type. Select the **OK** push button to create the action diagram.

Edit the Action Diagram

Follow these steps:

1. Expand all of the views. Notice that the Toolset created the export views necessary to produce a list of 1000 Scoring Records. That is because in the properties for the Scoring Record entity type, we set the maximum at 1000. Also, notice that it just lists Scoring Records in general, and not just those associated with a particular Golfer. In the properties for the relationship, we said a Golfer would maintain at most 40 Scoring Records, so we will change the export view to reflect that number, and change the READ EACH statement to only read Scoring Records associated with a particular Golfer. Other changes we will make will be to put the action diagram into the standard format for a procedure step action diagram (for example, initialize the exit state, put the logic within a case of command, and so on).
2. To provide a list of scoring records for a particular golfer, we need to know which golfer we want the scoring records for. Therefore, we need a view of Golfer in the import view. In the action diagram, select **IMPORTS:**, then from the Main Menu select **Edit**, then **Add View**, and then **Add Entity View...**
3. In the Add Import Entity View dialog, enter **import** in the Name entry field, change the drop-down menu to **Always used as input**, select **GOLFER** from the list of Entity Types, select **USERID** from the list of Entity Attributes, and then select the **OK** push button. If you have multiple adds turned on, select the **Cancel** push button to return to the action diagram.

4. Now, given a particular Golfer Userid in our import views, we can produce a list of their Scoring Records in our export views. However, our export views can use a little cleanup as well. Notice that in our export views we have a Group View with a name of group export that occurs 1000 times. Within the group view (as indicated by the indentation), we have an Entity View of Scoring Record named export. Essentially, what we have is an array containing 1000 occurrences of the information contained in the Scoring Record.

Since group views have nothing more than a name and a number of occurrences, it is generally good practice to give them meaningful names. This is particularly important when you have many group views in an action diagram.

Double-click **group export** in the EXPORTS: and change the Name to **export group of scoring records**. Then, change the cardinality of the group view to an average of **20** and a maximum of **40**, and then select the **OK** push button.

5. Since the export scoring record is part of the group view, we want to rename it from **export** to **export group**. Double-click the view name **export** and change the Name to **export group**. Select the **OK** push button.
6. The final thing we want to do with the export group view is to add a selection character. While a selection character is not needed for any processing logic we are going to be doing within the server procedure, it will be necessary for processing logic we will be doing within the client procedure. Since this export group view will be matched to an export group view in the client procedure that will contain the selection character, we need to add a selection character to this view to enable the view matching. To match a group view to another group view, the two views must have similar view structures.

To add a view to the group view, select export group of scoring records. Then, from the Main Menu select **Edit**, then **Add View**, and **Add Work View**. In the Add Export Work View dialog, enter **export group** for the Name, select the work set **IEF SUPPLIED** from the list of Entity Types, select the attribute **SELECT CHAR** from the list of Work Attributes, and then select the **OK** push button. If you have multiple adds turned on, you will need to select the **Cancel** push button to return to the action diagram. Notice that the Selection Character has been added to the Group View as indicated by its indentation under the Group View name.

Add an Entity View of Golfer

You should now be becoming more familiar with the three categories of views. The three categories are:

Entity views

Views of any information that you have modeled in your data model, an entity type. These would contain information about objects of interest to your business, like Golfers and Scoring Records.

Work views

Views of information needed that are not modeled in your data model. Examples of these would be things like selection characters, counters, flags, and subscripts. These would typically not be in a business data model, but can be required to support some functioning within your programs.

Group views

Mechanisms for grouping and creating arrays of other views. These other views can be entity views, work views, or even additional group views. What we have created earlier is an array of information that contains the combinations of a Select Character along with the Scoring Record's Date, Time, Adjusted Gross Score, Course Rating, Course Slope Rating, and Notice, which then occurs 40 times.

Within our action diagram logic, we will modify the READ EACH statement to read each scoring record for a given golfer, and for each scoring record we find, we will move it to the group view. Since our group view is using implicit indexing, CA Gen will handle incrementing the subscript for us automatically to move each occurrence found to the next position in the array. To support the read of Golfer, the final change we need to make to our views is to add an entity view of Golfer to our Entity Action views. Since this view will look just like the import view of Golfer, we can copy the import golfer to our ENTITY ACTIONS views.

Follow these steps:

1. Select **import golfer** in the IMPORTS:. Then press **F8** to copy this view, and with the cursor (which now resembles a hand) select **ENTITY ACTIONS**. Since we generally do not name our entity action views, leave the New name: field blank and select the **OK** push button to complete copying this view.
2. Save the model now that the views are complete.

Modify the READ EACH Statement

Follow these steps:

1. To modify the READ EACH statement, select **UNTIL FULL** in the TARGETING clause. Then, from the Main Menu select **Edit**, and toward the bottom of the drop-down menu select **Insert After....**

2. Now select the qualifier **Sorted Descending**, select the attribute **date**, select the qualifier **Sorted Descending**, select the attribute **time**, select the qualifier **Where expression**, select the expression **relationship view**, select the occurrence phrase **DESIRED persistent view**, select the Entity Relationship **is maintained by SOME golfer**, select the expression **AND**, select the expression **attribute view**, select the occurrence phrase **THAT persistent view**, select the relational operator **IS EQUAL TO**, select the expression **character view**, select the occurrence phrase **transient view**, select the entity view **import golfer**, and then select the **Add** push button.

Now the completed READ EACH statement will read each scoring record for some golfer whose userid was passed to it in the import view, returning them one at a time to the action diagram in the diagram's entity action view. They will be returned sorted by date, and time within the same date, and for each one found it will perform the actions described within the READ EACH statements brackets, which is simply to move it to a position in the export group view. This process will continue until it either runs out of scoring records to read for the given golfer, or when it fills up the export group view with 40 occurrences.

3. Unlike Create, Read, and Update statements, Read Each statements have no exception conditions. They simply perform the actions within their brackets for each occurrence found. If none are found, they perform no actions. However, we can test the export group view to see if it is been populated, and set an appropriate exit state value if it has not.

To add this new statement, select the bottom of the READ EACH bracket under the MOVE statement. From the Main Menu select **Edit**, then **Add Statement**, then select **If....**

4. Now select the expression **group view**, select the repeating group view **export group of scoring records**, select the repeating group view condition **IS EMPTY**, and then select the **Add** push button.
5. From the Tool Palette select the **Exit State** push button, and with the cursor (which now resembles a crosshair) select the **IF statement**
6. Now select **<Global Exit States>**, select the **Expand/Contract** push button if necessary, scroll down and select the **SCORING RECORD NF** exit state value, select the **Select** push button, and then select the **Add** push button.
7. Add a **CASE OF COMMAND** statement that performs this logic when the command value is **LIST**. Look back to a prior section if you need help.
8. The next step is to move these two statements within the CASE OF COMMAND statement. To do so, click-and-drag the cursor from the **READ EACH** to the bracket at the bottom of the **IF statement** to highlight both statements. (Be careful not to select the blank line above the READ EACH statement.) Then press **F7** and with the cursor (which now resembles a hand), select the **CASE statement** to move these two statements underneath.
9. Set an exit state value within the CASE OF COMMAND statement to handle the condition when an invalid command is passed to the action diagram.

10. Initialize the exit state value at the beginning of the action diagram.
11. Save your model.

The completed action diagram should look like the following example:

```

LIST_SCORING_RECORD
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
  userid (mandatory)
EXPORTS:
  Group View export_group_of_scoring_records (40,implicit,export only)
  Work View export_group ief_supplied (transient)
  select_char
  Entity View export_group scoring_record (transient)
  date
  time
  adjusted_gross_score
  course_rating
  course_slope_rating
  note
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
  userid
  Entity View scoring_record
  date
  time
  adjusted_gross_score
  course_rating
  course_slope_rating
  note

EXIT STATE IS processing_ok
CASE OF COMMAND
CASE list
  READ EACH scoring_record
    TARGETING export_group_of_scoring_records FROM THE BEGINNING UNTIL FULL
    SORTED BY DESCENDING scoring_record date
    SORTED BY DESCENDING scoring_record time
    WHERE DESIRED scoring_record is maintained by SOME golfer
    AND THAT golfer userid IS EQUAL TO import golfer userid
    MOVE scoring_record TO export_group scoring_record
  IF export_group_of_scoring_records IS EMPTY
    EXIT STATE IS scoring_record_nf
  OTHERWISE
    EXIT STATE IS invalid_command
  
```

Designing Client Procedures

The following sections detail designing client procedures.

Lesson Objectives and Time Allotment

After this lesson you will understand:

- How Client Procedures are defined
- How Client Procedures are designed

Allow yourself approximately 6 hours and 15 minutes to complete this lesson.

Client Procedures

With all the Elementary Processes implemented in the Server Procedures, the Client Procedures can concentrate on presentation, workflow, some data validation, and the invocation of the Server Procedures with the appropriate information.

Our Web application is going to have two client procedures:

- eGolf Services Home
 - eGolfer Registration
 - eGolfer Login
- eGolfer Home
 - Update eGolfer
 - Remove eGolfer
 - Add Scoring Record
 - Update Scoring Record
 - Remove Scoring Record

As with the servers, the number of client procedures is somewhat arbitrary, but it can be argued that having two procedures makes logical sense. The eGolf Services home page will allow new golfers to register to use the system or existing users to login. After registering or logging in, they will then be directed to their very own eGolfer home page. Their own eGolfer home will display their scoring records as well as their current Handicap Index. From their home page, golfers can then add new scoring records, or update or delete existing scoring records. In addition, they can update their personal information or remove themselves entirely from the system.

Each of the previous bulleted items will have its own Web page, so we will be designing nine Web pages. Web pages and Windows clients are designed using the Window Navigation Diagram, just like the servers. Each Web page will be represented in the Navigation Diagram as either a window or a dialog. Each client procedure will have one and only one primary window or primary dialog.

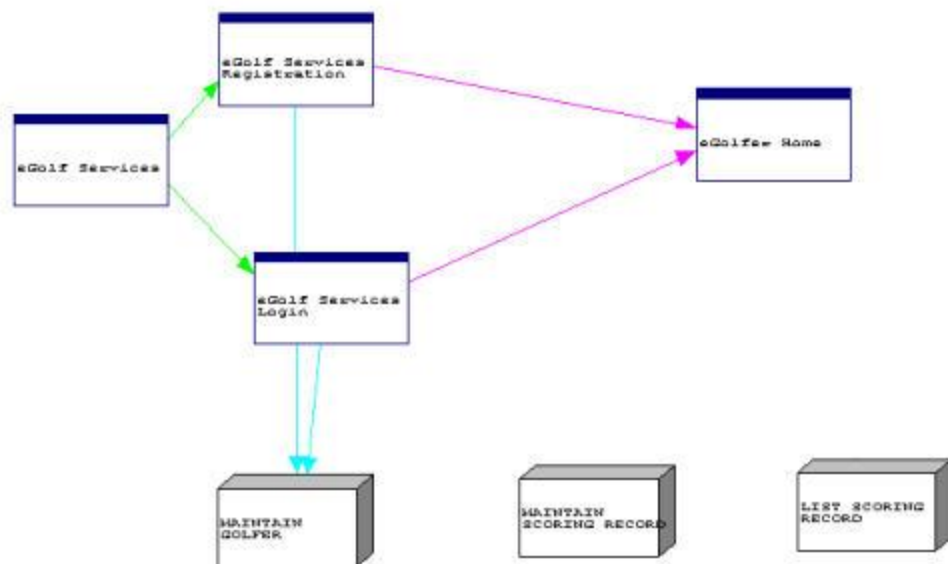
The main difference between a primary window and a primary dialog is that a primary window can be resized and can contain a menu bar, whereas a primary dialog cannot. Since this particular application will not make use of menu bars, we will be using primary dialogs for each of the two procedure steps.

In addition to a primary window or a primary dialog, each client procedure can also contain any number of secondary dialogs. For our Web page design, in addition to the two primary dialogs (one for each client procedure), we will make use of seven secondary dialogs. Looking at the structure of the previous bulleted list, we can see that eGolf Services Home will have two secondary dialogs, and eGolfer Home will have five secondary dialogs. All of the windows or dialogs associated with one client procedure step share that procedure step's action diagram and views.

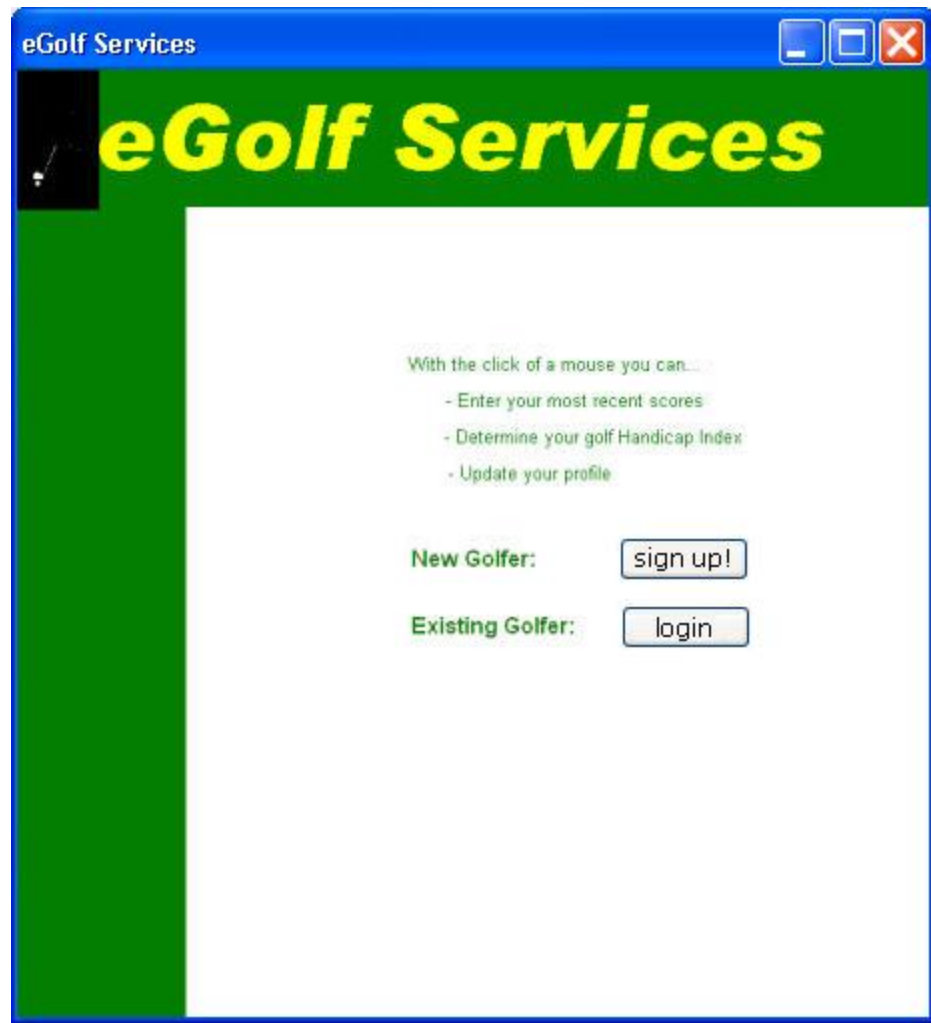
eGolf Services Home Page

From the eGolf Services home page, golfers can register to use our service, or if they are already registered, they can login. In either case, we will have to call the Maintain Golfer server procedure to either add them to the system or to verify their password and log them in. Once they are successfully registered or logged in, we can direct them to their eGolfer home page.

The previous basic system flow is depicted in the following Navigation Diagram:



The eGolf Services home page will appear as shown:



The eGolf Services Registration Web page appears as shown:

eGolf Services Registration

eGolfer Information

The User ID and Password will be used by you to login to eGolf Services.

The User ID can be up to 8 alphanumeric characters of your choosing, but must be unique within our website. We will let you know if it already exists.

User ID:

To ensure the security of your personal information, please designate a password of up to 8 alphanumeric characters.

Password:

Confirm Password:

eGolf Services maintains minimal information about our members, only so that we can personalize your web experience. All fields are required.

First Name:

Last Name:

Email Address:

Please review your information before continuing. Thank you!

< Back Next >

The eGolf Services Login Web page will appear as shown:



The screenshot shows a web browser window titled "eGolf Services Login". The page has a green header bar with a small graphic of two golfers on the left. Below the header, the text "eGolfer Login" is centered. Underneath, there are two input fields: "User ID:" and "Password:", both containing masked characters (X's). At the bottom, there are two buttons: "< Back" and "login".

eGolf Services Login

eGolfer Login

User ID:

Password:

< Back login

The eGolfer Home page will appear as shown:

eGolfer Home

Place Your Ad Here

Personal Profile

update remove

Welcome XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Your Handicap Index is ZZ9.9

Scoring Record

add update remove

Date	Time	Score	Rating	Slope	Note
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX

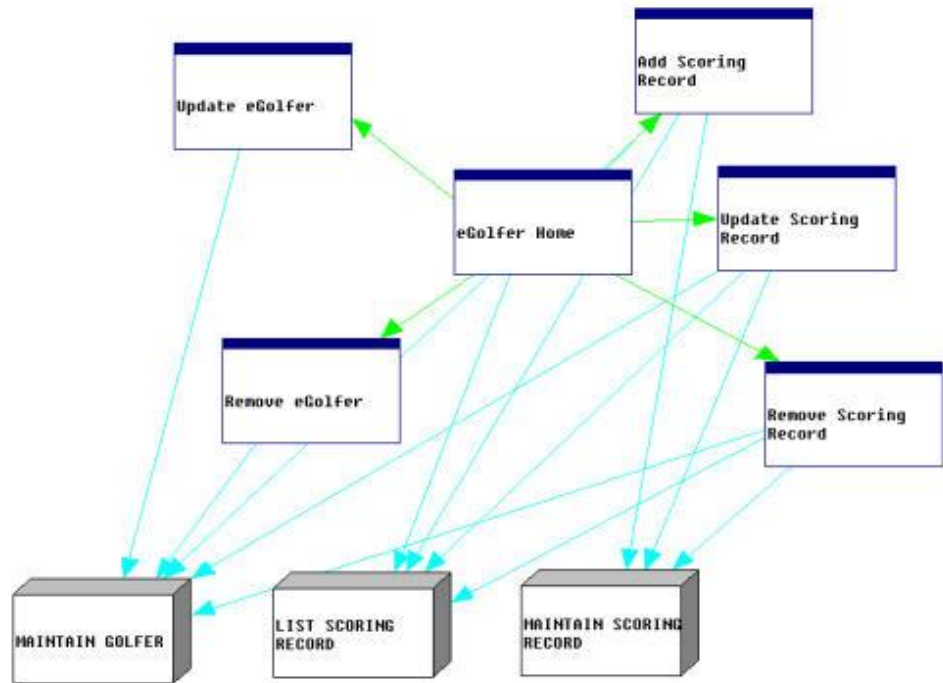
logout

Once directed to the eGolfer Home page, users can update their profile information or remove themselves entirely from our database. To do so, we will call the Maintain Golfer server procedure.

In addition, golfers will be able to add, update, and remove scoring records. Performing any of these actions will require:

- Adding, updating or deleting scoring records from the database, functions that are performed by the Maintain Scoring Record server procedure.
- Re-populating the Web page with the updated list of scoring records, a function that is performed by the List Scoring Record server procedure.
- Recalculating the golfer's handicap index based on the new and updated scoring records, a function that is performed by the Maintain Golfer server procedure.

The previous basic system flow is depicted in the following Navigation Diagram:



The Update eGolfer Information window will appear as shown:

update eGolfer

Place Your Ad Here

Update eGolfer Information

Your User ID is permanent and cannot be updated.

User ID:

To ensure the security of your personal information, please designate a password of up to 8 alphanumeric characters. Your User ID and Password are required to login to the system.

Password:

Confirm Password:

eGolf Services maintains minimal information about our members, only so that we can personalize your web experience. All fields are required.

First Name:

Last Name:

Email Address:

Please review your information before updating. Thank you!

update

cancel

The Remove eGolfer window will appear as shown:

remove eGolfer

Place Your Ad Here

Remove eGolfer

Are you sure you want to be completely removed from our database?

User ID: xxxxxxxx

First Name: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Last Name: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Email Address: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

remove cancel

The Add Scoring Record page will appear as shown:

Add Scoring Record

Place Your Ad Here

Add Scoring Record

Please enter the following scoring information about your game of golf

Date Played: (MMDDYY)

Time Played: (HHMM)

Adjusted Gross Score:

Course Rating:

Course Slope Rating:

Note:

Please review your information before continuing. Thank you!

The Remove Scoring Record window will appear as shown:

[illegible]

eGolfer Home Page

The eGolfer Home page will appear as shown:

eGolfer Home

Place Your Ad Here

Personal Profile

Welcome XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Your Handicap Index is ZZ9.9

Scoring Record

Date	Time	Score	Rating	Slope	Note
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX

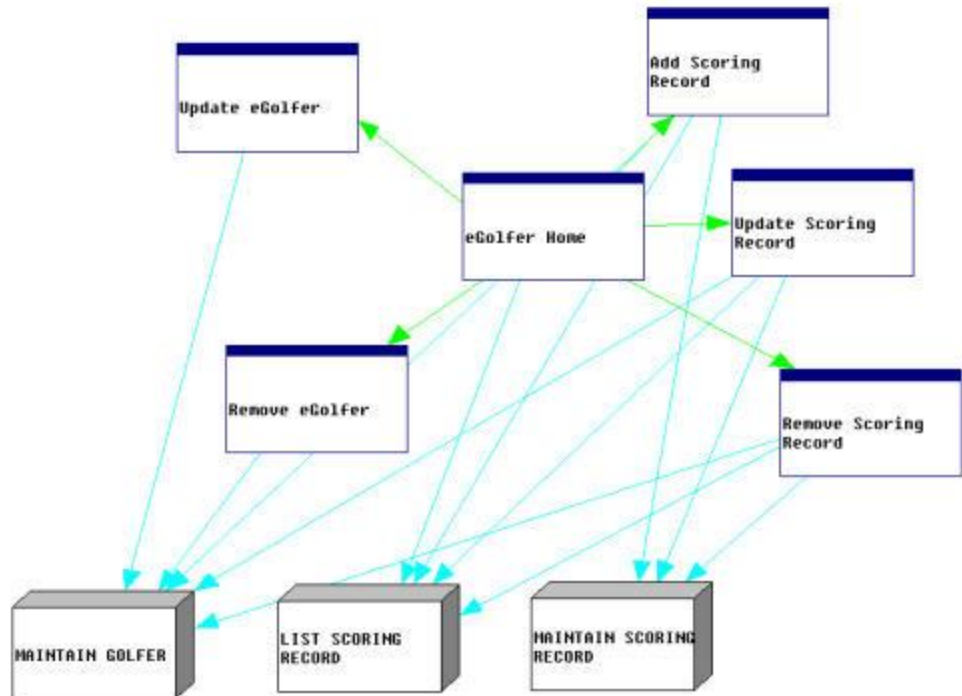
Once directed to the eGolfer Home page, users can update their profile information or remove themselves entirely from our database. To do so, we will call the Maintain Golfer server procedure.

In addition, golfers will be able to add, update, and remove scoring records. Performing any of these actions will require:

- Adding, updating or deleting scoring records from the database, functions that are performed by the Maintain Scoring Record server procedure.
- Re-populating the Web page with the updated list of scoring records, a function that is performed by the List Scoring Record server procedure.

- Recalculating the golfer's handicap index based on the new and updated scoring records, a function that is performed by the Maintain Golfer server procedure.

The previous basic system flow is depicted in the following Navigation Diagram:



The Update eGolfer Information window will appear as shown:

update eGolfer

Place Your Ad Here

Update eGolfer Information

Your User ID is permanent and cannot be updated.

User ID:

To ensure the security of your personal information, please designate a password of up to 8 alphanumeric characters. Your User ID and Password are required to login to the system.

Password:

Confirm Password:

eGolf Services maintains minimal information about our members, only so that we can personalize your web experience. All fields are required.

First Name:

Last Name:

Email Address:

Please review your information before updating. Thank you!

update

cancel

The Remove eGolfer window will appear as shown:

remove eGolfer

Place Your Ad Here

Remove eGolfer

Are you sure you want to be completely removed from our database?

User ID: xxxxxxxx

First Name: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Last Name: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Email Address: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

remove cancel

The Add Scoring Record page will appear as shown:

Add Scoring Record

Place Your Ad Here

Add Scoring Record

Please enter the following scoring information about your game of golf.

Date Played: (MMDDYY)

Time Played: (HHMM)

Adjusted Gross Score:

Course Rating:

Course Slope Rating:

Note:

Please review your information before continuing. Thank you!

The Update Scoring Record page will appear as shown:

Update Scoring Record

Place Your Ad Here

Update Scoring Record

Please note that the Date and Time Played cannot be updated.

Date Played:

MM/DD/YY

Time Played:

HH:MM

Please update the following scoring information about your game of golf.

Adjusted Gross Score:

ZZZ

Course Rating:

ZZZ

Course Slope Rating:

ZZZ

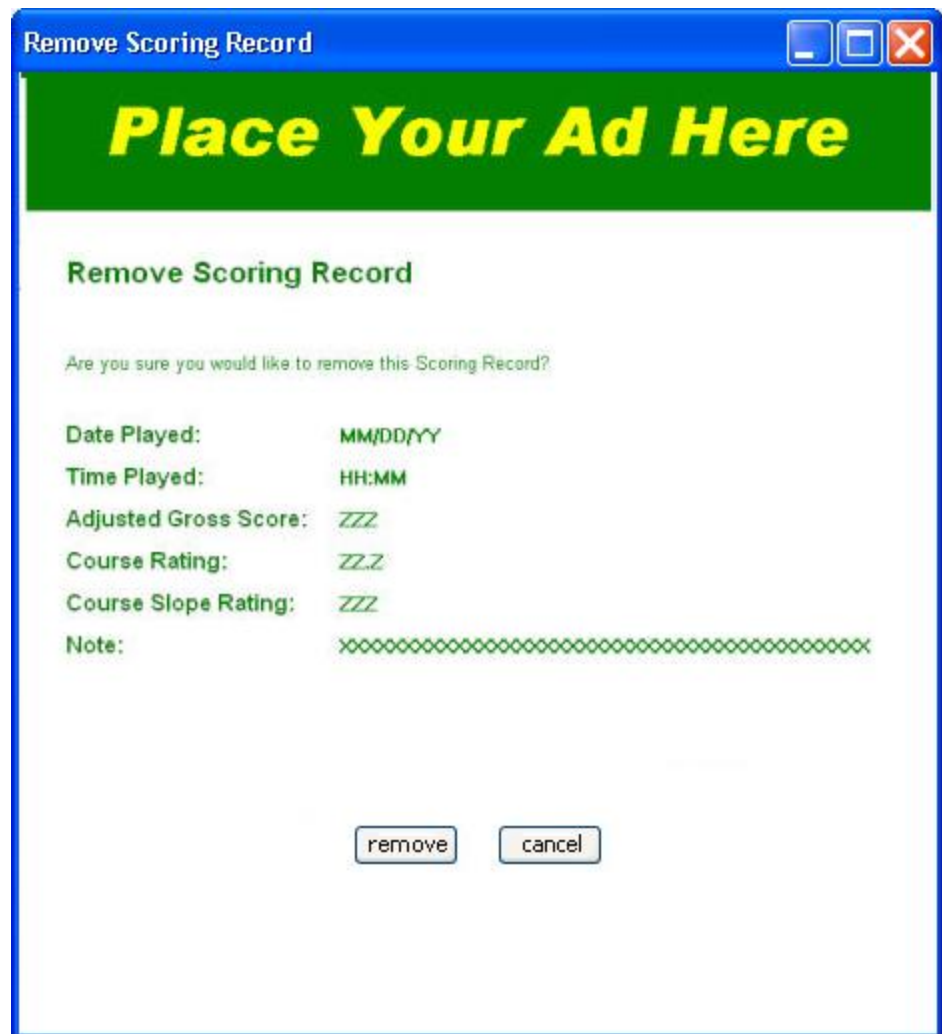
Note:

XX

Please review your information before continuing. Thank you!

updatecancel

The Remove Scoring Record window will appear as shown:



- Define the two client procedures:
 - eGolf Services Home
 - eGolfer Home


- Design the Web pages and add the logic for the two client procedures:
 - eGolf Services Home
 - eGolfer Registration
 - eGolfer Login
 - eGolfer Home
 - Update eGolfer
 - Remove eGolfer
 - Add Scoring Record
 - Update Scoring Record
 - Remove Scoring Record



eGolf Services Home

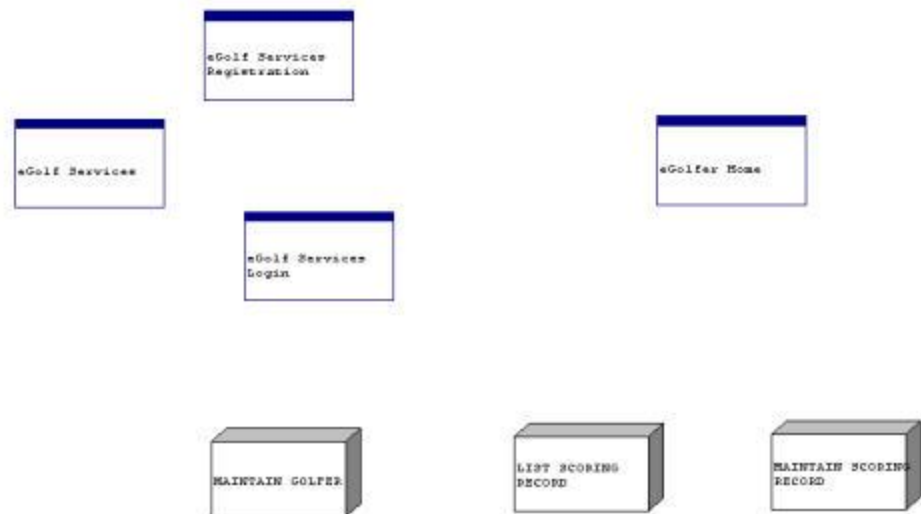
The following sections describe eGolf services home.

Window Navigation Diagram

The following steps explain the window navigation diagram.

1. To add a new client procedure to our Navigation Diagram, we must first open the **Window Navigation Diagram**. If it is closed, open the Window Navigation Diagram. Refer back to a prior section if you need help.
2. On the Window Navigation Diagram tool palette, there is an icon for adding a window, and another for adding a dialog. Regardless of whether you are adding a new client procedure with a primary window or a new client procedure with a primary dialog, you always use the Add Window icon for adding a new procedure. On the subsequent properties panel, you choose whether it is to be a primary window or a primary dialog. The Add Dialog icon is used only for adding secondary dialogs to existing procedures.
3. From the tool palette, select the **Add Window**  icon and, with the cursor, select any location in the Navigation Diagram that is above and to the left of the Maintain Golfer server procedure. If the cursor resembles the international no symbol, it is because you have a server highlighted. You can ignore that symbol for now.
4. In the Window / Dialog Properties panel, under Type select the **Primary Dialog** **Box** radio button. Under Style, un-check System Menu. Then, in the Title: entry field, enter **eGolf Services** (which is the name that will appear on the Web Browser title bar), and in the Name: entry field enter **egolf services home**. Select the **OK** push button.

5. To add the Registration secondary dialog, from the tool palette select the **Add Dialog**  icon and with the cursor (which now resembles the international no symbol), click directly on the **eGolf Services** primary dialog you just added. In the Window/Dialog Properties panel, notice that the Type is already set to Dialog and cannot be changed. Change the Initial Position to **System Placed**, remove the check from **System Menu**, for the Title: enter **eGolf Services Registration**, and for the Name: enter **egolfer registration**. Select the **OK** push button.
6. To add the Login secondary dialog, do exactly the same thing we did earlier in step number five for the Registration secondary dialog, but for the Title:, enter **eGolf Services Login**, and for the Name:, enter **egolfer login**.
7. To add the eGolfer Home procedure step and primary dialog, from the main menu select the **Add Window**  icon and select a position on the Navigation Diagram somewhere above the Maintain Scoring Record server procedure. In the Window / Dialog Properties panel, change the Type to **Primary Dialog Box**, under Style un-check the **System Menu**, in the Title: enter **eGolfer Home**, and for the Name: enter **egolfer home**. Select the **OK** push button.
8. Arrange the dialogs roughly as shown by click-and-dragging them to the desired location:



While it is not imperative that you use the names suggested earlier, using the recommended names will help avoid confusion later in the Tutorial, as statements in the procedure step action diagrams will reference these dialog names.

Note: If you need to change the names or properties of the dialogs, select the dialog. Then from the Main Menu select **Detail**, and then select **Properties**.


Your dialogs should have a solid blue bar across the top as shown in the previous diagram. If they do not, you can have added them as primary windows instead of as primary dialogs. If you need to change them, go to their Properties panel. Alternatively, you can select them, delete them, and add them back again as primary dialogs.

Design the User Interface

When designing the User Interface (UI), whether it is for a Blockmode, Windows, or Web application, we want to apply a consistent standard. Standards can be defined at the Business System level, and then overridden on a case-by-case basis as necessary.

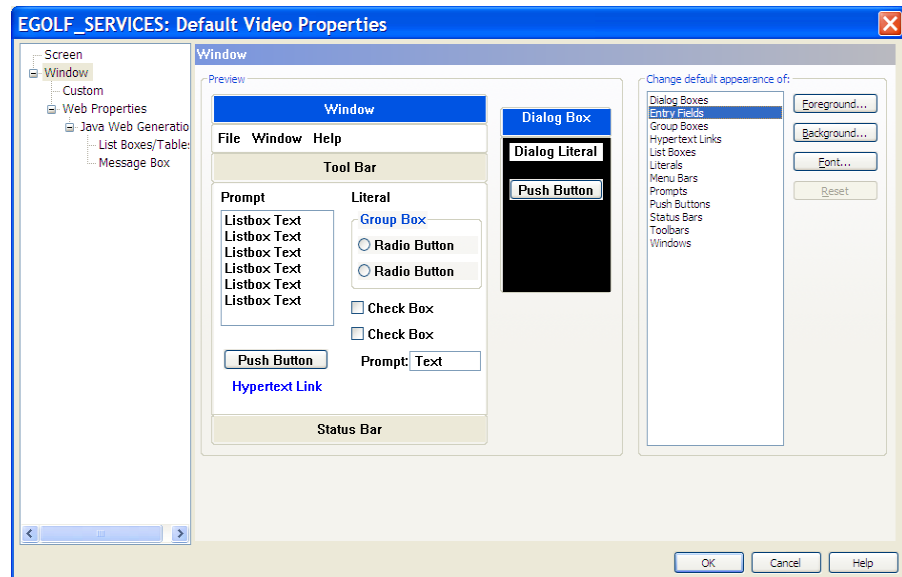
Follow these steps:

1. To set up the UI standards for our system, we need to update the Business System Defaults. From the Tree View, double-click **Business System Defaults** under the Design folder. Then, in the Business System Defaults panel, select Business System **EGOLF SERVICES** and **Diagram** and **Open** from the Main Menu. This will place two asterisks next to the business system name indicating that the default business system is EGOLF SERVICES. Next, from the Toolbar, select the **Window Video**

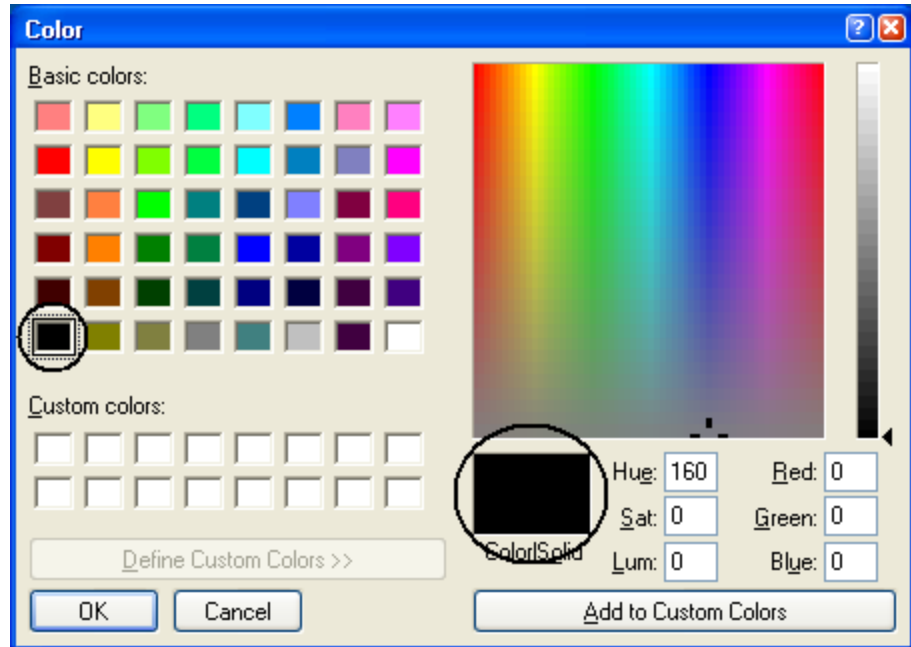
 icon. In the treeview on the EGOLF_SERVICES: Default Video Properties dialog, select **Window**. In the treeview on the EGOLF_SERVICES: Default Video Properties dialog, select **Window**.

2. We want to set default values for the foreground and background colors, and for the fonts for Entry Fields, Prompts, Literals, Push Buttons, List Boxes, and Group Boxes. In some cases, selecting these objects from the list on the far right and changing their properties will result in these changes being reflected in the sample window and dialog to the left.

From the Change default appearance of: list box on the far right, select **Entry Fields** as shown in the following example:



3. Now, select the **Foreground...** push button. Verify that the foreground color for the entry field is black by examining the Color/Solid field shown. If it is not set to black, select the black color box in the Basic colors field as indicated in the following example and then select **OK**.



4. Select the **Background...** push button and verify that the background is white. If it is not, correct it using the method described in step number three, then click **OK**.
5. Select the **Font...** push button. Verify that the Font: is System, the Font Style: is Bold, and the Size: is 10. Select the **OK** push button.
6. While the System font is normally a good choice to have the same look and feel as other applications within your environment, it is somewhat limited. Therefore, for everything else we are going to use Arial. If you do not have Arial on your system, choose another font of your liking that allows for Regular, Bold, and Bold Italic as well as the sizes 8, 10, 12, and 16. Adjust the following video properties in the same manner as we did earlier for the Entry Fields.
 - Group Boxes:
 - Foreground—**Dark Green** (3rd column, 4th row)
 - Background—**White**
 - Font—**Arial/Italic/8**

- List Boxes:
 - Foreground—**Dark Green**
 - Background—**White**
 - Font—**Arial/Bold/12**
- Literals:
 - Foreground—**Dark Green**
 - Background—**White**
 - Font—**Arial/Regular/10**
- Prompts:
 - Foreground—**Dark Green**
 - Background—**White**
 - Font—**Arial/Bold/12**
- Push buttons:
 - Foreground—**White**
 - Background—**Dark Green**
 - Font—**Arial/Bold/10**

Notice that the colors for the push buttons are reversed from the other items.


- Dialog Boxes:
 - Foreground—Does not need to be set
 - Background—**White**
 - Font—Does not need to be set

7. When you are finished, select the **OK** push button.


Add Exit States

While we are here at the Business System Defaults diagram, we can as well add a couple of Exit State Values that we will be using later.

Follow these steps:

1. From the Toolbar select the **Exit States...**  icon.
2. Select **sys EGOLF SERVICES** from the ExitStates panel and then select the **Add...** push button. Add a new exit state with the following properties:
 - Exit State Name—**xfr to egolfer home**
 - Termination Action—**Normal**
 - Message Type—**None**
 - Message Text—Leave it blank.
3. Select **<Global Exit States>** from the Exit States panel and then select the **Add...** push button. Add a new exit state with the following properties:
 - Exit State Name—**password mismatch**
 - Termination Action—**Normal**
 - Message Type—**Error**
 - Message Text—**The passwords entered do not match. Re-enter your passwords.**

Select the Close push button
4. Expand **<Global Exit States>** (if necessary) and select the exit state **RETURN**. Select the **Properties...** push button and change the Message Type to **None**. Select **OK** and then **Close**.

Notice that from the Business System Defaults diagram you can set up other business system-specific things like Commands, Edit Patterns, and Web Properties. Some of these options are available from the Toolbar, others have to be selected from the Menu Bar Detail option.
5. Close the Business System Defaults panel by selecting the lower of the two  in the upper right hand corner.

Design Windows and Web Pages

We are now about ready to start designing our windows/Web pages. Regardless of the fact that we ultimately intend these to be Web pages, we use the window/dialog design facilities within the Toolset. For the sake of discussion, for the remainder of this section we will generically refer to them as windows.

Designing windows generally consists of seven steps:

1. Deciding what data is going to be displayed on the window
2. Creating the information views in the procedure step action diagram necessary to support that data
3. Deciding what GUI controls will be used to represent that data
4. Adding the data GUI controls and related events (if any)
5. Detailing the data-related events
6. Adding non-data GUI controls and related events (if any)
7. Detailing the non data-related events

Any information that you are going to display on a window must have a corresponding export view in the client procedure step action diagram. If you expect to bring that information back into the procedure step action diagram, then you must have a corresponding import view as well. These import and export views are mapped to the corresponding entry field on the window. However, the number of situations in which information needs to be displayed without having to be brought back into the procedure step action diagram for further processing is so low that, in most cases, the import views can look exactly like the export views.

When placing these data fields on the window, you need to decide how that information will be presented. For instance, if you were going to put a field on a window representing a person's gender, that field can be just an entry field in which you type either male or female. Yet, this information can just as easily be presented as a radio button, a check box, or even a drop-down list. The Toolset will default to a particular GUI control based on the number of permitted values for an attribute, or the size of the group view. These defaulted GUI controls can then be changed if required.

For many of these GUI controls, you can define events. For example, you can define a click event for a check box. If someone clicks on that check box, the event handler associated with it will execute. Event handlers are like little paragraphs of code that are executed. Event handlers appear below the main logic of the client procedure step action diagram.

In addition to data-related GUI controls, you can have non data-related GUI controls. Examples of these include push buttons, menus, tool bars, group boxes, and pictures. Some of these can have events assigned to them as well, such as the push button, which often has a click event associated with it.

Finally, the window or dialog as a whole can have events assigned to it as well. For example, if you want a dialog to open with some information already pre-populated, then you can assign an open event to the dialog to get that information.

eGolf Services Home

The first window we are going to design is the eGolf Services home page. See the window examples shown previously to familiarize you with how the window will look.

In addition, referring to the seven steps for designing a UI, the first two steps involve dealing with the data-related GUI controls. This particular window does not display any data. However, it does have non-data controls (the two push buttons), some literals, and three images (the title banner across the top, a banner down the left side, and a golfer image in the top left corner). Therefore, we will place them on the window and detail any events associated with them.

To use images in a model, they must be included in part of the model directory. Before we start designing the window, let us make sure the images are accessible.

Follow these steps:

1. Locate the component model directory `djscap1a.ief` and copy the contents of the `html\image` and `bitmap` folders to your model's (`egolf.ief`) `html\image` and `bitmap` folders. These images and bitmaps were not really needed by the component model, but the component model made for a nice delivery mechanism.
2. Open the window designer by double-clicking on the **eGolf Services** dialog in the navigation diagram. A representation of the window will appear. Roughly, resize the window design by moving the cursor along one edge of the window or another until you get the double-sided arrow. Then click-and-drag the edge to the desired size. For now, be sure to give yourself plenty of room on the window in which to work. Later, we will resize it back to a better size when we are finished laying out the rest of the controls.

Add the Push Buttons

The first things we are going to add to the window are the two non-data GUI control objects, the push buttons.

Follow these steps:

1. From the Menu Bar select **Add**, and then select **Push Button....**
2. For the Push Button Properties, enter `sign up!` in the Text: entry field. Notice that the default Button Action is to execute the procedure step. However, for Client Procedure Steps, most logic is executed with event handlers.

3. Select the **Events...** push button and examine the Event Processing dialog. At the top, it displays the control object, the sign up! push button. The Defined Events panel shows the events that have been assigned to this object. Currently there are none. In the Event, Details section, you can create new events for this object. In the Event Type: drop-down list, you can select the type of event you want to create for this object. Only click events are allowed for push buttons. In the Action Name: entry field, you can type in the name for the click event that you want to create.

We are going to let the Toolset create the event and generate a name for the event as well. To do so, select the **Add** push button. An event is created and added to the Defined Events panel.


The naming convention for the event used by the Toolset is the window name (EGOLF SERVICES), an abbreviation for the object type, (PB, for push button), the name of the object, (SIGN UP), and the type of event, (CLICK). In this case, this gives us the name EGOLF SERVICES PB SIGN UP CLICK.

4. Select the **Close** push button.
5. The Button Action on the Push Button Properties panel now indicates that the push button Executes Click Event. Select the **OK** push button. With the cursor (which now resembles a rectangle), select a position on the window close to where we ultimately want the push button. Do not worry if it is not in the exact position, as we can reposition it later.
6. If you have multiple adds turned on, we can continue and add the next push button. If not, from the Menu Bar, select **Add**, and then select **Push Button....** Enter **login** for the Text, check the **Is the Default Push Button** check box, and then select the **Events...** push button.
7. On the Event Processing dialog, select the **Add** push button to have the Toolset automatically create an event for us, and then select the **Close** push button.
8. On the Push Button Properties dialog select the **OK** push button, and then place this push button below the first one. Do not worry so much about getting it exactly underneath the first one; we will position them more accurately later. If you have multiple adds turned on, select **Cancel** on the Push Button Properties panel after positioning the login push button.

Add GUI Statements to Action Diagram

We can now add the action diagram statements to these two events. The only thing we want either of these two actions to do is to open the appropriate secondary dialog.

Follow these steps:

1. If necessary, move the window design out of the way by click-and-dragging it from the title bar, and from the Toolbar select the **Action Diagram**  icon to open the procedure step action diagram. If necessary, scroll up and notice that the two event handlers have been added below the main procedure step logic.
2. Select **EVENT ACTION egolf servic pb sign up click**. Then from the Menu Bar, select **Edit**, then **Add GUI Statement**, and then select **Open....** Select the Dialog **EGOLFER REGISTRATION**, and then select the **OK** push button to add the statement.
3. Select **EVENT ACTION egolf servic pb login click**. Then from the Menu Bar, select **Edit**, then **Add GUI Statement**, and then select **Open....** Select the Dialog **EGOLFER LOGIN**, and then select the **OK** push button to add the statement.
4. Close the procedure step action diagram.


Add Literals

Follow these steps:

1. From the Menu Bar select **Add**, and then select **Literal....** For the Literal Properties enter **New Golfer:**, then uncheck the **Center Vertically** and **Horizontally** check boxes. Select the **OK** push button and position the literal to the left of the sign up! push button.
2. If you have multiple adds turned on, you can continue to do the same for each of the literals in the following list. Select the **Cancel** push button when you are through. If you do not have multiple adds turned on, add the literals from the Menu Bar as described in step number one. Lay each one out roughly as indicated in the window example shown earlier.

- Existing Golfer:
- With the click of a mouse you can...
- Enter your most recent scores
- Determine your golf handicap index
- Update your profile

Note: If you make a mistake, just double-click the literal and correct it. If you need to move them around a little, you can right-click-and-drag them with the mouse.

3. While the literals inherited the properties stated in our Business System Defaults, we want to override the fonts for the New Golfer and Existing Golfer literals. Select the **New Golfer:** literal. From the Toolbar, select the **Video Properties ...**  icon. Uncheck the Automatic checkbox for Font and press the **Select...** push button to bring up the Font dialog. Change the Font properties to **Arial/Bold/12** and select the **OK** push button. Resize the literal box the way we did with the window so that the entire literal is visible.
4. Perform the same steps for Existing Golfer: as you did for New Golfer:.

Add Images

Now we want to add the images. Bitmaps are used for window design, while GIFs and JPGs are used for Web pages. Generally, you need both bitmaps and GIF/JPG for Web design. Since our Web generation is based on our window design, you need a bitmap for the window design and a corresponding GIF/JPG with the same name for the Web generation.

Follow these steps:


1. From the Menu Bar select **Add**, and then select **Picture....** In the Picture Properties panel, select the bitmap **egolf services** from the Bitmap: drop-down list, in the Name: entry field enter **picture1**, and for the HTML Extension: enter **gif**. Select the **OK** push button and position the picture at the top left of the window. Resize the right side of the window to fit with the heading banner.
2. Repeat the previous steps to add the **egolfside** picture to the left side of the window below the eGolf Services heading banner. Name it **picture2** with the HTML Extension: of **gif**. Overlap them slightly where they meet.
3. Add the **duffer** picture to the top left corner. Name it **picture3** with the HTML Extension: of **gif**.

Position the Objects

The Toolset provides the capability to easily position many objects on a window relative to one other object on the window. The first object is like a pivot point. You manually position one object on the window where you want it, and then automatically position the other objects relative to the first object.

Follow these steps:

1. First, we want to left justify **New Golfer:** and Existing Golfer: relative to each other. Manually position New Golfer: on the window where you want it by right-clicking it and dragging it to the desired position. With the focus still on New Golfer:, press and hold the **Ctrl** key and select **Existing Golfer:**.
2. From the Menu Bar, select **Edit**, and then **Position....** In the Field Positioning dialog, select the **Align Vertically Left** checkbox. Then select the **Separate Vertically** checkbox, enter the Distance value **10**, and select the **OK** push button.

3. We want to align the push buttons relative to the prompts, but first we want to give the push buttons a more svelte appearance. Select the **sign up!** push button and resize it making it slightly less tall, but not so much as to impact the lettering. Now we want to position it relative to the **New Golfer:** literal. Manually position it to where it is the correct distance to the right of **New Golfer:** This will place it to the right of **Existing Golfer:** as well.
4. Select **New Golfer:** While holding the **Ctrl** key on your keyboard, select the **sign up!** push button. From the Menu Bar select **Edit**, and then select **Position....** In the Field Positioning dialog, select the **Align Horizontally Top** check box, and then select the **OK** push button.
5. Now, we want to position the login push button relative to the location and height of the sign up! push button. Select the **sign up!** push button. With the **Ctrl** key on your keyboard held down, select the **login** push button. From the Menu Bar select **Edit**, and then select **Position....** On the Field Positioning panel, select the **Equal Height** and **Equal Width** check boxes. Select **Align Vertically Left**. Then select the **Separate Vertically** checkbox, enter the Distance value of **10**, and select the **OK** push button.
6. To align the three bulleted literals, select – **Enter your most recent scores.** Then, using the **Ctrl** key on your keyboard, select the other two literals. From the Menu Bar, select **Edit**, and then select **Position....** Select the **Align Vertically Left** checkbox and the **Separate Vertically** checkbox, then enter a distance **4**. While they all still have focus, right-click-and-drag them as a group to where you want them.
7. Finally, right-click-and-drag the **With the click of a mouse...** literal to a suitable location. Your window/Web page should now look very similar to the example shown earlier.
8. To close the window designer, from the Toolbar select the **Close Window**  icon.
9. Save your model.

eGolfer Registration

See the window examples shown earlier and familiarize yourself with the eGolf Services Registration Web page. On this page, we have a couple of images, some literals, some push buttons, and some entry fields. The entry fields are data-related controls and normally we place them first. For each data-related object, we need to determine what information views are required to support that object, and what GUI data-related control will be used to represent each piece of data.

All the information about a golfer (other than their handicap index) is used on this page. However, we are going to pass all of this information, plus the handicap index, to the golfer's home page. Essentially, we need an export view with all of the golfer's information. In addition, we have two views of a golfer's password. One is for their confirmation. This means that we actually need two export views of a golfer, one with all of their information, and another with just the password. Once we create the necessary export views, we will just copy them to the imports. We will then place (map) these views to data-related controls on the window.

In addition to deciding which views are required, we would decide which type of data-related control would be used to represent the data. Examples of data-related controls are entry fields, radio buttons, check boxes, and drop-down lists. We are going to be using entry fields.

Add Views

Follow these steps:

1. In the Navigation Diagram, select the **eGolf Services Registration** dialog. From the Toolbar, select the **Action Diagram** icon. Notice that it takes you to the same EGOLF SERVICES HOME procedure step action diagram that we were at before. This is because all of the secondary dialogs share the same diagram as the primary window or primary dialog.
2. In the procedure step action diagram, select EXPORTS. Then, from the Menu Bar, select Edit, then Add View..., and then select Add Entity View.... In the Name entry field, enter export, then select GOLFER, then select all of its attributes by selecting **GOLFER** again, and then select the **OK** push button.
3. Add another export view of golfer with just the password attribute. Name the view **export confirmation**.
4. Expand all the views.
5. Copy the export views to the import views. Name them **import** and **import confirmation**.
6. If a golfing foursome all registered at the same time using the same browser session, each time the registration page came up it would probably contain the previous golfer's information. To prevent this from happening, we can create a local view of a golfer that serves no purpose but to initialize the export views before opening the registration or login pages. Copy the **export golfer** to the **LOCALS:** view, naming the local view **local**.
7. Now, we want to initialize the export golfer with the blank golfer. Select **EVENT ACTION egolf_servic_pb_sign_up_click**. From the Menu Bar, select **Edit, Add Statement...**, select **Move...**, select **local golfer**, select **export golfer**, and then select the **Add** push button.
8. Add another statement to move **local golfer** to **export confirmation golfer**.

9. Copy the two move statements to the login event action. Your action diagram so far should look like the following example:

```

EGOLF_SERVICES_HOME
IMPORTS:
  Entity View import_confirmation golfer (optional,transient,import only)
    password (optional)
  Entity View import golfer (optional,transient,import only)
    password (optional)
    first_name (optional)
    last_name (optional)
    email_address (optional)
    handicap_index (optional)
    userid (optional)
EXPORTS:
  Entity View export_confirmation golfer (transient,export only)
    password
  Entity View export golfer (transient,export only)
    password
    first_name
    last_name
    email_address
    handicap_index
    userid
LOCALS:
  Entity View local golfer
    password
    first_name
    last_name
    email_address
    handicap_index
    userid
ENTITY ACTIONS:

EVENT ACTION egolf_servi_pb_sign_up_click
MOVE local golfer TO export golfer
MOVE local golfer TO export_confirmation golfer
OPEN Dialog Box EGOLFER_REGISTRATION


EVENT ACTION egolf_servi_pb_login_click
MOVE local golfer TO export golfer
MOVE local golfer TO export_confirmation golfer
OPEN Dialog Box EGOLFER_LOGIN

```

10. **Save** your model.

Lay Out the Data Control Objects

Follow these steps:

1. Close the action diagram. In the Navigation Diagram, double-click the **eGolf Services Registration** dialog. We would like all of our windows to be roughly the same size. To assist with resizing this window, open the eGolf Services window and place one window over the other such that you can resize the eGolf Services Registration window to be roughly the same size as the eGolf Services window. Once they are about the same size, close the eGolf Services window by selecting it first and then selecting the **Close Window**  icon from the Toolbar.

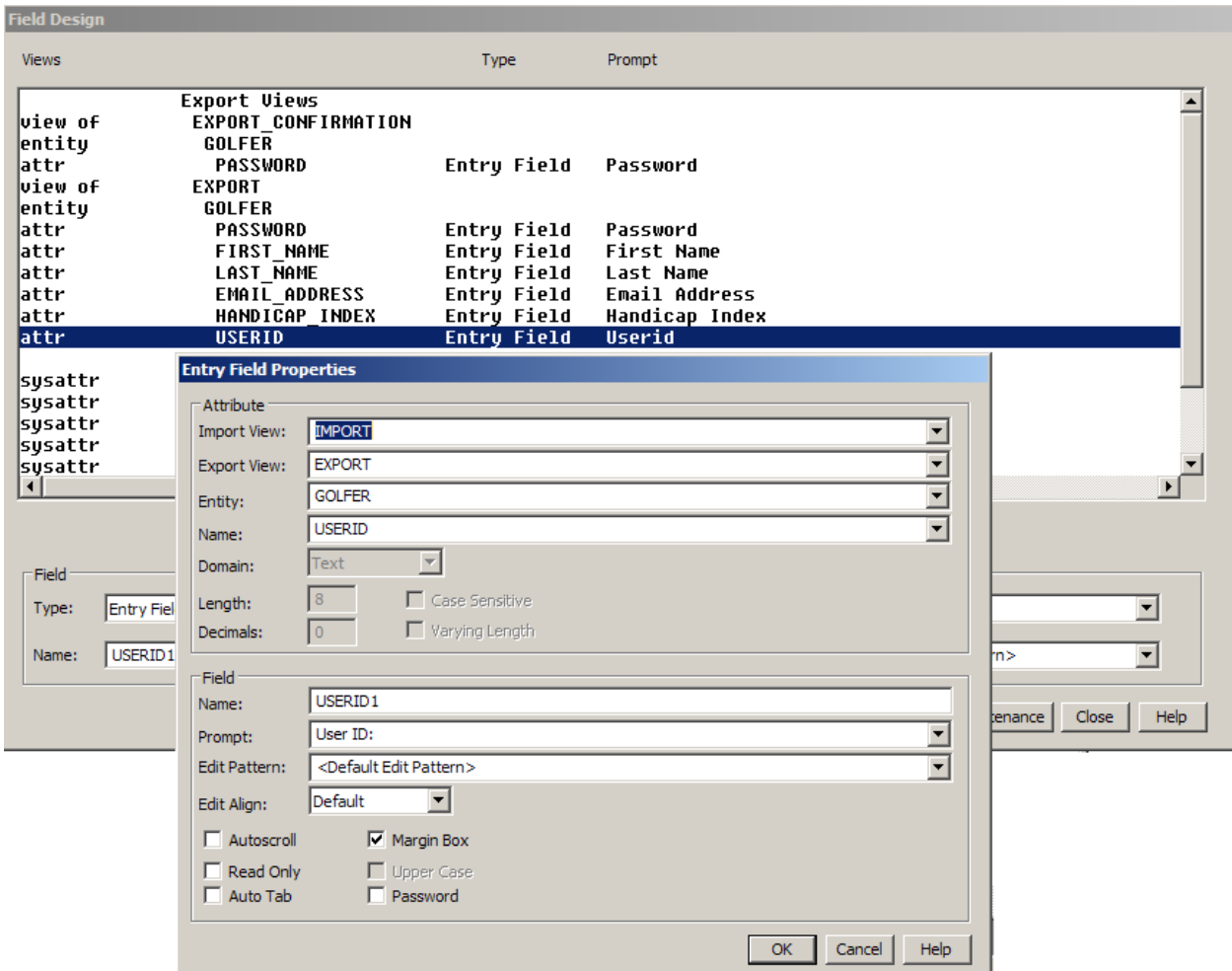
2. With the eGolf Services Registration window design open, from the Menu Bar select **Edit**, and then select **Field Design....**

In the Field Design dialog, you are presented with a listing of the procedure step action diagram's export views, along with some special fields which include things like the current date, current time, error message field, and current timestamp. Next to the export view, it tells you the type of field as well as the prompt for the field. These are the defaulted values for these export views. However, we can change each of these defaulted values by first selecting the export view from the list, and then modifying its properties in the Field group box at the bottom of the panel. For example, all of these have defaulted to the type of entry field. While it does not make particular sense in this example, we might want one of the entry fields to be a check box. So we can select it from the list, and then in the Type: drop-down list at the bottom of the panel, change it to a check box control.

While we want all of these to be entry fields, according to our specifications, each of the prompts are suffixed with a colon. Prompts are the labels that go with each field. Other changes include changing the prompt Userid to User ID: and the confirmation password from Password to Confirmation Password: In addition, we want to verify the view mapping.

3. View mapping is a fairly important concept that should not be confused with view matching. View matching is used to define which information view in one action diagram is passed to which information view in another action diagram. View mapping is similar, but it is between an information view in an action diagram and a field on a window, Web page, or screen. Windows, Web pages, and screens are actually separate pieces of generated code. View mapping defines which field on a window will be used to display the information placed in the action diagram's export view, and which import view in the action diagram will receive the data that was entered into a field on the window. In the Field Definition dialog, select the **EXPORT GOLFER USERID**. In the Field group box in the lower half of the panel, you should see that the Type: is an Entry Field, the Name: is USERID1, the Prompt: is Userid, and it is using the <Default Edit Pattern>.

To check the view mapping, select the **Properties...** push button. Then, in the subsequent Entry Field Properties dialog, in the Import View: drop-down selection list select **IMPORT**, and in the Prompt: entry field, enter **User ID:**.



What this panel is showing is that on the window we are going to have an entry field. This entry field will be used for two purposes. One purpose is to display whatever information the action diagram logic places into the export golfer userid. The other purpose is to take whatever information is entered into that entry field and place it into the procedure step's import golfer userid.

The other thing we changed is the prompt. Prompts are the labels associated with fields. When we actually place this field on the window, we will first be presented with the prompt to place on the window, and after placing the prompt, we will then be presented with the actual field to place on the window.

Notice also the following check boxes.

Margin Box check box

Places a box around the entry field

Read Only check box

Prevents anyone from entering any information in the field

Password check box

Prevents the characters being typed into the field from being displayed

4. To finish placing this field on the window, on the Entry Field Properties panel select the **OK** push button, and on the Field Design dialog select the **Place** push button. The cursor will change to a rectangle indicating the prompt. Move the rectangle (prompt) to the location for the User ID prompt that was indicated in the window examples shown earlier and select that location. Immediately after placing the prompt, you will be presented with another rectangle that indicates the field. Select a location some distance to the right of the prompt for the field placement. If, after placing the field, you still see some leftover screen refresh artifacts (such as a rectangle in the center of the window), you can safely ignore them.

Note: If you make a mistake, after placing the field onto the window simply double-click it to bring its Properties panel back up. To move it, right-click-and-drag it.

5. To place the password, from the Menu Bar select **Edit**, then select **Field Design....** Select **EXPORT GOLFER PASSWORD** and then select the **Properties...** push button. To complete the view mapping, select **IMPORT** from the **Import View:** drop-down list, place a colon (:) after the Prompt: **Password**, place a checkmark in the **Password** check box, select the **OK** push button, select the **Place** push button, and place the prompt followed by the field onto the window. Resize the prompt if necessary.
6. To place the confirmation password, from the Menu Bar select **Edit**, then select **Field Design....** Select **EXPORT CONFIRMATION GOLFER PASSWORD** and then select the **Properties...** push button. To complete the view mapping, select **IMPORT CONFIRMATION** from the **Import View:** drop down list, change the Prompt: to **Confirm Password:**, place a checkmark in the **Password** check box, select the **OK** push button, select the **Place** push button, and place the prompt followed by the field onto the window.

7. You can place multiple fields consecutively. Using Field Design, define the view mapping and prompt properties for the **EXPORT GOLFER FIRST NAME, LAST NAME,** and **EMAIL ADDRESS** without actually placing them.

Note: To define the properties for the next field, you will have to unselect the first field to enable the **Properties...** push button.

8. After defining all of their properties, re-select all three fields and then select the **Place** push button. You will be presented with the prompt for the first field in the selection list, followed by its actual field. This will be repeated for all three fields. It is useful to first order the fields in the Field Design dialog by using the **Move** push button. First, select the attribute you want to move, and then select the attribute you want to move the initially selected attribute below.

Positioning

By now, you can be thinking that your window does not look very great. Let us stop for a moment and go through a few positioning steps to make it look a little bit better.

The first thing you can have noticed when placing the email entry field is that it extended off the right side of the window. We can make it look better by making it the same size as the name fields.

Follow these steps:

1. Select the **Last Name entry field** (not the prompt). Hold down the **Ctrl key** on your keyboard while selecting the **Email Address** entry field. From the Menu Bar select **Edit**, select **Position...**, select the **Equal Width** check box, and then select the **OK** push button.
2. Reviewing the window design, there are two determining factors affecting our overall layout. **Confirm Password** is the widest prompt, and we want all of the fields to align with its right side. In addition, the combined Email Address prompt and field is the widest total line, which we would like centered.

Select the prompt **Confirm Password:** and then, using the **Ctrl key**, select all of the other prompts. From the Menu Bar select **Edit**, select **Position...**, select the **Equal Width** check box, and then select the **OK** push button. If you get a warning that you are going to overlap some fields, select **Yes**.
3. Now, select the **Email Address** prompt and, with the **Ctrl key**, select the **Email Address** field. From the Menu Bar select **Edit**, select **Position...**, select **Align Horizontally Bottom**, select **Separate Horizontally** by a Distance of **5**, select **Move Horizontally Center**, and then select the **OK** push button.
4. Select the **Email Address field** (not the prompt) and with the **Ctrl key** held down select the **Last Name** field and the **First Name field** (again, just the fields, not the prompts). Then, from the Menu Bar, select **Edit**, select **Position...**, select **Align Vertically Left**, select **Separate Vertically** by a Distance of **5**, Select **Arrange prompts** against the fields, and then select the **OK** push button.

5. Next, select the **First Name** field, and with the **Ctrl key** held down select the **Confirm Password** field, and the **User ID** field. From the Menu Bar, select **Edit**, select **Position...**, select **Align Vertically Left**, select **Arrange prompts against the fields**, and then select the **OK** push button.
6. Finally, select the **Confirm Password** field and with the **Ctrl key** held down select the **Password** field. From the Menu Bar, select **Edit**, select **Position...**, select **Align Vertically Left**, select **Separate Vertically** by a Distance of **5**, select **Arrange prompts against the fields**, and then select the **OK** push button.

Add the GUI Controls

Now we can add the non-data-related GUI controls. We can start with the pictures.

Add the Pictures

Follow these steps:


1. From the Menu Bar select **Add**, then select **Picture....** In the Picture Properties drop-down list select **golfrainback**, in the Name: entry field enter **picture1**, in the HTML Extension: entry field enter **gif**, and then select the **OK** push button. Position the picture at the top of the window.
2. Add another picture on top of the previous picture. In the Picture Properties drop-down list select **golfrain_banner**, in the Name: entry field enter **picture2**, in the HTML Extension: entry field enter **gif**, and then select the **OK** push button. Center this picture on top of the previous picture.

If you have multiple adds turned on, cancel out of the Picture Properties dialog.

Add the Literals

Follow these steps:

1. From the Menu Bar select **Add**, then select **Literal....** Enter eGolfer Information, remove the check boxes for **Center Vertically** and **Center Horizontally**, and then select the **OK** push button. Place the literal as indicated in the examples shown earlier. Then, change the default font by selecting the **Detail and Video Properties...**

from the Menu Bar or by selecting the **Video Properties...**  icon from the Toolbar. Then unselect the **Automatic** checkbox for Font and select the **Select...** pushbutton to bring up the Font dialog and set the font to **Arial/Bold/16**. Resize the literal if necessary.

2. Add the remaining five literals in the same way. Use the default fonts for all of them except the last one, which states, **Review your information before continuing. Thank you!**. Make it **Arial/Bold/10**.
3. Left justify vertically all of the literals except the last one, which we want to **Move Horizontally Center**. You may have to slightly re-position some of your fields and literals.


Note: In addition to using the Ctrl key to select multiple objects, you can use the mouse to lasso several objects by click-and-dragging the lasso around them. Objects can then be moved as a group by right-click-and-dragging them to the desired location or by using the positioning feature.

Add the Push buttons

We will now add the two push buttons and their associated events and logic.

Follow these steps:

1. From the Menu Bar select **Add**, then select **Push Button....** In the Push Button Properties Text: entry field enter **<Back>**. For the back Button Action, we want to perform a special action. Select the **Special Action** radio button and in the drop-down list select **Cancel – Close without Execution**. This will close the current dialog and take us back to the originating dialog. Select the **OK** push button on the **Push Button Properties** dialog and place the button as shown in the example.
2. If you have multiple adds turned on, you can continue and add the next push button. Otherwise, from the Menu Bar select **Add**, then select **Push Button....** In the Push Button Properties Text: entry field enter **<Next>**. Check the **Is the Default Push Button** check box.
3. For the next Button Action, we want to execute an event. Select the **Events...** push button. In the Event processing dialog, select the **Add** push button to let the Toolset generate a new click event for the push button. Notice the event name and then select the **Close** push button. Select the **OK** push button on the Push Button Properties dialog and place the button as shown in the example.

4. Resize one of the buttons making it slightly less tall. Then, with focus still on the resized button, click the other push button while holding down the Ctrl key. Then, from the Menu Bar select **Edit**, and then select **Position....** Then select **Equal Height** and **Equal Width**, select **Align Horizontally Bottom**, select **Separate Horizontally** by a Distance: of **20**, select **Move Horizontally Center**, and then select the **OK** push button.
5. For the back push button, we just performed a special action. For the next push button, we want to actually call the Maintain Golfer server to register the golfer. To open the procedure step action diagram, from the Toolbar select the **Action Diagram**  icon. If the icon is not enabled, make sure you have one of the dialogs selected in the **Navigation Diagram**.
6. We only want to register the golfer if the two passwords that were entered match.

This brings up another interesting topic. Where should this type of validation be performed? Should it be in the client procedure step, as we are about to put it? Should it be in the server procedure step? Alternatively, more likely, should it be in the elementary process?

If you want the same business rule to apply regardless of who performs the transaction, or where, when or how the transaction is performed, then you would probably want the validation to be in the elementary process. If the rules can vary for any of the reasons specified earlier, then you will likely want the rules in the client procedure step.

For example, in this particular case we are allowing golfers to register themselves online. When the golfers do that, we do not want them to enter their password incorrectly, so we make them enter it twice. But we can also have a batch procedure in which we get feeds from local golf clubs, pro shops, and sporting goods stores where we set up prospects with some common password, like their last name, or the store name from which we got the feed, and then email them and invite them to login, change their password, and enter their scores. We can set the password differently depending on how we set them up. In such a case, it would make more sense to place the validation rules in the different procedure steps, online versus batch. We are going to put the check here in the client procedure.

In the procedure step action diagram, from the Tool Palette select the **If** push button and with the cursor (which now resembles a crosshair), select the event handler **EVENT ACTION egolfer_reg_pb_next_click**. Then in the Add Statement panel, select the expression **attribute view**, select the **entity view import golfer**, select the **attribute password**, select the operator **IS EQUAL TO**, select the expression **character view**, select the **entity view import confirmation golfer**, and then select the **Add** push button.

7. If the golfer passes the test, then we want to set the system attribute Command to register and then call the server procedure step. With the If statement still highlighted, from the Menu Bar select **Edit**, then select **Add Statement....** Then select **Command is...**, select the expression **command value**, select the command **REGISTER**, select the **OK** push button, and then select the **Add** push button.

Add Procedure Step Use Statement

Now we want to add the procedure step use statement, matching the import views of the server with the import views of the client, and the export views of the server with the export views of the client.

Follow these steps:

1. With the Command is statement highlighted, from the Menu Bar select **Edit**, select **Add Statement...**, select **Procedure Step Use...**, select the non-display procedure step **maintain golfer**, and then select the **Add** push button.
2. In the MAINTAIN GOLFER Import View Matching panel select **IMPORT GOLFER**, then from the possible supplying views in the client EGOLF SERVICES HOME select **IMPORT GOLFER**, select the **Match** push button, and then select the **Close** push button.
3. Next, in the MAINTAIN GOLFER Export View Matching panel, select **EXPORT GOLFER**, and in the Possible Receiving Views for EGOLF SERVICES HOME select **EXPORT GOLFER**, then the **Match** push button, and then the **Close** push button.

Set Exit State

If the call to the server was successful, we want to close this window and flow to the eGolfer home page. Setting the special system attribute Exit State can trigger flows. To determine if the call was successful, we can first check the Exit State value to see if it was set to processing ok.

Follow these steps:

1. From the Tool Palette, select the **If** push button, and then select the **USE maintain golfer** statement. In the **Add Statement** panel select the expression exit state, select the relational operator **IS EQUAL TO**, expand the Global Exit States and select **Processing OK**, select the **Select** push button, then the **Add** push button.
2. From the Tool Palette, select the push button **Exit State**, and then select the **IF EXITSTATE IS** statement. Then in the **Exit State Selection panel**, expand **EGOLF SERVICES**, select **XFR TO EGOLFER HOME**, select the **Select** push button, and then select the **Add** push button.
3. With the EXIT STATE IS statement still highlighted, from the Menu Bar select **Edit**, then **Add GUI Statement...**, then select **Close...** Next, select **Dialog Box EGOLFER REGISTRATION**, and then select the **OK** push button.
4. We still need to set an exit state to handle a situation in which the passwords do not match. Select the bottom of the bracket for the **IF EXITSTATE IS** statement, then from the Menu Bar select **Edit**, and then **Add Statement...**, then select **Else**. Then, from the Tool Palette select the **EXIT STATE IS** push button, select the **Else** statement you just added, select the **Exit State value PASSWORD MISMATCH**, select the **Select** push button, and then select the **Add** push button.

5. The completed event handler should look as shown:

```
EVENT ACTION egolfer_reg_pb_next_click
IF import golfer password IS EQUAL TO import_confirmation golfer password
COMMAND IS register
USE maintain_golfer (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
  WHICH EXPORTS: Entity View export golfer FROM Entity View export golfer
  IF EXITSTATE IS EQUAL TO processing_ok
  EXIT STATE IS xfr_to_egolfer_home
  CLOSE Dialog Box EGOLFER_REGISTRATION
  ELSE
  EXIT STATE IS password_nismatch
```

6. Save your model.

Note: When designing a window, you have the opportunity to place fields on a window that have not first been defined in your export views. This can be accomplished in a number of different ways, but ultimately results from changing the view (mapping) properties of the field as you place it on the window. This feature allows you to develop windows using a RAD- or RAP-type methodology, whereby you start relatively early designing your user interface before the underlying action diagrams have been developed.

7. New users often end up adding fields to their windows incorrectly mapped to a known view and end up with additional unknown or extra views in their action diagrams. Take a few minutes to review your import and export views in the client procedure step action diagram. If you have any views in your action diagram's import or export views other than what was shown, then you will need to correct them.

Correcting them involves determining which field on the window design caused the view to be created, re-mapping that field to the correct views in the action diagram, and finally deleting the extraneous views from the action diagram. The extra views cannot be deleted if they are mapped to a field on a window.

If you have any extraneous views, return to the window design, double-click each field on the window, and review its view mapping. Change the field's view mapping to the correct views in the import and exports, and then delete the view from the action diagram. If you are not allowed to delete the view, then it is probably being reused on another field as well.

8. If necessary save your model again.
9. Close the action diagram.

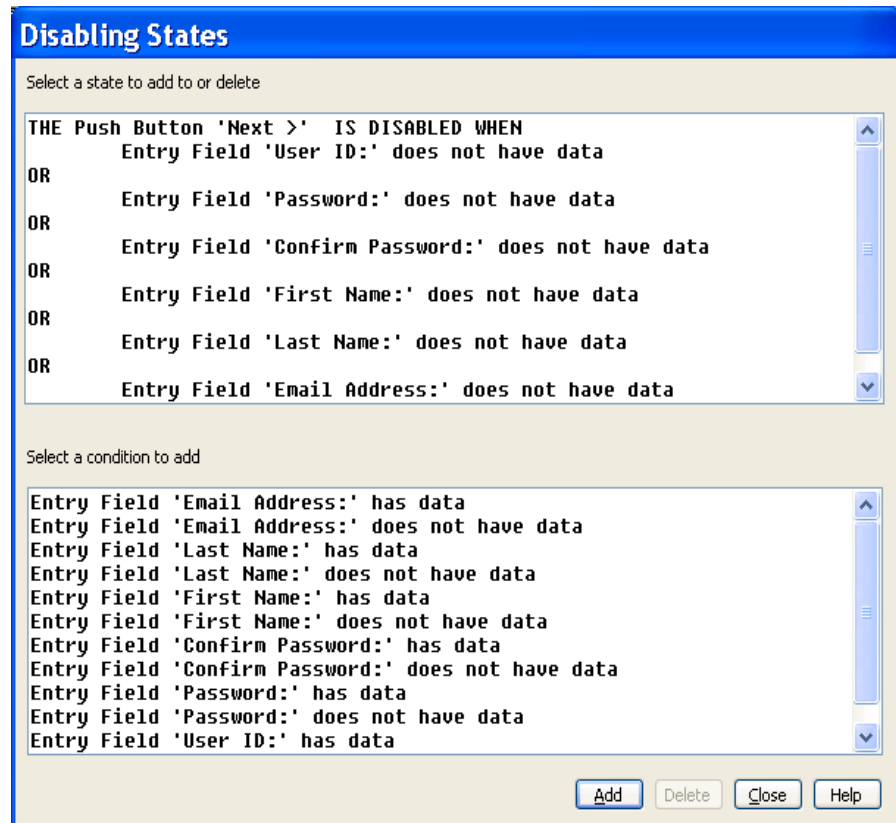
Fine-tune Push Buttons

The last thing we want to do is to disable the next push button until all of the entry fields have something entered into them. In other words, all fields are required, and we do not want the golfers to continue until they have entered something into every field.

Follow these steps:

1. On the window design, select the **Next>** push button. From the Menu Bar select **Detail**, and then select **Disabled By....** In the Disabling States dialog, in the upper panel select the state **THE Push Button 'Next>' IS DISABLED WHEN**, and in the lower panel scroll down if necessary and select the condition **Entry Field 'User ID:'** does not have data. Select the **Add** push button to add the condition to the upper panel.
2. Now select the unmatched **OR** in the upper panel and select the **Entry Field 'Password:'** does not have data from the lower panel and select the **Add** push button.
3. Follow the steps described in step number two to add the rest of the disabling conditions:
 - Confirm Password:
 - First Name:
 - Last Name:
 - Email Address:

4. The completed disabling states should look similar to the following dialog:



5. Select the **Close** push button.
6. Save your model.

eGolfer Login


Try to build the eGolf Services Login window on your own. We will include systematic instructions at the end of the section, but try to do it on your own first. Review all of the following key points before building the window:

1. Resize the window to make it roughly the same size as the other two. You can open one of the other two windows and place it behind the new one to help you size it.
2. Use the same two pictures we used on the Registration window, **golfrainback** and **golfrain_banner**.
3. The heading **eGolfer Login** is a literal. After adding it, change its font to **Arial/Bold/16**.
4. The line across the page is a literal also, made up of approximately **40** underscores.

5. When building additional secondary dialogs, you do not necessarily have to add additional export and import views. Where possible, you reuse the existing action diagrams views. For this dialog, we do not need to add any additional views. The User Id and Password fields are from the EXPORT GOLFER userid and password. When placing them on the dialog, make sure you map them to the IMPORT GOLFER userid and password.
6. Prompts created for a field are saved, such that you can reuse for that same field on any other window or dialog. When defining the field's properties, use the prompt drop-down list to select the prompts that already exist for these two fields that have the colon after them.
7. Since the password field is a password field, check the password check box on the field properties panel. That will cause asterisks to be echoed back to the user when typing in their password.
8. The back push button just performs a Special Action, to Cancel – Close without Execution.
9. The login push button performs a Click event. Let the Toolset generate the event name. The event logic is virtually identical to the event logic for the Registration dialog, except you do not need to check the passwords. You can just copy the logic from the Registration event to the Login event and double-click each element of the action diagram statements that you need to change. These two elements are the command, and the dialog you want to close. Make it the default push button and do not forget the disabling logic.

The following are the systematic instructions:

1. Resize the window:
 - a. In the Navigation Diagram, double-click the **eGolf Services Login** dialog. Then, in the Navigation Diagram, double-click the **eGolf Services Registration** dialog.
 - b. Position the Login dialog over the Registration dialog, and with the cursor, click-and-drag the edges of the Login dialog until they are even with the Registration dialog. Close the Registration dialog and save your model.
2. Add the two pictures:
 - a. From the Menu Bar select **Add**, then select **Picture....** In the Picture Properties drop-down list, select **golfrainback**, in the **Name:** entry field enter **picture1**, in the HTML Extension: entry field enter **gif**, and then select the **OK** push button. Position the picture at the top of the window.
 - b. Add another picture on top of the previous picture. In the Picture Properties drop-down list select **golfrain_banner**, in the **Name:** entry field enter **picture2**, in the HTML Extension: entry field enter **gif**, and then select the **OK** push button. Center this picture on top of the previous picture.

3. Add the heading:
 - a. From the Menu Bar select **Add**, then select **Literal....** Then enter **eGolfer Login**, un-check **Center Vertically** and **Center Horizontally**, and then select the **OK** push button.
 - b. Position the literal roughly as indicated in the example shown earlier. Then, with focus on the literal, from the Toolbar select the **Video Properties...**  icon. Uncheck the **Automatic** checkbox for Font and press the **Select...** pushbutton to bring up the Font dialog. Change the Font properties to **Arial/Bold/16** and select the **OK** push button. Resize the literal so that all of the lettering shows.
4. Add the line:
 - a. From the Menu Bar select **Add**, then select **Literal....** Then, enter the underscore key approximately 40 times. Un-check **Center Vertically** and **Center Horizontally**, select the **OK** push button and position the literal roughly as indicated in the example shown earlier.
 - b. Select the underscore literal first, and then with the Ctrl key held down, select the **eGolfer Login** literal. From the Menu Bar select **Edit**, then **Position....** Now select **Align Vertically Left**, select **Separate Vertically** by a Distance: of **5**, select **Move Horizontally Center**, and then select the **OK** push button.
5. Place the two fields on the screen:
 - a. From the Menu Bar select **Edit**, then select **Field Design....** Then from the Field Design panel, select **EXPORT GOLFER USERID**, select the **Properties...** push button, and verify that the **EXPORT GOLFER USERID** is mapped to the **IMPORT**. Then, from the Prompt: drop-down list, select **User ID:**, select the **OK** push button, and then select the **Place** push button and position the field (prompt first and then the field) on the dialog in roughly the position indicated on the example.
 - b. From the Menu Bar select **Edit**, then **Field Design....** Then from the Field Design panel, select **EXPORT GOLFER PASSWORD**, select the **Properties...** push button, and verify the **EXPORT GOLFER PASSWORD** is mapped to the **IMPORT**. Then, from the Prompt: drop-down list, select **Password:**, check the **Password** check box, select the **OK** push button, and then select the **Place** push button and position the field (prompt first and then the field) on the dialog in roughly the position indicated on the example. Resize the prompt if necessary.

- c. Select the **Password** prompt and then, with the Ctrl key held down, select the **User ID** prompt (not the fields). From the Menu Bar, select **Edit**, select **Position...**, check the **Equal Width** check box, and then select the **OK** push button. Then select the **Password** field and, with the **Ctrl** key held down, select the **User ID** field.
 - d. After making sure that only the Password field and the User ID field have focus, from the Menu Bar select **Edit**, then select **Position...**. Then select **Align Vertically Left**, select **Separate Vertically** by a Distance: of **5**, check **Arrange prompts against the fields**, and then select the **OK** push button.
 - e. Finally, using the cursor, lasso both the fields and prompts (but not the two literals) by click-and-dragging a box around them. Then select **Edit**, select **Position...**, select **Move Horizontally Center**, and then select the **OK** push button.
6. Add the back push button:
 - a. From the Menu Bar select **Add**, then select **Push Button**.
 - b. In the Text: field enter **<Back**, select the radio button **Special Action**, in the drop-down list select **Cancel – Close without Execution**, and then select the **OK** push button and position the push button roughly as indicated in the example.
7. Add the login push button:
 - a. From the Menu Bar select **Add**, then select **Push Button**. In the Text: field enter **login**, check the **Is the Default Push Button** check box, select the **Events...** push button, select the **Add** push button to automatically add a new event, select the **Close** push button, and then select the **OK** push button and position the push button roughly as indicated in the example.
 - b. Resize the login push button with the cursor, making it not quite so tall. Then with the **Ctrl** key held down, select the **<Back** push button. From the Menu Bar select **Edit**, then **Position....**. Then select the **Equal Height** and **Equal Width** check boxes, select **Align Horizontally Bottom**, select **Separate Horizontally** by a Distance: of **20**, select **Move Horizontally Center**, and then select the **OK** push button.
8. Disable the login push button:
 - a. Select the **login** push button. Then from the Menu Bar, select **Detail**, then **Disabled By...**
 - b. In the Disabling States dialog, select **THE Push Button 'login' IS DISABLED WHEN**, select the condition Entry Field 'User ID:' does not have data, and then select the **Add** push button. Then, select the OR state, select the condition Entry Field 'Password:' does not have data, select the **Add** push button, and then select the **Close** push button.

9. Add the action diagram logic:
 - a. Since the login logic is very similar to the registration logic, we can just copy the registration logic and modify it. From the Toolbar, select the **Action Diagram** icon. In the **EVENT ACTION egolfer reg pb next click** event handler, click-and-drag the cursor from the **COMMAND IS** statement to the bracket ending the **IF EXITSTATE IS** statement to highlight the section of code shown:

```
COMMAND IS register
USE maintain_golfer (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
  WHICH EXPORTS: Entity View export golfer FROM Entity View export golfer
  IF EXITSTATE IS EQUAL TO processing_ok
  [ EXIT STATE IS xfr_to_egolfer_home
    CLOSE Dialog Box GOLFER_REGISTRATION
```

- b. Select the **F8 (copy)** key and, with the cursor (which now resembles a hand), select the **EVENT ACTION egolfer_log_pb_login_click** event handler. Then double-click the command value **register** in the login click event, select the expression command **value** in the Change panel, select the Command **LOGIN**, select the **OK** push button, and then select the **Change** push button.
 - c. Now, double-click **EGOLFER REGISTRATION** in the **CLOSE** Dialog statement in the login click event, select **Dialog Box EGOLFER LOGIN** in the Window Selection panel, and then select the **OK** push button.
10. Save your model.

11. The completed action diagram should look like the following example. Close the action diagram.

EGOLF SERVICES HOME**IMPORTS:**

Entity View import_confirmation golfer (optional,transient,import only)
password (optional)

Entity View import golfer (optional,transient,import only)

password (optional)
first_name (optional)
last_name (optional)
email_address (optional)
handicap_index (optional)
userid (optional)

EXPORTS:

Entity View export_confirmation golfer (transient,export only)
password

Entity View export golfer (transient,export only)

password
first_name
last_name
email_address
handicap_index
userid

LOCALS:

Entity View local golfer

password
first_name
last_name
email_address
handicap_index
userid

ENTITY ACTIONS:



12. **Close** the eGolf Services Login window.

eGolfer Home

Refer back to the window examples shown earlier and familiarize yourself with the eGolfer Home Web page and the pages directly accessed from it. On this window, we have an advertising banner, two group boxes, some push buttons, and some entry fields, including a scrolling list box.

To get to this page, golfers have had to either successfully register or login. When this page comes up, it will automatically be populated with the golfer's name and handicap index (if there is one) passed to it from the eGolf Services action diagram. Additionally, we want it to open pre-populated with a listing of the golfer's scoring records, if the golfer has any. To get the scoring records, we will assign an open event to the dialog that will call the List Scoring Record server procedure. Open events are triggered when the window or dialog they are associated with are opened. They perform any logic associated with them prior to the actual display of the dialog or window.

Once here, golfers can then add, update, or remove scoring records. They can also update their own personal profiles or remove themselves entirely from our system. When updating information about themselves, or removing themselves from our database, we need only call the Maintain Golfer server. However, when adding, updating, or deleting scoring records, there are three server procedures that we need to call:

- The Maintain Scoring Record server procedure to make the indicated change
- The List Scoring Record server procedure to re-populate/sort the scoring record list on our primary dialog
- The Maintain Golfer server procedure to recalculate the handicap index based on this new information.

There are ways that are more efficient to handle this, but this will keep our logic relatively simple.

Reviewing the previous information and the window examples, we can determine that, for our eGolfer Home primary dialog and its associated secondary dialogs; we are going to need the combined views from the three server procedure steps. To get a jump start on creating the views, we can go into the eGolfer Home procedure step action diagram, add procedure step use statements for each of the three servers, and use match and copy to create the necessary import and export views.

Add Procedure Step Use Statements for Servers

To add Procedure Step Use Statements for Servers, read the following sections.

Maintain Golfer

Follow these steps:

1. In the Navigation Diagram, select the **eGolfer Home** primary dialog. Then from the Toolbar, select the **Action Diagram** icon. Within the EGOLFER HOME procedure step action diagram, select the **<blank line>** following the empty ENTITY ACTIONS: view. From the Menu Bar select **Edit**, then **Add Statement**, then **Procedure Step Use....** In the Add Statement dialog, select the non-display procedure step maintain golfer, and then select the **Add** push button.
2. In the MAINTAIN GOLFER Import View Matching dialog, under the Import Views for MAINTAIN GOLFER, select **IMPORT GOLFER**. There are no Possible Supplying Views in our new action diagram yet, so we want to select the **Match** push button to open the Match and Copy dialog.
3. Accept the default Import View by selecting the **Copy** push button then select the **OK** push button. A matching import view is created.
4. Select the Close push button to bring up the MAINTAIN GOLFER Export View Matching dialog. Under the MAINTAIN GOLFER Export Views, select **EXPORT GOLFER**. There are no Possible Receiving Views in our new action diagram yet, so we want to select the Match push button to open the Match and Copy dialog.

5. Accept the default Export View by selecting the **Copy** push button, then select the **OK** push button. A matching export view is created.
6. Select the **Close** push button to complete the view-matching dialog.
7. Expand all of the views by selecting **View** from the Menu Bar, then **Expand All Views**.

Maintain Scoring Record

When calling the Maintain Scoring Record server procedure, we have to tell it which golfer the scoring record is for. Thus, in addition to it requiring information about the scoring record, it also requires information about the golfer that the scoring record is for. We will reuse the golfer views we just added for the Maintain Golfer server procedure step, but we will use Match and Copy to create the new views for the scoring record. The Maintain Scoring Record server has no export views.

Follow these steps:

1. Repeat the steps we used to add a procedure step use statement for the Maintain Golfer procedure step to add a procedure step use statement for the Maintain Scoring Records procedure step. However, when matching views, use the existing golfer from EGOLFER_HOME instead of adding a new golfer, and accept the name for the new scoring record import view created through the Match and Copy dialog import.
2. Your action diagram should now look like the following example:

```

EGOLFER_HOME
IMPORTS:
  Entity View import scoring_record (optional,transient,import only)
    date (optional)
    time (optional)
    adjusted_gross_score (optional)
    course_rating (optional)
    course_slope_rating (optional)
    note (optional)
  Entity View import golfer (optional,transient,import only)
    password (optional)
    first_name (optional)
    last_name (optional)
    email_address (optional)
    handicap_index (optional)
    userid (optional)
EXPORTS:
  Entity View export golfer (transient,export only)
    password
    first_name
    last_name
    email_address
    handicap_index
    userid
LOCALS:
ENTITY ACTIONS:
  USE maintain_scoring_record (procedure step)
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
                  Entity View import scoring_record TO Entity View import scoring_record
  USE maintain_golfer (procedure step)
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
    WHICH EXPORTS: Entity View export golfer FROM Entity View export golfer

```

If your action diagram does not look like the previous example, take time now to correct it. You should only have two import views, one of import scoring record and the other of import golfer. The order of the two views is not important; in other words, you can have import golfer and then import scoring record. In the export views you should only have one view, that of the export golfer. All three of these views should have the same attributes in them as shown, but the order of the attributes within the views is not important. Likewise, the order of the two USE statements is not important either; eventually we are going to move them down into event handlers.

Correcting the views is relatively easy to do. Here are some tips on how to fix them:

- If you have the correct three views and the only thing wrong with them is you mistyped or forgot to enter a name for each view, simply double-click the Entity View name and correct it.
- If you have a duplicate import golfer, then it is probably because you did not reuse the import golfer that was already there and instead created a new golfer using Match and Copy. If that is the case, the second golfer cannot be named import (since we already had a golfer named import) and you had to either misspell the name import to add it or you did not give it any name at all. Furthermore, the use statement of the Maintain Scoring Record server probably references this extraneous view.
- To fix this problem, first you have to change the reference to it; then you can delete it from the views. Double-click the reference to it in the USE statement. In the MAINTAIN SCORING RECORD Import View Matching dialog, select **IMPORT GOLFER** from under the MAINTAIN SCORING RECORD Import View. You should see that it is matched to the incorrect view. Select the **Unmatch** push button to unmatch the two views. Then, from the Possible Supplying Views shown in the bottom panel, select **IMPORT GOLFER** and then select the **Match** push button. Then select the **Close** push button. It should now be matched to the correct import view and you can delete the extra view by selecting it in the IMPORTS: and then selecting the **Delete...** icon from the Toolbar.

Your diagram should now look the same as the example shown, notwithstanding the order of the views or statements.

3. For procedure steps, and particularly for client procedure steps, the import views and the exports view usually need to be identical. In fact, the occasions for when they are not identical are so rare it is not even worth thinking about. For client procedures steps, then, make your import and export views the same.

To make these the same, we need to copy the import scoring record to the EXPORTS. Select **import-scoring record**, then press **F8** on your keyboard and with the cursor (which now resembles a hand), select **EXPORTS**. Name the copied import view **export**, and select the **OK** push button.

4. Save your model.

List Scoring Record

We can now add the call to the final server procedure step. The List Scoring Record server procedure requires a golfer as input, and produces a list (group view) of scoring records for that golfer as output. We want to reuse the existing import golfer view, but we will use Match and Copy for the export group view.

Follow these steps:

1. Repeat the previous steps to add the procedure step use statement for the List Scoring Record server procedure.

Your import views should look the same as they did before, but your export views should now have the export group of scoring records added to it as in the example:

```
EXPORTS:
Group View export_group_of_scoring_records (40,implicit,export only)
Work View export_group ief_supplied (transient)
select_char
Entity View export_group scoring_record (transient)
date
time
adjusted_gross_score
course_rating
course_slope_rating
note
Entity View export scoring_record (transient,export only)
date
time
adjusted_gross_score
course_rating
course_slope_rating
note
Entity View export golfer (transient,export only)
password
first_name
last_name
email_address
handicap_index
userid
```

2. To make the import views look exactly the same as the export views, copy the export group of scoring records to the **IMPORTS:** using the name **import group of scoring records** to replace export group of scoring records and **import group** to replace export group.
3. The Update eGolfer dialog actually has two views of a golfer's password. One view is used to confirm the other. We only have one view of a golfer containing all of its attributes, including the password. Therefore, we need to add another view of a golfer, containing just the password attribute.

In the action diagram select **EXPORTS:**, and from the Menu Bar select **Edit**, then **Add View**, and then **Add Entity View**. In the Name entry field, enter **export confirmation**. From the list of Entity Types select **GOLFER**, from the list of attributes select **PASSWORD**, and then select the **OK** push button.

4. Copy that new view to the **IMPORTS:** and name it **import confirmation**.

5. If a golfer decides to add two or more scoring records during the same session, then we will want the Add Scoring Record dialog to be cleared each time. In a similar fashion to what we did in the eGolf Services home procedure step, we want a local blank scoring record. Copy the **import scoring record** to the **LOCALS:** and name the new view **local blank**.
6. It can be helpful at this point to arrange the views in a more logical sequence. In the following example, we have reordered the views such that the golfer appears first, followed by the confirmation golfer, and followed by the single entity view of a scoring record, followed by the group of scoring records. You can do this by first selecting the Entity View, pressing the **F7** key on your keyboard, and then selecting the Entity View you want to move the selected view directly underneath.

At this point, your action diagram should look like the following example:

```

EGOLFER_HOME
IMPORTS:
  Entity View import golfer (optional,transient,import only)
    password (optional)
    first_name (optional)
    last_name (optional)
    email_address (optional)
    handicap_index (optional)
    userid (optional)
  Entity View import confirmation golfer (optional,transient,import only)
    password (optional)
  Entity View import scoring_record (optional,transient,import only)
    date (optional)
    time (optional)
    adjusted_gross_score (optional)
    course_rating (optional)
    course_slope_rating (optional)
    note (optional)
  Group View import_group_of_scoring_records (optional,40,implicit,import only)
  Work View import_group ief_supplied (optional,transient)
    select_char (optional)
  Entity View import_group scoring_record (optional,transient)
    date (optional)
    time (optional)
    adjusted_gross_score (optional)
    course_rating (optional)
    course_slope_rating (optional)
    note (optional)

EXPORTS:
  Entity View export golfer (transient,export only)
    password
    first_name
    last_name
    email_address
    handicap_index
    userid
  Entity View export_confirmation golfer (transient,export only)
    password
  Entity View export scoring_record (transient,export only)
    date
    time
    adjusted_gross_score
    course_rating
    course_slope_rating
    note
  Group View export_group_of_scoring_records (40,implicit,export only)
  Work View export_group ief_supplied (transient)
    select_char
  Entity View export_group scoring_record (transient)
    date
    time
    adjusted_gross_score
    course_rating
    course_slope_rating
    note

LOCALS:
  Entity View local_blank scoring_record
    date
    time
    adjusted_gross_score
    course_rating
    course_slope_rating
    note

ENTITY ACTIONS:
USE list_scoring_record (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
  WHICH EXPORTS: Group View export_group_of_scoring_records FROM Group View export_group_of_scoring_r
USE maintain_scoring_record (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
  Entity View import scoring_record TO Entity View import scoring_record
USE maintain_golfer (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
  WHICH EXPORTS: Entity View export golfer FROM Entity View export golfer

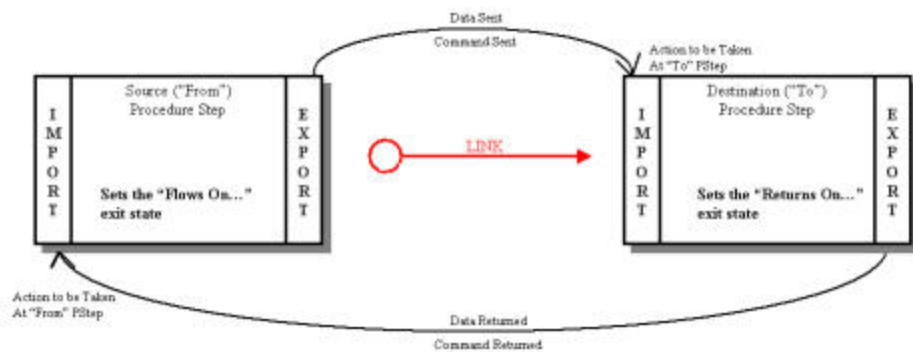
```

7. Save your model.
8. Make any corrections necessary and save your model again.
9. Close the action diagram.

Complete the Flows

Now that we have our views defined in the eGolfer home page, we can complete the flow from the eGolf Services home page to the eGolfer home page.

If you recall, after successfully registering or logging in to the system, the golfer should be sent to their home page. We already set the triggering mechanism. Within the eGolf Services procedure step action diagram logic we set an exit state to xfr to egolfer home. Flows from one client procedure to another are triggered when there is a flow defined between the two procedure steps, and the source procedure step completes execution with the exit state value set to the defined flows on exit state. Within the flow, we can also pass data and commands. Data is matched from the export view of the source procedure to the import view of the destination procedure.



There are two types of flows: Links and Transfers. For client/server-type applications, including Web applications, we usually use Links. That said, for this particular application, we are going to use two Transfer flows.

Links are essentially two-way flows. In addition to flows on exit state value, you also define a Returns on Exit state value. It is the destination procedure's responsibility to set the Returns On exit state value. You can also return data from the destination procedure step's export views back to the source procedure steps import views, along with a command.

One other property of a flow is the action you want to take when you get to the procedure you are flowing to or from. For a link, there are two actions to define: The action you want to take at the top procedure step when you get there, and the action you want to take back at the from procedure step when you return.

The choice of action to take includes Display First or Execute First.

- **Display First**—No logic is executed at the destination procedure step. The screen or window associated with the destination procedure step is displayed, along with any data passed to it.


- **Execute First**—Execute the logic in the destination procedures step's action diagram. After executing the logic, it can display the screen or window associated with it, or it can flow somewhere else, depending on the logic.

For Windows and Web applications, we usually use Display First actions. Any logic we want executed prior to the window actually appearing is usually placed in an open event for the destination procedure.

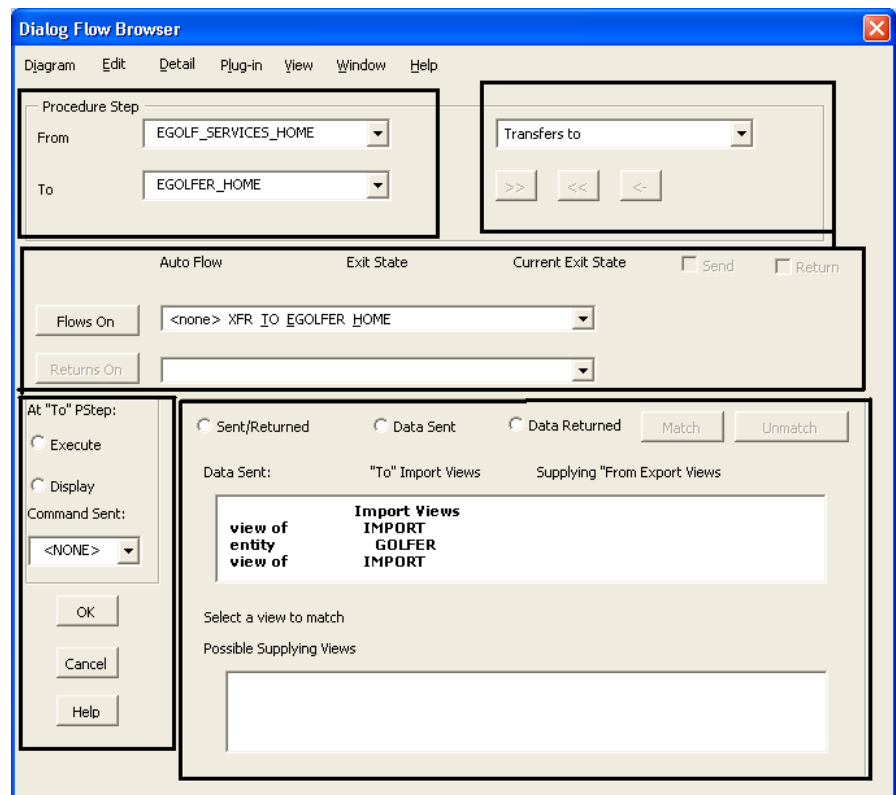
The Dialog Flow Browser

Let us look at the Dialog Flow Browser, which is used to detail the flows between Client procedure steps.

Follow these steps:

1. To open the Dialog Flow Browser, select **eGolf Services** from the Navigation Diagram and, while holding down the **Ctrl** key on your keyboard, select **eGolfer Home**. Then, from the Toolbar, select the **Join...**  icon to bring up the Dialog Flow Browser.

There are essentially five sections to the Dialog Flow Browser:



At the top, the browser displays the two procedure steps involved in the flow. In our case, we are flowing From: EGOLF SERVICES HOME (source) To: EGOLFER HOME (destination).

2. To the right of the procedures, we specify whether we want this flow to be a Transfer or a Link. Change the drop-down list to **Transfers to**.

If we were using a Link type flow, we would have to define two of everything, since a Link essentially represents a two-way flow. We would have to specify two exit state values, two sets of actions to be taken, two commands, and two sets of view matching if necessary. We will still have to define two of everything, but only once, for each of the two Transfer flows we will define.

3. The Flows On... exit state represents the exit state value the From: procedure step needs to set to initiate the flow. The exit state value we want to trigger the flow from the EGOLF SERVICES HOME page to the EGOLFER HOME page is XFR TO EGOLFER HOME. Select the **Flows On...** push button. Select the business system **EGOLF SERVICES** and expand it by selecting the **Expand/Contract** push button. Select the exit state value **XFR TO EGOLFER HOME** and select the **Select** push button.

You may be thinking that we already set that exit state value in the eGolf Services Home procedure step. However, merely setting the exit state system attribute to some value does not automatically cause a flow. We have to add the flow (as we are doing now), and for each flow that we add, we have to define what value the exit state system attribute has to have in order for this particular flow to be taken. We can have multiple flows emanating from a procedure step, but only one will be taken upon completion of that procedure step, and it will be the flow with the matching flows on exit state value defined for it.

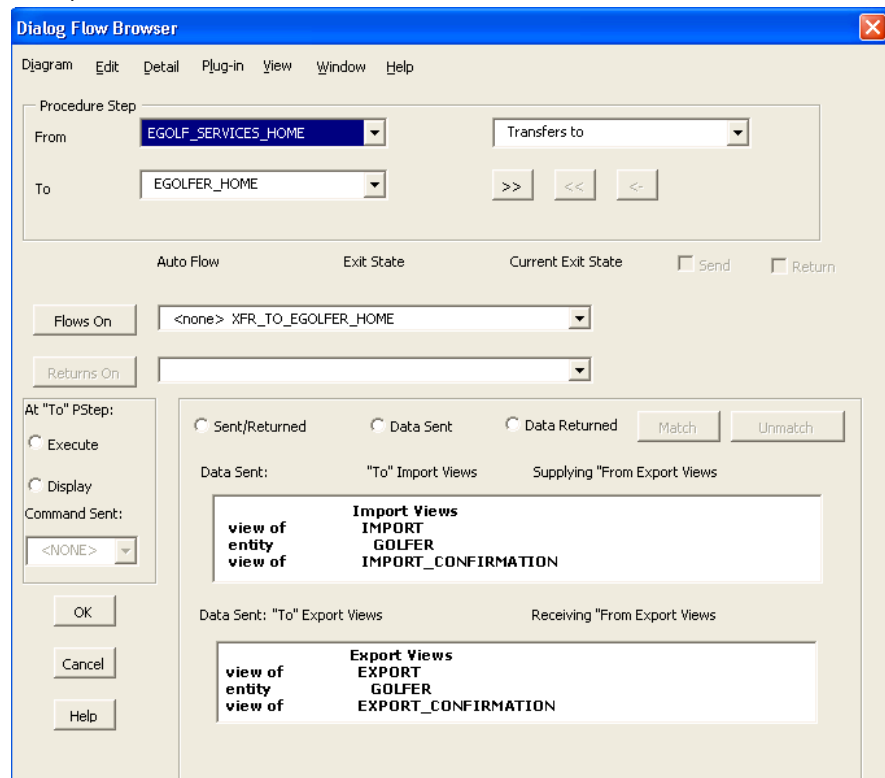
If this were a Link, the Returns On... exit state would be the exit state value the To: procedure step would need to set to initiate the return back to the From: procedure step.

4. To the left side of the Dialog Flow Browser are the actions to be taken and commands to be passed. For these types of applications, we normally use Display actions and seldom have a need to send a command. For the action to be taken At 'To' Pstep:, select the **Display** radio button.

- Finally, we need to specify the data we want sent from EGOLF SERVICES HOME to EGOLFER HOME. Select the **Data Sent** radio button. This is similar to the standard view-matching dialog. To the left side of the top panel it shows the import views of the To procedure step (in this case, the EGOLFER HOME procedure step). To the right of those views, the browser displays the views from the from procedure step that we have matched to them; presently there are none. We know that we want to pass the golfer information from the EGOLF SERVICES HOME page to the EGOLFER HOME page. For flows, we always match exports to imports. Therefore, we want to match the export golfer in EGOLF SERVICES HOME to the import golfer in EGOLFER HOME.

In the top panel, which shows the import views in the destination procedure, select the **IMPORT GOLFER**. The bottom panel now shows the Possible Supplying Views (export views) from the EGOLF SERVICES HOME. Scroll down in the bottom panel if necessary and select **EXPORT GOLFER** and then select the **Match** push button. In the top panel you can now see the two matched views.

- Select the **OK** push button at the bottom left of the dialog.
- Save your model.
- Set up another Transfer, this time from EGOLFER HOME to EGOLF SERVICES HOME. There is no data to send. The Dialog Flow Browser should look like the following example:



9. Save your model.

Note: When you add new objects to the Navigation Diagram, the diagram can show dashed or self-referencing lines, indicating a possible incomplete definition of some objects. To refresh the Navigation Diagram, close and re-open it.

Get Scoring Records

Now that we have the golfer information passed to eGolfer Home, we can get the scoring records for that golfer. Since we want this information populated when the dialog opens, we will add an open event to the eGolfer Home primary dialog.

Follow these steps:

1. Double-click eGolfer Home to bring up the primary dialog. From the Menu Bar select **Detail**, and then select **Events**.
2. In the Event Processing dialog, change the Event Type: drop-down selection to **Open**, select the **Add** push button to allow the Toolset to generate an event name, and then select the **Close** push button.
3. The next step is to detail this event. From the Toolbar, select the **Action Diagram** icon. Add the statement **Command is...** for the command value of **LIST** to the **EVENT ACTION** egolfer_home_open. Then move the **USE list scoring record statement** from the main body of the action diagram to the event. You can move the statement by selecting it, pressing **F7** on your keyboard, and then selecting the **COMMAND IS** statement to move it directly below that statement.

The action diagram should now look like the following example:

```

EGOLFER HOME
IMPORTS: ...
EXPORTS: ...
LOCALS: ...
ENTITY ACTIONS:

USE maintain_scoring_record (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
                Entity View import scoring_record TO Entity View import scoring_record
USE maintain_golfer (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
  WHICH EXPORTS: Entity View export golfer FROM Entity View export golfer


EVENT ACTION egolfer_home_open
COMMAND IS list
USE list_scoring_record (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
  WHICH EXPORTS: Group View export_group_of_scoring_records FROM Group View export_group_of_scoring_record

```

Lay Out the GUI Control Objects

We can now begin to layout the GUI data and non-data control objects of the window.

Follow these steps:

1. In the Navigation Diagram, double-click **eGolfer Home** to bring up the dialog design. Double-click another dialog and resize eGolfer Home relative to the other dialog. **Close** the other dialog.
2. With the eGolfer Home widow resized correctly, from the Menu Bar select **Edit**, then select **Field Design....**
3. From the Field Design dialog, select **EXPORT GOLFER FIRST_NAME**, then select the **Properties...** push button. In the Entry Field Properties dialog, change the mapping property from [none] to IMPORT, add a new prompt named **Welcome**, un-check the **Margin Box** check box, check the **Read Only** check box, select the **OK** push button, and finally select the **Place** push button. Then place the prompt followed by the field in roughly the position indicated by the example shown earlier.
4. Now, we want to change the default sizes for this field. First, select the prompt **Welcome**, then from the Toolbar select the **Video Properties...**  icon. Uncheck the **Automatic** checkbox for Font and press the **Select...** pushbutton to bring up the Font dialog. Change the Font to **Arial/Bold Italic/14**, then select the **OK** push button. Resize the prompt so that you can see all of the text.
5. Change the font for the entry field the same as in the last step. Do not worry if you cannot get the entire field to display. Leave room for the Group Box around the right-hand side.
6. Repeat steps two through five for the EXPORT GOLFER HANDICAP INDEX, mapping it to the IMPORT GOLFER HANDICAP INDEX and adding the new prompt Your Handicap Index is. Do not forget to un-check the **Margin Box** check box and to check the **Read Only** check box. Change the fonts to **Arial/Bold Italic/14**.
7. To add the group box around the golfer's information, select **Add** from the Menu Bar, and then select **Group Box....** In the Group Box Properties panel, enter **Personal Profile** in the Text: field, and then select the **OK** push button. Place the group box just above the two push buttons. If you have multiple adds turned on, select the **Cancel** push button when prompted to add the second group box. Then, click-and-drag the edges of the group box with the cursor such that it encircles the golfer information as shown in the window example earlier.

Add Push buttons

We can now add the two push buttons. One will open the update golfer dialog, and the other will open the remove golfer dialog.

Follow these steps:

1. From the Menu Bar select **Add**, and then select **Push Button....** In the Push Button Properties dialog, enter the Text: **update**, and then select the **Events...** push button.
2. This time we want to give the event a name of our own choosing. This procedure step will have twelve events before we are through, and it starts to get a bit complicated unless we use some very straightforward event names. For the Event Details Action Name: enter **open update golfer db**. Select the **Add** push button to add the new event, then select the **Close** push button, then the **OK** push button. Position the push button within the Personal Profile group box as shown in the example earlier.
3. Repeat steps one and two to add the remove push button. Name the new event **open remove golfer db**.
4. Save you model.

Add the List Box

Follow these steps:

1. To add the List Box, from the Menu Bar select **Edit**, and then select **List Box Design....** In the List Properties dialog, make sure the Export View: EXPORT GROUP OF SCORING RECORDS is mapped to the Import View: IMPORT GROUP OF SCORING RECORDS. List Boxes are added to the window design all at once, and it is easier to define all of the field and prompt properties prior to adding them.
2. First of all, check to make sure the fields are in the order you want them to appear in the list box on the window. In our case, we want them to be in the following order:
 - a. SELECT CHAR
 - b. DATE
 - c. TIME
 - d. ADJUSTED GROSS SCORE
 - e. COURSE RATING
 - f. COURSE SLOPE RATING
 - g. NOTE

If they are not in the correct order, select the attribute you want to move, select the **Move** push button, and then select the attribute you want to move the selected attribute directly beneath. If you have multiple attributes to move, you will need to unselect the previous one before attempting to move the next one.

3. The next thing we want to do is define the Properties for each attribute. Unselect any selected attributes and then select the **SELECT CHAR**. Select the **Properties...** push button. If the Properties... push button is not enabled, then it is because you have more than one attribute selected. In the List Box Field Properties dialog, if the Import View: indicates [none], ignore it. Check the check box indicating that we want to use this field for the **Selection Indicator**. While this field will not explicitly be placed on our window design, it will be used to keep track of highlighted/selected fields. Select the **OK** push button.
4. Unselect **SELECT CHAR**, select **DATE**, and select the **Properties...** push button. If necessary, add the prompt **Date**. Then change the Prompt Align: to **Center**, select the Edit Pattern: **MM/DD/YY**, change the Edit Align: to **Center**, and then select the **OK** push button.
5. Unselect **DATE**, select **TIME**, and select the **Properties...** push button. If necessary, add the prompt **Time**. Then change the Prompt Align: to **Center**, add the Edit Pattern: **HH:MM**, change the Edit Align: to **Center**, and then select the **OK** push button.
6. Unselect **TIME**, select **ADJUSTED GROSS SCORE**, and select the **Properties...** push button. Then add the prompt **Score**, then change the Prompt Align: to **Center**, add the Edit Pattern: **ZZZ**, change the Edit Align: to **Center**, and then select the **OK** push button.
7. Unselect **ADJUSTED GROSS SCORE**, select **COURSE RATING**, and select the **Properties...** push button. Then add the prompt **Rating**, change the Prompt Align: to **Center**, add the Edit Pattern: **ZZ.Z**, change the Edit Align: to **Center**, and then select the **OK** push button.
8. Unselect **COURSE RATING**, select **COURSE SLOPE RATING**, and select the **Properties...** push button. Then add the prompt **Slope**, change the Prompt Align: to **Center**, add the Edit Pattern: **ZZZ**, change the Edit Align: to **Center**, and then select the **OK** push button.
9. Unselect **COURSE SLOPE RATING**, select **NOTE**, and select the **Properties...** push button. Then add the prompt **Note**, leave the Prompt Align: to **Left**, leave the Edit Pattern: **<Default Edit Pattern>**, leave the Edit Align: **Default**, and then select the **OK** push button.
10. Since the entire Note will not appear on the window, select **Horizontal Scroll**. Then select **Adjust Position** so that only whole lines will appear in the List Box. We only want to allow **Single Selections**.
11. To identify which fields to actually use in the list box, select all of the attributes except the **SELECT CHAR**, and then select the **Include** push button.

12. Finally, to add the list box to the window design, select the **OK** push button and place it onto the window. The list box will be positioned to cover the entire width of the window. Resize it by using the right mouse button to drag it slightly off center, and then select an edge of the list box and drag it to the proper size. Use the window example shown earlier to gauge your dimensions.

You may have to slightly resize a column to get the entire field to display. If so, slowly move your mouse between two columns in the list box until it turns into a double-sided arrow. Then click-and-drag the column to the desired width. You may also have to resize the prompts by selecting one of them on the window, and then slightly resizing the box they appear in. If something does not look quite centered, or the edit patterns are not what you thought they should be, double-click on the list box to open its List Properties panel and review each individual attribute's properties once again.

13. Add a group box around the scoring records list box, making sure to leave room for the push buttons. Name the group box **Scoring Record**.
14. Add three push buttons named **add**, **update**, and **remove**. The add push button should execute an event named **open add scoring record db**, the update push button should execute an event named **open update scoring record db**, and the remove push button should execute an event named **open remove scoring record db**. Make the add push button the default push button for this window by selecting the **Is the Default Push Button** check box.
15. Add a **logout** push button to the bottom of the window. Add an event for it, letting the Toolset generate the event name.

Add the Advertising Banner

Follow these steps:

1. To add the advertising banner to the top of the window, select Add from the Menu Bar, then select **Picture....**. Then from the Bitmap: drop-down list, select **adbanner**, in the Name: entry field enter **picture1**, in the HTML Extension: entry field enter **gif**, and select the **OK** push button. Position the banner at the top of the dialog.

If the adbanner overlays the Personal Profile group box, reposition it. Then select the bottom edge of the group box, and with the right mouse button, click-and-drag the group box to a better position.

2. Save your model.

Position Objects on the Dialog

Now that we have all of the GUI objects added to the primary dialog, we can do a little cleanup as to the size and position of the objects using the positioning feature.

Follow these steps:

1. Choose the remove push button (as it has the longest name) make it a little smaller. Then, with the **Ctrl** key held down, select all of the other push buttons and from the Menu Bar select **Edit**, and then **Position....** Check the **Equal Height** and **Equal Width** check boxes, and then select the **OK** push button.
2. Now that all the push buttons are the same size, you can choose one to use as the pivot point and align the rest (not including the logout push button) vertically left or right relative to the selected one. Separate the groups of buttons within each group box by a vertical distance of **5**.
3. Make the two group boxes of equal width and position one vertically left or right relative to the first one selected.
4. Position the prompt (**Welcome**) and the field (**First Name**) horizontally middle relative to each other, set them to equal height, and separate them horizontally by **5**.
5. Position the prompt (**Your Handicap Index is**) and the field (**Handicap Index**) horizontally middle relative to each other, set them to **equal height**, and separate them horizontally by **5**.
6. Align the two fields containing the name and handicap index horizontally relative to the two push buttons.

A few other tweaks can be required, but by now your dialog, design should be looking fairly close to that shown in the window example earlier.

Add Push Button Disabling Logic

A golfer should not be able to update or delete a scoring record unless they have first selected a scoring record from within the list of scoring records. Thus, both the update and remove scoring record push buttons need disabling logic. Set the disabling states for both the update and remove scoring record push buttons to be disabled when the **listbox 'Date' has none selected**.

Detail the Logic to Open the Secondary Dialogs

We can now detail the logic for the six events we added. If you recall, we added one event for each push button we added. For the most part, each event is going to open a secondary dialog. Before we can open a secondary dialog, we first have to create the secondary dialogs.

Follow these steps:

1. Close the eGolfer Home primary dialog design by selecting the **Close Window** icon from the Toolbar. Then, from the Tool Palette, select the **Add Dialog** icon and click directly on the **eGolfer Home** primary dialog. In the Window / Dialog Properties, change the Initial Position to **System Placed**, un-check **System Menu**, enter **Update eGolfer** for the Title, and enter **update egolfer db** for the Name, then select the **OK** push button. Locate the new dialog on the Navigation Diagram and drag it to the position roughly indicated in the picture of the Navigation Diagram earlier.
2. Add the four remaining secondary dialogs in the same manner. See the following table for the titles and names of the dialogs. If your cursor indicates that you cannot select the primary dialog, ignore it.

Title	Name
Remove eGolfer	remove egolfer db
Add Scoring Record	add scoring record db
Update Scoring Record	update scoring record db
Remove Scoring Record	remove scoring record db

- From the Navigation Diagram, select the **eGolfer Home** primary dialog. From the Toolbar, select the **Action Diagram** icon. Your action diagram should now contain the six empty event handlers as in the following example. We will detail each of these event handlers in the next section.

```

EGOLFER_HOME
IMPORTS: ...
EXPORTS: ...
LOCALS: ...
ENTITY ACTIONS:

USE maintain_scoring_record (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
               Entity View import scoring_record TO Entity View import scoring_record
USE maintain_golfer (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
  WHICH EXPORTS: Entity View export golfer FROM Entity View export golfer

EVENT ACTION egolfer_home_open
COMMAND IS list
USE list_scoring_record (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
  WHICH EXPORTS: Group View export_group_of_scoring_records FROM Group View export_group_of_scoring_records

EVENT ACTION open_update_golfer_db
EVENT ACTION open_remove_golfer_db
EVENT ACTION open_add_scoring_record_db
EVENT ACTION open_update_scoring_record_db
EVENT ACTION open_remove_scoring_record_db
EVENT ACTION egolfer_home_pb_logout_click

```

- Save your model.

Review Secondary Dialog Design

Follow these steps:

- Review the design for the update golfer secondary dialog. When this dialog opens, we want all of the golfer information displayed, except for the password. Since we do not have a separate dialog for handling password changes, we want the golfer to re-enter and verify their password as part of any changes made.
 - In the action diagram, select the event handler **EVENT ACTION open update golfer db**. Then, from the Menu Bar select **Edit**, then **Add Statement**, and then **Set...**
 - From the options, select **attribute view**, from the list of entity views select **export golfer**, and from the list of attributes select **password**. Then select the **operator TO** expression, select the expression **spaces**, and then select the **Add** push button to add the statement to the event handler. We want to add another Set statement just like the last one, only we want to set the export confirmation golfer password to spaces. To accomplish this, we can copy the statement we created in step b and change the entity view.

With the statement you just added still highlighted, press **F8** on your keyboard, and with the cursor (which now resembles a hand), select the **SET** in the highlighted statement again. This will add the new statement directly below the highlighted statement.

- c. Double-click export golfer in the second statement. In the Change dialog, select **attribute view**, then select the **entity view export confirmation golfer**. Since there is only one attribute view in the export confirmation golfer entity view, and since that attribute is of a character domain, the Toolset selected it for us automatically. Select the **Change** push button to change the statement. Now, we can add the open dialog statement. Select the **SET** in the second Set statement. Now, from the Menu Bar select **Edit**, then **Add GUI Statement**, and then **Open....** Select **Dialog UPDATE EGOLFER DB**, then select the **OK** push button.

The completed event handler should like the following example:

```
EVENT ACTION open_update_golfer_db
SET export golfer password TO SPACES
SET export_confirmation golfer password TO SPACES
OPEN Dialog Box UPDATE_EGOLFER_DB
```

- d. Save your model.
2. Review the design for the remove golfer secondary dialog. When this dialog opens, we are just going to display the golfer's name and email address and ask for confirmation of the delete.
 - a. In the action diagram, select the **event handler EVENT ACTION open remove golfer db**. Then, from the Menu Bar select **Edit**, then **Add GUI Statement**, and then **Open....** From the Window Selection list select **Dialog REMOVE EGOLFER DB**, and then select the **OK** push button to add the statement to the event handler.

The completed event handler should look the following example:

```
EVENT ACTION open_remove_golfer_db
OPEN Dialog Box REMOVE_EGOLFER_DB
```

- b. Save your model.

3. Review the design for the add scoring record secondary dialog. When this dialog opens, it should be blank. Remember that if a golfer adds several scoring records during one session, the information from the previous scoring record will be redisplayed unless we explicitly prevent that from happening. In this case, we just want to move the local blank scoring record to the export scoring record prior to opening the secondary dialog.
 - a. In the action diagram, select the **event handler EVENT ACTION open add scoring record db**. Then, from the Menu Bar select **Edit**, then **Add Statement**, and then **Move....**
 - b. From the list of entity views, select **local blank scoring record**. The Toolset automatically selects the **export scoring record**. Select the **Add** push button to add the statement. Now add the GUI statement to open the add scoring record db.

The completed event handler should look like the following example:

```

EVENT ACTION open_add_scoring_record_db
MOVE local_blank_scoring_record TO export_scoring_record
OPEN Dialog Box ADD_SCORING_RECORD_DB
  
```

- c. Save your model.
4. Review the design for the update scoring record secondary dialog. When this dialog opens, we want to see the details of a specific scoring record selected from the list of scoring records on the **eGolfer Home** page.

The logic for this event handler is slightly more complicated than it is for the other event handlers. We need to use a GUI statement to locate the selected scoring record from the group view, then move it into a single entity view to display it to the golfer.

All GUI statements that deal with group views require that the group views be explicitly indexed. We created our views by using the Match and Copy feature to copy the server views into our client procedure. The group views as they existed in the server procedure step were implicitly indexed. As such, they were copied into our client procedure step as implicitly indexed as well. We need to change the group view properties in our client procedure to make them explicitly indexed.

- a. From the Menu Bar, select **View**, then **Expand All Views**. In the IMPORTS, double-click **Group View import group of scoring records**.
 - b. In the Detail Import Group View dialog, change the **and is always** drop-down list from **Implicitly indexed** to **Explicitly indexed** and select the **OK** push button.
 - c. Repeats steps a and b for the **EXPORTS: Group View export group of scoring records**, making them **Explicitly indexed**.
 - d. Now, from the Menu Bar, select **View**, then **Contract All Views**.

- e. Save your model.
- f. Now, we can add the GUI statement. In the action diagram, select the **event handler EVENT ACTION open update scoring record db**. From the Menu Bar select **Edit**, then **Add GUI Statement**, then and **Get Row....** From the list of row types, select **HIGHLIGHTED IN**, from the list of repeating group views select **import group of scoring records**, and from the list of expressions select **1**. Then select the push button **Complete**, select the expression **subscript**, select the repeating **group view import group of scoring records**, select the push button **Complete**, and then select the **Add** push button to add the statement to the **event handler**.

What this statement is essentially doing is sequentially scanning through all of the occurrences in the group view, starting from the first occurrence in the group view. When it finds a highlighted row, it sets the subscript or pointer of the group view to that row number. This will then allow us to move the highlighted occurrence of the import group view to the single entity view in the export view, which will then populate the secondary dialog fields when it opens.

- g. From the Menu Bar select **Edit**, then **Add Statement**, then **Move....** In the list of entity views, select **import group scoring record**, in the list of entity views select **export scoring record**, and then select the **Add** push button to add the statement to the **event handler**. Now, add the GUI statement to **open the update scoring record db**.

The completed event handler should look like the following example:

```
EVENT ACTION open_update_scoring_record_db
GET ROW HIGHLIGHTED IN import_group_of_scoring_records STARTING AT 1 GIVING SUBSCRIPT OF import_group_of_scoring_rec
MOVE import_group scoring_record TO export scoring_record
OPEN Dialog Box UPDATE_SCORING_RECORD_DB
```

- h. Save your model.
5. Review the design for the remove scoring record secondary dialog. The logic for this event handler is the same as the logic for the update scoring record secondary dialog. We want to get the row highlighted in the import group and move it to a single entity view in the export view, then open the secondary dialog.
 - Copy the statements from the **EVENT ACTION open update scoring record db** to the **EVENT ACTION open remove scoring record db**. Change the **OPEN Dialog Box** statement to **open the REMOVE SCORING RECORD DB**.

The completed event handler should look like the following example:

```
EVENT ACTION open_remove_scoring_record_db
GET ROW HIGHLIGHTED IN import_group_of_scoring_records STARTING AT 1 GIVING SUBSCRIPT OF import_group_of_scoring_rec
MOVE import_group scoring_record TO export scoring_record
OPEN Dialog Box REMOVE_SCORING_RECORD_DB
```

Save your model.

6. The last event handler is associated with the logout push button on the **eGolfer Home primary dialog**. When golfers select this push button, we want to transfer them to the eGolf Services home page. If you will recall, we set up transfers between these two procedure steps. The exit state value that will initiate the transfer flow back is RETURN. Add an **EXIT STATE IS return** statement.

The completed event handler should look like the following example:

```
EVENT ACTION egolfer_home_pb_logout_click
EXIT STATE IS return
```

7. Save your model.
8. Close the action diagram.

Update eGolfer

Refer to the example Web pages shown earlier and familiarize yourself with the Update eGolfer Web page. As you can have noticed, it bears a remarkable resemblance to the eGolf Registration dialog. In fact, the two dialogs are so similar; we will simply copy the Registration dialog to the Update dialog, and then make necessary changes.

Follow these steps:

1. In the Navigation Diagram, double-click **eGolf Services Registration** to open the dialog design. We want to copy everything except for the banner at the top. To do this, click-and-drag from the top-left corner directly below the banner to the bottom-right corner of the dialog. This places the rubber band around all the objects on the dialog.
2. From the Menu Bar select **Edit**, and then select **Copy....** Notice that the subsequent Window Copy dialog is divided into two sections. The top has the Source group box giving you information about the window/dialog selected for copy, while the bottom has the Destination group box, which gives you information about the destination window/dialog. Under the Controls: drop-down, the items currently selected for copying are listed.
3. To select a destination dialog, select the **Destination...** push button. In the Window Copy - Destination Selection dialog, select **EGOLFER-HOME Dialog—Box Update eGolfer** and select the **OK** push button. In the Window Copy dialog select **Copy** and select **Yes** to confirm. Make sure you are copying to **UPDATE EGOLFER DB**, then acknowledge the Note by selecting **OK**. Close the Window Copy dialog.
4. In the Navigation Diagram, double-click **Update eGolfer** to open its dialog design. The controls were copied, but the dialog has not been resized. Had we copied the entire dialog, however, the dialog would have been resized.

Use the **eGolf Services Registration** dialog to resize the Update eGolfer dialog. It will work best if you drag the top and right side of the Update eGolfer dialog.
5. Gain focus on the **eGolf Services Registration** dialog design and select the **Close Window** icon to close it.

Edit the Window

Starting at the top, we can begin to make the necessary changes.

Follow these steps:

1. To add the new banner, from the Menu Bar select **Add**, then **Picture....** In the Bitmap: drop-down list, select **adbanner**, in the Name: entry field enter **picture1**, in the HTML extension entry field enter **gif**, and then select the **OK** push button. Position the picture at the top of the dialog.
2. Double-click the literal **eGolfer Information** and add the word **Update** to the beginning of the literal. If necessary, resize the literal box to display the entire literal.
3. Select the **User ID** and **Password...** literal and delete it by selecting the **Delete** icon from the Toolbar. Select the **Yes** push button to confirm.
4. Double-click the **User ID can be up to...** literal and change it to **Your User ID is permanent and cannot be updated.**
5. If necessary, use the **Position...** feature to re-align the field and literals.

Re-map Fields

As part of the copy of the Registration dialog design to the Update eGolfer dialog design, new views of golfer were created in our procedure step action diagram. We want to re-map the fields on the screen to the views of the golfer that existed prior to the copy.

Follow these steps:

1. Double-click the **User ID** field and change the **Export View:** drop-down from **EXPORT_1** to **EXPORT**. Then, change the **Import View:** drop-down from **IMPORT_1** to **IMPORT**.
Note: If you are having trouble with this step, try mapping the views to (none), then going back and mapping them to the appropriate view.
2. This field is read-only. Un-check the **Margin Box** check box and check the **Read Only** check box, then select the **OK** push button.
3. Double-click the **To ensure the security...** literal and add the sentence **Your User ID and Password** are required to login to the system. Resize the literal box as necessary and use the **Position...** feature to re-align the objects if needed.
4. Double-click the **Password** field and re-map the views to **IMPORT** and **EXPORT**, then select the **OK** push button.
5. Double-click the **Confirm Password** field and re-map the views to **IMPORT CONFIRMATION** and **EXPORT CONFIRMATION**, then select the **OK** push button.
6. Double-click the **First Name** field and re-map the views to **IMPORT** and **EXPORT**, then select the **OK** push button.

7. Double-click the **Last Name** field and re-map the views to **IMPORT** and **EXPORT**, then select the **OK** push button.
8. Double-click the **Email Address** field and re-map the views to **IMPORT** and **EXPORT**, then select the **OK** push button.
9. Double-click the **Review...** literal and change continuing to **updating**, then select the **OK** push button.
10. Double-click the **<Back** push button and change the Button Text: to **cancel**, then select the **OK** push button.
11. Double-click the **Next>** push button, change the Button Text: to **update**, ensure that it is the **Default** Push Button, and select the **OK** push button. We will need to change the event that it executes as well, but we will change that a little bit later.
12. Reverse the position of the two push buttons. They should be aligned **horizontally bottom, separated horizontally** by a distance of **20** and moved **horizontally center**.
13. Save your model and close the dialog design.

Edit the Action Diagram

Follow these steps:

1. Open the **EGOLFER HOME** action diagram by selecting the **Action Diagram** icon on the Toolbar. If necessary, first select it in the Navigation Diagram.
2. Eventually, we want to delete the extra views created as part of the copy, but first we need to remove any references to them in the action diagram. In addition to copying the dialog design and the views necessary to support the dialog design, the copy feature also copied the events associated with the copied objects. In this case, the next push button, which is now the update push button, had a click event associated with it. We want to rename the click event to something that makes more sense for this action diagram.

To do this, double-click the **EVENT ACTION egolfer reg pb next click**, and change the Action Name to **update egolfer pb update click**, then select the **OK** push button.
3. Within that same event action, in the IF statement, double-click **import confirmation 1 golfer**. Then in the Change dialog, select character view, select the entity view **import confirmation golfer**, and then select the **Change** push button.
4. Within the same IF statement, double-click **import 1 golfer**. Then, in the Change dialog, select character view, select the **entity view import golfer**, select the attribute **password**, and then select the **Change** push button.

5. Within the same event action, double-click the **command value register**. In the Change dialog, select **command value**, in the Command Selection list select **UPDATE**, select the **OK** push button, and then select the **Change** push button.
6. Within the same event action, in the USE maintain golfer statement, double-click the Entity View **import 1 golfer**. In the MAINTAIN GOLFER Import View Matching dialog, select **IMPORT GOLFER** from the top panel, select **IMPORT GOLFER** from the bottom panel, select the **Match** push button, and then select the **Close** push button.
7. Within the same event action, in the USE maintain golfer statement, double-click the Entity View **export 1 golfer**. In the MAINTAIN GOLFER Export View Matching dialog, select **EXPORT GOLFER** from the top panel, select **EXPORT GOLFER** from the bottom panel, select the **Match** push button, and then select the **Close** push button.
8. Within the same event action, select the **EXIT STATE IS xfr to egolfer home statement**. Then select the **Delete** icon from the Toolbar and select the **Yes** push button to confirm.
9. Save your model.

Delete Extra Views

Follow these steps:

1. From the Menu Bar select **View**, and then select **Expand All Views**.
2. In the IMPORTS:, select Entity View **import 1 golfer**.
3. With the **Ctrl** key held down, select Entity View **import confirmation 1 golfer**.
4. Scroll to the EXPORTS: and, with the **Ctrl** key still held down, select Entity View **export 1 golfer**.
5. With the **Ctrl** key still held down, select Entity View **export confirmation 1 golfer**.
6. From the Menu Bar select **Edit**, then **Delete Views....** Select **Yes** at the confirmation.
7. Save your model.

The completed Event Handler should look like the following example:

```

EVENT ACTION update_egolfer_pb_update_click
[
  IF import_confirmation golfer password IS EQUAL TO import golfer password
  COMMAND IS update
  USE maintain_golfer (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
  WHICH EXPORTS: Entity View export golfer FROM Entity View export golfer
  [
    IF EXITSTATE IS EQUAL TO processing_ok
    CLOSE Dialog Box UPDATE_EGOLFER_DB
  ]
  ELSE
  EXIT STATE IS password_mismatch
]

```

8. Close the action diagram.

Remove eGolfer

See the window examples shown earlier and familiarize yourself with the Remove eGolfer Web page. As you can have noticed, it bears a remarkable resemblance to the Update eGolfer dialog. The primary differences are that all of the fields are read-only and the Password is not displayed. This time, we will copy the entire Update eGolfer dialog to the Remove eGolfer dialog, and then make necessary changes. If you recall, last time we did not copy all of the objects on the dialog.

Follow these steps:

1. In the Navigation Diagram, select the **Update eGolfer** dialog. From the Menu Bar select **Edit**, then select **Copy....** The Controls: drop-down list now indicates All objects.
2. Select the **Destination...** push button, then select **EGOLFER HOME Box Dialog Remove eGolfer**, and then select the **OK** push button. Then select the **Copy** push button, select the **Yes** push button to confirm the copy, and select the **OK** push button to acknowledge the copy. Select the **Close** push button to close the Window Copy dialog.
3. In the Window Navigation diagram, double-click **Remove eGolfer**.

Since we copied the entire dialog, it was resized to match the copied dialog. The last time we only copied some of the objects and subsequently had to size the box manually. You will also see shortly that this time it used the same views as the copied dialog, instead of making up new views for us. This is because we are copying from a dialog in the same procedure step, whereas last time, we were copying a dialog from a different procedure step.

Make Adjustments

Follow these steps:

1. Double-click the first literal, **Update eGolfer Information**, and change it to **Remove eGolfer**.
2. Double-click the second literal, **Your User ID...**, and change it to **Are you sure you want to be completely removed from our database?**
3. Double-click the **User ID** field and verify that the field is correctly mapped to the IMPORT GOLFER USERID and the EXPORT GOLFER USERID.
4. The Password fields and the literals immediately before and after the passwords are not required. Click-and-drag from the left side of the **To ensure...** literal down to the right side of the **All fields are required** literal to create a rubber band effect to select both literals, the two passwords, and their two prompts all at once. Make sure nothing else is selected and then select the **Delete** icon from the Toolbar. Select the **Yes** push button to confirm.

5. Double-click each of the remaining fields and verify that they are correctly mapped to the IMPORT and EXPORT GOLFER. Then un-check the **Margin Box** check mark and check the **Read Only** check box.
6. To position the fields, select the **User ID** field (not the prompt). Then, with the **Ctrl** key held down, select the **First Name** field (once again, the field and not the prompt), the **Last Name** field, and the **Email Address** field. Then, from the Menu Bar, select **Edit** and then **Position....** Next, select **Align Vertically Left**, select **Separate Vertically** by a Distance: of **5**, select **Arrange prompts against the fields**, select **Move Vertically Center**, and then select the **OK** push button.
7. Select the **Review...** literal and delete it.
8. Double-click the **update** push button and change the Button Text: to **delete**. Check the **Is the Default Push Button** check box. Then select the **Events...** push button.
9. Since we copied this dialog from the **Update eGolfer** dialog, it is still executing the update event. We want to disassociate the update event from this push button and add a new event to contain the delete logic.

To do this, in the Defined Events panel, select Click **UPDATE EGOLFER PB UPDATE CLICK**, and then select the **Remove** push button. Notice that while this action removes the update event from the Defined Event list, it is still in the Event Details Action Name: list. Change the name to **REMOVE EGOLFER PB DELETE CLICK** and then select the **Add** push button to create and associate this new event to the push button. Select the **Close** push button, and then select the **OK** push button.
10. With focus still on the delete push button, hold the **Ctrl** key down and select the cancel push button. Then right-click-and-drag them to roughly the position shown in the sample example earlier. Then, with focus still on both of them, from the Menu Bar select **Edit**, then **Position....**, then **Move Horizontally Center**, and then select the **OK** push button.
11. Save your model.

Edit the Action Diagram

Follow these steps:

1. Open the action diagram by selecting the **Action Diagram** icon on the Toolbar. The new EVENT ACTION remove egolfer pb delete click has been added to the bottom of the action diagram. The event logic will be very similar to the Update eGolfer logic added from the last section.
2. Copy the logic in the **EVENT ACTION update egolfer pb update click**, starting with the **COMMAND IS** statement and ending with the bottom of the brackets associated with the **IF EXITSTATE IS EQUAL TO** statement to the **Remove eGolfer** event.
3. Change the **COMMAND IS update** statement to **COMMAND IS delete**.
4. In the **IF EXITSTATE IS processing ok** statement, delete **CLOSE Dialog Box UPDATE GOLFER DB** and add the statement **EXIT STATE IS return** in its place.

5. Save your model.

The completed event handler should look like the following example:

```
EVENT ACTION remove_egolfer_pb_delete_click
COMMAND IS delete
USE maintain_golfer (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
  WHICH EXPORTS: Entity View export golfer FROM Entity View export golfer
  IF EXITSTATE IS EQUAL TO processing_ok
  EXIT STATE IS return
```

6. Close the action diagram.
7. Close the dialog.

Add Scoring Record

See the window examples shown earlier and familiarize yourself with the Add Scoring Record Web page. Most of the skills that we need to build this dialog have already been discussed, so the steps to complete this dialog are going to be more abbreviated than the steps for the other dialogs. See prior examples if you need help in accomplishing a particular step.

Follow these steps:

1. Open the **Add Scoring Record** dialog design.
2. Resize the **Add Scoring Record dialog** such that it is similar in size to other dialogs we have designed.
3. Add the **adbanner** picture to the top of the dialog design.
4. Add the literal **Add Scoring Record**, making the font size **Arial/Bold/16**.
5. Add the literal **Enter the following scoring information about your game of golf**. Use the default font size.
6. The next step is to prepare the Scoring Record fields for their addition to the dialog design.
 - a. From the Menu Bar select **Edit**, then select **Field Design....** In the Field Design dialog, scroll to the EXPORT SCORING RECORD fields.
 - b. If necessary, arrange the attributes in the entity view to be in the order shown on the window example shown earlier. You can accomplish this by selecting the attribute you want to move, selecting the **Move** push button, and then selecting the attribute you want to move your selected attribute directly below.

Note: The Move push button will be disabled if you have more than one attribute highlighted. To un-select a selected attribute, just select it again.

- c. Once you have the attributes in order, select each attribute again, and detail its properties. You can accomplish this by selecting the attribute and then selecting the **Properties...** push button.

Note: The Properties... push button will be disabled if you have more than one attribute highlighted.

For each attribute, ensure the following:

- The Import View mapping is IMPORT
 - The Export View mapping is EXPORT
- d. Enter the appropriate Prompt: for each field as shown:
 - Date—Date Played:
 - Time—Time Played:
 - Adjusted Gross Score—Adjusted Gross Score:
 - Course Rating—Course Rating:
 - Course Slope Rating—Course Slope Rating:
 - Note—Note:
 - e. Enter the appropriate Edit Pattern: for each field as shown:
 - Date—MMDDYY
 - Time—HHMM
 - Adjusted Gross Score—ZZZ
 - Course Rating—ZZ.Z
 - Course Slope Rating—ZZZ
 - Note—<Default Edit Pattern>

Also, ensure that:

- Each field has Margin Box checked
- No field has Read Only checked
- The Note field has Autscroll checked

7. In the Field Design dialog, select all six fields and then select the **Place** push button. They will be presented to you, one at a time, in the order in which they appear in the Field Design panel. For each field, you will be presented with the prompt first, followed by the field. Place each one on the dialog design in roughly the position indicated from the example shown earlier. Verify that all of the edit patterns are as specified and if not, double-click the field and re-enter the edit pattern.

Positioning

Follow these steps:

1. You can have noticed that the **Note** field extended off the right side of the dialog.
To fix this, select the **Note** field. Then, with the cursor, drag the left edge of the field to the right several inches. Then right-click-and-drag the **Note** field back onto the dialog. If necessary, continue to resize the left size and drag the field onto the dialog until you can see the right edge of the field.
2. Using the cursor, stretch each prompt so that you can see the entire prompt.
3. To align everything left, first select the **Add Scoring Record** literal, and then using the **Ctrl** key, select the **Enter...** literal and each of the prompts. Use the Position feature to **Align Vertically Left**.
4. Now select just the prompts starting with **Date Played:** and, using the positioning feature, **Separate them Vertically** by a Distance of **5**.
5. Now select the longest prompt, which is **Adjusted Gross Score:**. Then, with the **Ctrl** key held down, select its field. Using the Position feature, make them of **Equal Height, Align them Horizontally Middle**, and **Separate them Horizontally** by a Distance of **5**.
6. Finally, select just the **Adjusted Gross Score** field, then with the **Ctrl** key held down, select each of the other fields. Then use the Position feature to make them of **Equal Height, Align them Vertically Left**, and **Separate them Vertically** by a Distance of **5**.
7. Now use the cursor to rubber band every object on the dialog design (except for the **adbanner**) and use the Position feature to **Move** the entire group **Horizontally Center**.
8. Add the literal **(MMDDYY)** to the right of the Date Played field.
9. Add the literal **(HHMM)** to the right of the Time Played field.
10. Add the literal **Review your information before continuing. Thank you!**. Use the default font size but make it bold.
11. Add the **add** push button and allow the Toolset to make up a click event for it. Make it the **Default Push Button**.
12. Add the **cancel** push button and have it execute the special action to **Cancel – Close without Execution**.
13. Resize the push buttons and **Align them Horizontally Center, Separated Horizontally** by a Distance of **20**, and **Move Horizontally Center**.
14. Disable the **add** push button if any of the fields (except **Note**) does not have data.
15. Save your model.

Edit the Action Diagram

Follow these steps:

1. Open the action diagram. Locate the event handler the Toolset created for us. Its name should be **EVENT ACTION add scoring pb add click**.
2. Move, do not copy, the two procedure step USE statements added to create views into the **EVENT ACTION add scoring pb add click** event handler. To move both of them, using the **Ctrl** key, select each **USE** statement, press **F7** on your keyboard, and with the cursor (which now resembles a hand), scroll down (if necessary) and select **EVENT ACTION add scoring pb add click**.
3. This event is also going to need to use the ListScoringRecords server procedure step. We created an open event that uses the List Scoring Records server procedure step. Locate the **EVENT ACTION egolfer home** open and copy (do not move) both of its statements into our **Add Scoring Record** event. To copy them, select the statements, press **F8** on your keyboard, and with the cursor (which now resembles a hand), scroll down (if necessary) and select **EVENT ACTION add scoring pb add click**.
4. Rearrange or copy the statements in the **EVENT ACTION add scoring pb add click** event handler to produce the following logic. You will need to add an **IF** statement in addition to the GUI **CLOSE** statement. The completed event handler should look like the example:

```

EVENT ACTION add_scoring_pb_add_click
COMMAND IS create
USE maintain_scoring_record (procedure step)
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
    Entity View import scoring_record TO Entity View import scoring_record
IF EXITSTATE IS EQUAL TO processing_ok
COMMAND IS handicap
USE maintain_golfer (procedure step)
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
    WHICH EXPORTS: Entity View export golfer FROM Entity View export golfer
COMMAND IS list
USE list_scoring_record (procedure step)
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
    WHICH EXPORTS: Group View export_group_of_scoring_records FROM Group View export_group_of_scoring_records
CLOSE Dialog Box ADD_SCORING_RECORD_DB

```

5. Save your model.

Update Scoring Record

See the window examples shown earlier and familiarize yourself with the Update Scoring Record Web page. Most of the skills that we need to build this dialog have already been discussed, so the steps to complete this dialog are going to be more abbreviated than the steps for the other dialogs. See prior examples if you need help in accomplishing a particular step.

Follow these steps:

1. Copy the **Add Scoring Record** dialog design to the **Update Scoring Record** dialog.
2. Change the literal **Add Scoring Record** to **Update Scoring Record**.

3. Change the literal **Enter the following...** to **Please update the following...** and move it between the Time Played and the Adjusted Gross Score.
4. Add a new literal that states, **Please Note that the Date and Time Played cannot be updated.**
5. Delete the literal **(MMDDYY)**.
6. Delete the literal **(HHMM)**.
7. In the Date Played field properties, un-check the **Margin Box** and make the field **Read Only**. Then add the **MM/DD/YY** Edit Pattern.
8. In the Time Played field properties, un-check the **Margin Box** and make the field **Read Only**. Then add an **HH:MM** Edit Pattern.
9. Change the add Button Text to **update**, remove the **Click event ADD SCORING PB ADD CLICK**, delete the **ADD SCORING PB ADD CLICK** event name from the Action Name: entry field and allow the Toolset to add a new click event for this push button. Make it the **Default Push Button**.
10. Locate the new event handler in the action diagram. Copy all of the statements from the **EVENT ACTION add scoring pb add click** event handler into the new event handler. In the new event handler, change the **COMMAND IS create** statement to **COMMAND IS update**.
11. Change the **CLOSE Dialog Box ADD SCORING RECORD DB** to **CLOSE Dialog Box UPDATE SCORING RECORD DB**.

The completed event handler should look like the example:

```

EVENT ACTION update_scor_pb_update_click
COMMAND IS update
USE maintain_scoring_record (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
               Entity View import scoring_record TO Entity View import scoring_record
  IF EXITSTATE IS EQUAL TO processing_ok
  COMMAND IS handicap
  USE maintain_golfer (procedure step)
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
                 WHICH EXPORTS: Entity View export golfer FROM Entity View export golfer
  COMMAND IS list
  USE list_scoring_record (procedure step)
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
                 WHICH EXPORTS: Group View export_group_of_scoring_records FROM Group View export_group_of_scoring_records
  CLOSE Dialog Box UPDATE_SCORING_RECORD_DB

```

12. Save your model.

Remove Scoring Record

See the example Web pages shown earlier and familiarize yourself with the Remove Scoring Record Web page. Most of the skills that we need to build this dialog have already been discussed, so the steps to complete this dialog are going to be more abbreviated than the steps for the other dialogs. See prior examples if you need help in accomplishing a particular step.

Follow these steps:

1. Copy the **Update Scoring Record** dialog design to the **Remove Scoring Record** dialog.
2. Change the literal **Update Scoring Record** to **Remove Scoring Record**.
3. Change the literal **Please note that the...** to **Are you sure you would like to remove this Scoring Record?**
4. Delete the literal **Please update the following information....**
5. In the properties panel of the following fields, ensure that the **Margin Box** check box is un-checked and that the **Read Only** check box is checked:
 - Adjusted Gross Score
 - Course Rating
 - Course Slope Rating
 - Note
6. Delete the literal **Review your information before....**
7. Change the **update** Button Text to be remove, ensure it **Is the Default Push Button**, remove the **Click event ADD SCORING PB ADD CLICK**, delete the **ADD SCORING PB ADD CLICK** event name from the Action Name: entry field and allow the Toolset to add a new click event for this push button.
8. Re-Position the fields as necessary.
9. Locate the new event handler in the action diagram. Copy all of the statements from the **EVENT ACTION update scor pb add click** event handler into the new event handler. In the new event handler, change the **COMMAND IS update** statement to **COMMAND IS delete**.

10. Change the **CLOSE Dialog Box UPDATE SCORING RECORD DB** to **CLOSE Dialog Box REMOVE SCORING RECORD DB**.

The completed event handler should look like the following example:

```
EVENT ACTION remove_scor_pb_remove_click
COMMAND IS delete
USE maintain_scoring_record (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
  Entity View import scoring_record TO Entity View import scoring_record
  IF EXITSTATE IS EQUAL TO processing_ok
  COMMAND IS handicap
  USE maintain_golfer (procedure step)
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
    WHICH EXPORTS: Entity View export golfer FROM Entity View export golfer
  COMMAND IS list
  USE list_scoring_record (procedure step)
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
    WHICH EXPORTS: Group View export_group_of_scoring_records FROM Group View export_group_of_scoring_records
  CLOSE Dialog Box REMOVE_SCORING_RECORD_DB
```

11. Save your model.

Chapter 5: Construction and Test

This section contains the following topics:

[Objectives and Time Allotment](#) (see page 263)

[Generating the Database](#) (see page 264)

[Referential Integrity Triggers](#) (see page 277)

[Packaging Procedures into Load Modules](#) (see page 281)

[Generating the Application](#) (see page 284)

[Testing the Application](#) (see page 293)

[Web Generation](#) (see page 304)

Objectives and Time Allotment

After this module, you will be familiar with:

- Database Generation
- Referential Integrity Triggers
- Load Module Packaging
- Application Generation
- Application Test
- Web Generation and Deployment

Note: Successful completion of this entire section of the *Tutorial* depends on your having a properly configured CA Gen Workstation, Build Tool, and Runtime environment, as well as a properly configured and supported DBMS, SQL pre-compiler, source language compiler, Web Server, Web Application Server, Java Runtime Environment, and Java Compiler. Instructions for setting each of these up are outside the scope of this Tutorial. However, most of this section can be completed just having a properly configured GUI environment. Just be aware that some of the instructions will be for a particular DBMS or Web Server, and will have to be modified for your unique environment.

Note: The instructions for Generation within this section apply only to workstation construction.

Generation is a separately purchasable item within CA Gen and your workstation configuration might not support it. In many cases, models (or subsets) must be uploaded to an Encyclopedia where generation would take place. While the actual mechanics of generating a model from an Encyclopedia differ from those of generating the model from the workstation, the concepts involved in each remain the same.

Allow yourself approximately 3 hours and 10 minutes to complete this module.

If you would like to reference a CA Gen model reflecting the completed objectives of this chapter, you can find the model with the CA Gen installation in the \Samples\Models\Tutorial Models\Completed Models directory corresponding with this chapter.

Generation and Test

At this point in a typical development project you would have a very large, nicely bound Requirements Specification document and probably another equally nice Detailed Design document. You would then turn these over to a horde of programmers who would start writing the source code necessary to meet the Design. In addition, with any luck at all, it would be months until the written code adhered to the Detailed Design document and the Detailed Design fully encompassed the stated Requirements.

At this same point in an CA Gen project, however, we are almost done. In addition, since the Procedures have implemented the Processes, and the Processes contain our Business Requirements, you can be assured that the generated code will adhere to those Requirements.

All we have to do now is push the big red generate button.

100 Percent Code Generation

When we push that button, CA Gen generates 100 percent of the code necessary to run the application in a targeted (and supported) environment from our environment-independent diagrams.

Generating the Database

The following sections provide detailed information on generating the database.

Lesson Objectives and Time Allotment

After this lesson you will understand:

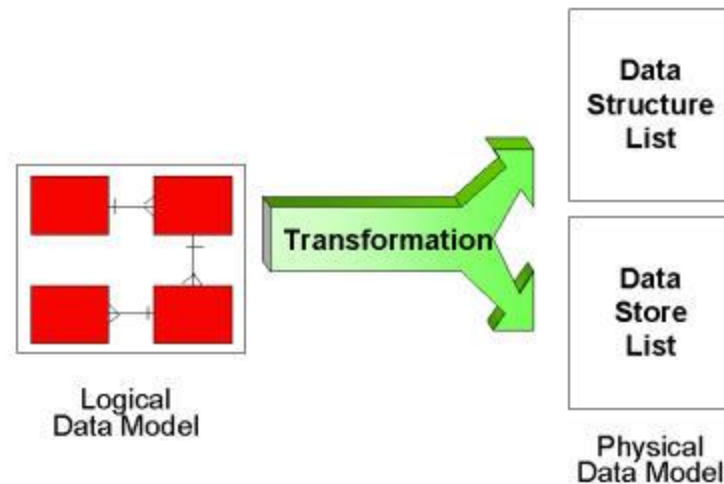
- How to transform a logical data model into a physical data model
- How to set the generation defaults
- How to generate and install a local database

Allow yourself approximately 40 minutes to complete this lesson.

Database Generation

Database generation consists of two steps and is normally done by a Database Administrator.

1. Transform your logical data model into a physical data model.
2. Generate the necessary DDL (Data Definition Language) from the physical data model for the DBMS software to create the database.



The Data Model (or Entity Relationship Diagram) is often known as the Logical Data Model. For the most part, it is free of implementation-specific detail. To create a physical database, we have to first transform our logical Data Model objects into physical Data Model objects. The physical Data Model is actually represented in two diagrams, the Data Structure List diagram, and the Data Store List diagram. Collectively these two diagrams are sometimes known as the Technical Design.

In the Data Structure List diagram, you make implementation-specific structure changes and decisions such as:

- Reordering columns of a table
- Making indexes ascending or descending
- De-normalizing data between tables

In the Data Store List diagram, you make implementation-specific space allocation changes and decisions such as:

- Adjusting tablespace sizes
- Adjusting indexspace sizes

Basic Data Model Transformation

During the Transformation process, there is a set of basic rules applied to create the Data Structure List:

- Entity Types become Tables
- Attributes become Columns within the Tables
- Identifiers become Unique Indexes
- Relationships become Foreign Key Columns in one of the Tables involved in the relationship or the other with Indexes and RI (Referential Integrity) Constraints created on them

Additionally, the Transformation process calculates tablespaces and indexspaces to create the Data Store List.

Once the Transformation process has completed, you would then make any implementation-specific changes required to these two diagrams prior to generating the DDL and DML.

DDL Generation

Once you have completed your customizations, you can then push that big red button to generate the Data Definition Language that will ultimately get passed to the DBMS to create your database.

In addition to generating the DDL, you can choose to install the DDL as well. For a local installation, the install option will automatically pass the generated DDL to the DBMS, which will then create the database. If the database is on a remote platform, the installation will create a remote file that can be transferred to the remote location for installation.

Lesson Activity



In this exercise, we are going to:


- Select a supported database for the Technical Design Defaults
- Transform the Data Model
- Briefly review the Data Structure and Data Store Lists
- Set the Generation Defaults
- Generate and Install the DDL

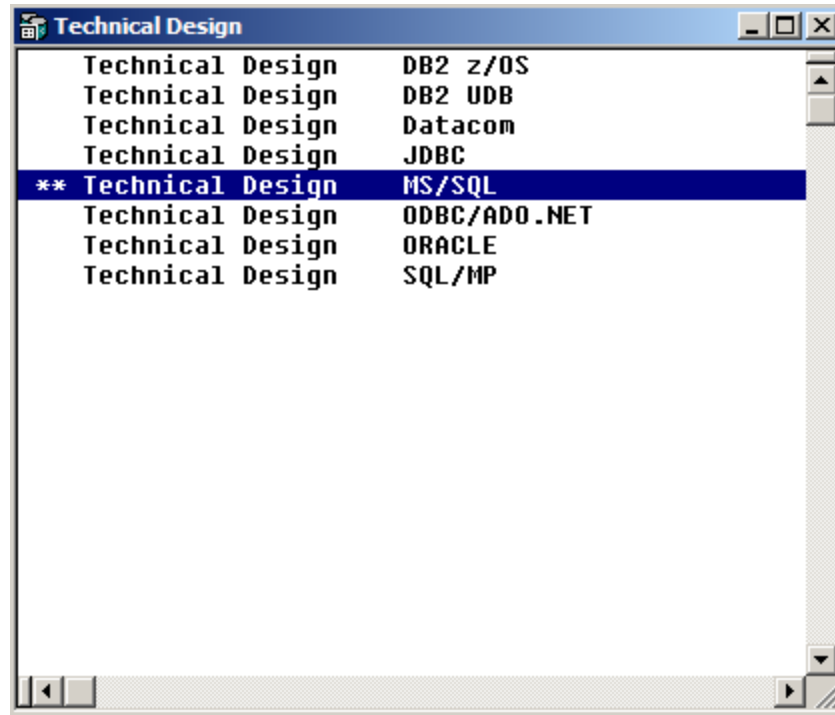
Select the Defaults for Technical Design

Follow these steps:

1. If necessary, open the eGolf Services model.
2. In the tree control, scroll down to the Technical Design folder and double-click **Technical Defaults**.

3. In the list of Technical Designs, select your target DBMS. In the example shown in step 4, the target DBMS is Microsoft's SQL Server (MS/SQL).

4. From the Toolbar, select the **Open...**  icon to activate the selected Technical Design. The two asterisks next to the selected Technical Design indicate that the design has been activated.



5. If you would like to review the default Technical Design Properties, double-click the selected Technical Design. These settings influence the DDL and DML (Data Manipulation Language) that subsequently are generated. Review the Help for information on each control.

When you are finished reviewing the properties, select the Cancel push button so as not to make any changes.

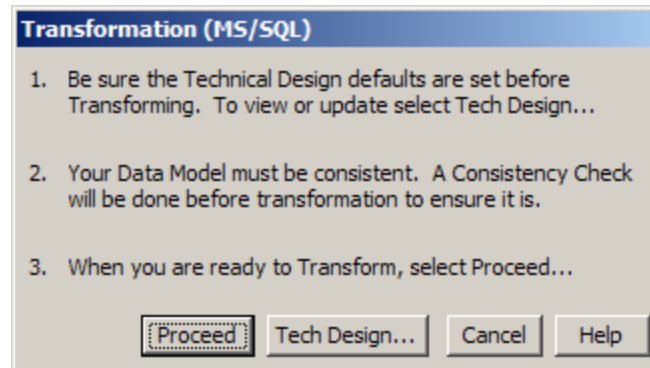
6. Close the list of Technical Designs.
7. Save your model.

Transform the Data Model

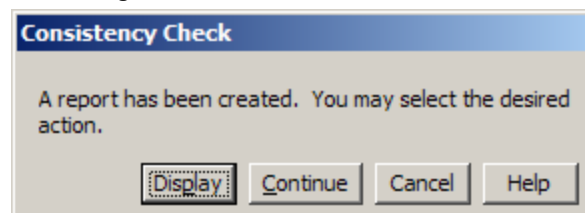
Follow these steps:

1. In the Tree Control, scroll down to the Technical Design folder and start the Transformation Wizard by double-clicking **Transformation** in the Tree Control. The Transformation Wizard dialog will open.

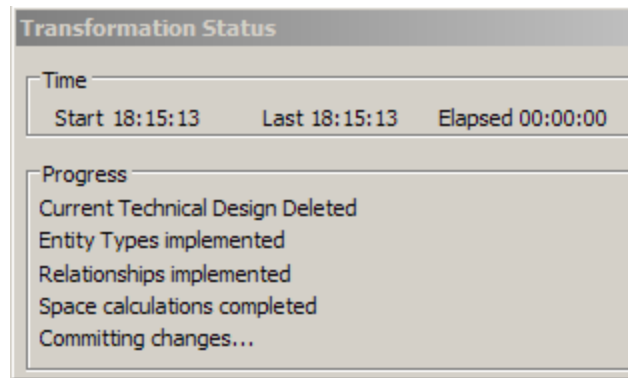
Note: The activated DBMS Technical Design (in this case MS/SQL) is indicated in the Title Bar.



2. We have already set the Technical Design defaults. Our Data Model should still be consistent. Select the **Proceed** push button to initiate the transformation.
3. If a Consistency Check dialog pops up as shown below, then that indicates that your Data Model has Errors or Warnings. If the Continue button is disabled, then you have Errors that must be corrected before Transformation can take place. If you only have Warnings then you can continue the Transformation Wizard, but you should strive to understand the Warnings so as to determine their impact on your database generation.



4. If you have a clean Consistency Check or **Continue** the Transformation with only Warnings, the Transformation should complete successfully. Notice from the messages in the Transformation Status dialog that the current Technical Design (if any) is deleted and replaced with a new Technical Design with no customizations. Therefore, had you previously done a Transformation and customized the Data Structure List and the Data Store List, transforming the Data Model again to pick up changes would remove those customizations. To preserve changes to the two diagrams after an initial transformation, use Retransformation.



5. Click **Ok** on the Informational dialog indicating the successful Transformation.

Review the Data Structure and Data Store Lists

Follow these steps:

1. In the Tree Control under the Technical Design folder, double-click the **Data Structure List**. Then from the Main Menu, select **View**, and then select **Expand Diagram**.

Notice the column names for Adjusted Gross Score and Course Slope Rating. Make no changes to the diagram. When you are finished reviewing the diagram, close it.

MS/SQL Data Structure List						
Type	Name	Format	Length	Optionality	SEQ	
Table	GOLFER					
Column	PASSWORD	Char	8	Not Null		
Column	FIRST_NAME	Char	30	Not Null		
Column	LAST_NAME	Char	30	Not Null		
Column	EMAIL_ADDRESS	Varchar	100	Null		
Column	HANDICAP_INDEX	Decimal	4,1	Null		
Column	USERID	Char	8	Not Null		
Index (U)	golferid (Primary)					
Column	USERID	Char	8	Not Null	Asc	
Table	SCORING_RECORD					
Column	DATE0	Datetime	8	Not Null		
Column	TIME0	Datetime	6	Not Null		
Column	ADJUSTED_GROSS_SCORE	Smallint	3	Not Null		
Column	COURSE_RATING	Decimal	3,1	Not Null		
Column	COURSE_SLOPE_RATING	Smallint	3	Not Null		
Column	NOTE	Varchar	100	Null		
FK Column	FK_GOLFERUSERID	Char	8	Not Null		
RI Constraint	<No Name> GOLFER					
Index (U)	scoreid (Primary)					
Column	FK_GOLFERUSERID	Char	8	Not Null	Asc	
Column	DATE0	Datetime	8	Not Null	Asc	
Column	TIME0	Datetime	6	Not Null	Asc	

2. In the Tree Control under the Technical Design folder, double-click the **Data Store List**. Then from the Main Menu, select **View**, and then select **Expand Diagram**. Make no changes to the diagram. When you are finished reviewing the diagram, close it.



The screenshot shows a window titled "MS/SQL Data Store List". It contains a tree control on the left and a list of objects on the right. The tree control has a root node "IEFDB" which is expanded to show "Data File", "Log File", "GOLFER", and "SCORING_RECORD". The list of objects has columns for "Type", "Name", "Filegroup", and "FillFactor".

Type	Name	Filegroup	FillFactor
Database	IEFDB		
	Data File		
	Log File		
Table	GOLFER		
Index	golferid		0
Table	SCORING_RECORD		
Index	scoreid		0

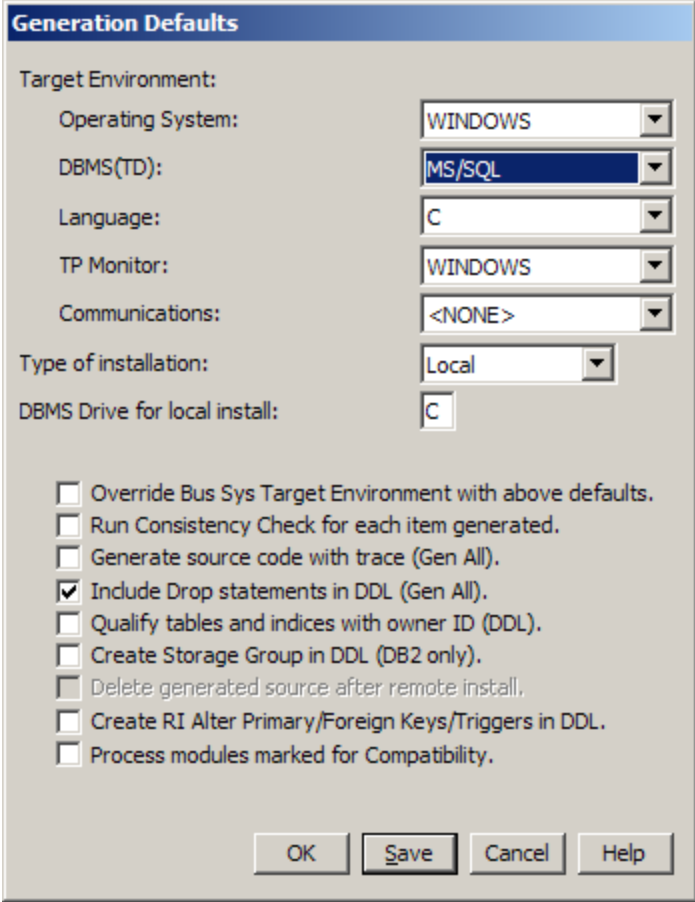
Set the Generation Defaults

Follow these steps:

1. In the Tree Control, scroll down to the Construction folder and double-click **Generation**.
2. From the Main Menu, select **Options** and then select **Generation Defaults....**

3. Set the Generation Defaults as indicated below. Select the **Save** push button to save these settings for the next time you open the Toolset, and then select the OK push button.

Note: Your DBMS(TD) may differ if you are using some other database rather than SQL Server (MS/SQL). The rest of the settings should be the same.




The image shows a 'Generation Defaults' dialog box with a blue title bar. It contains several settings for database generation. The 'Target Environment' section includes dropdown menus for Operating System (WINDOWS), DBMS(TD) (MS/SQL), Language (C), TP Monitor (WINDOWS), and Communications (<NONE>). Below this is a dropdown for 'Type of installation' (Local) and a text field for 'DBMS Drive for local install' (C). A list of checkboxes follows, with 'Include Drop statements in DDL (Gen All)' checked. At the bottom are four buttons: OK, Save, Cancel, and Help.

Generation Defaults	
Target Environment:	
Operating System:	WINDOWS
DBMS(TD):	MS/SQL
Language:	C
TP Monitor:	WINDOWS
Communications:	<NONE>
Type of installation:	Local
DBMS Drive for local install:	C
<input type="checkbox"/> Override Bus Sys Target Environment with above defaults.	
<input type="checkbox"/> Run Consistency Check for each item generated.	
<input type="checkbox"/> Generate source code with trace (Gen All).	
<input checked="" type="checkbox"/> Include Drop statements in DDL (Gen All).	
<input type="checkbox"/> Qualify tables and indices with owner ID (DDL).	
<input type="checkbox"/> Create Storage Group in DDL (DB2 only).	
<input type="checkbox"/> Delete generated source after remote install.	
<input type="checkbox"/> Create RI Alter Primary/Foreign Keys/Triggers in DDL.	
<input type="checkbox"/> Process modules marked for Compatibility.	
OK Save Cancel Help	

Generate and Install the Database

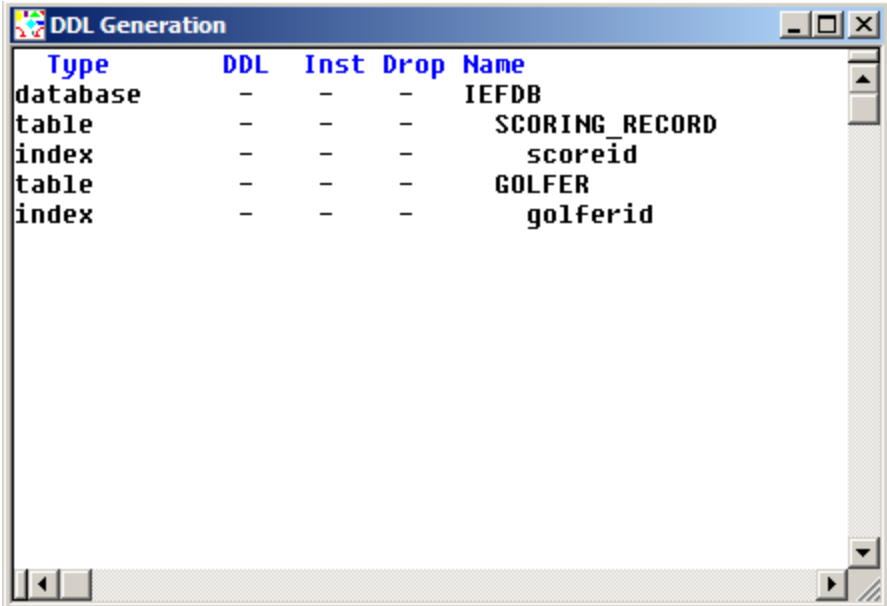
Follow these steps:

1. With the Generation panel still open, select the **DDL**  icon from the Toolbar.

Notice in the DDL Generation diagram the five columns:


- Type
- DDL
- Inst
- Drop
- Name

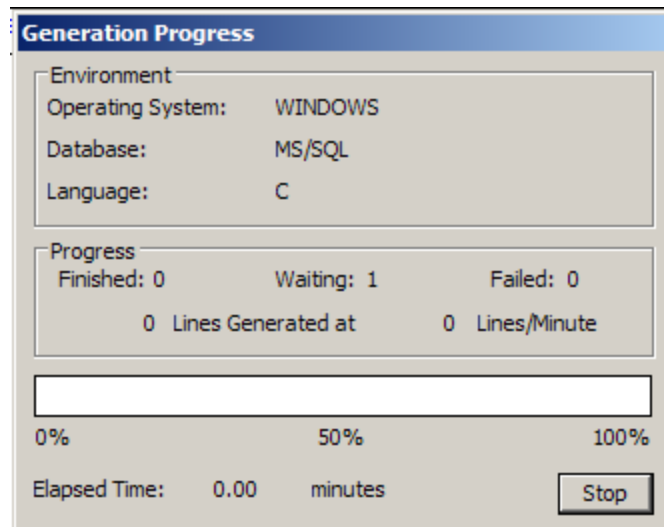
The Type column describes the type of database object you are referring to, while the Name column gives the name of that database object. For example, SCORING_RECORD is a database table, while scoreid is an index on that table. The database itself is named IEFDB.



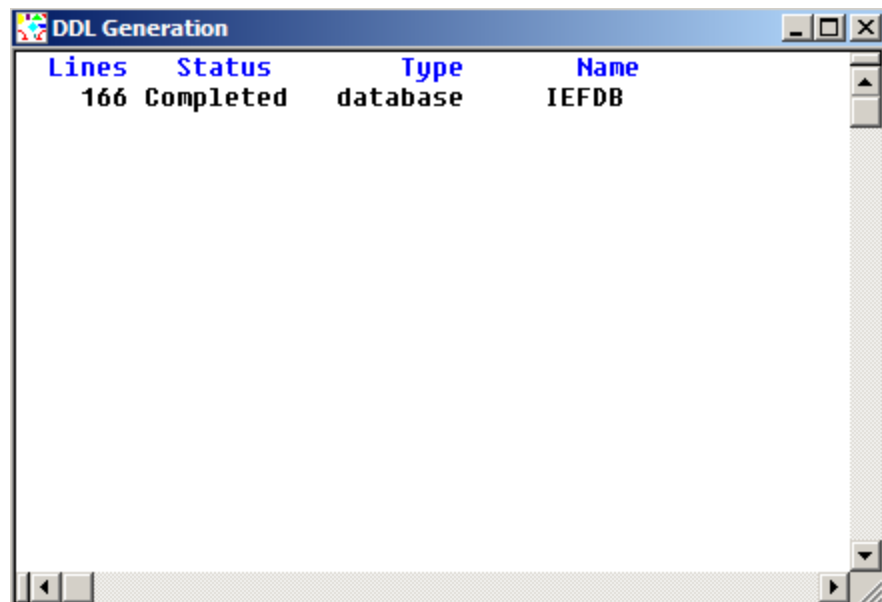
Type	DDL	Inst	Drop	Name
database	-	-	-	IEFDB
table	-	-	-	SCORING_RECORD
index	-	-	-	scoreid
table	-	-	-	GOLFER
index	-	-	-	golferid

We can select individual database objects for generation by selecting the dash (–) under the DDL column. Selecting the dash will replace it with a Y. However, selecting just the dash under the DDL column just causes the DDL to be generated, and not installed. To have it install as well, or to have a remote file created for installing on a different machine, you need also to select the corresponding dash under the Inst column. Finally, if we wanted a drop statement to be added to the generated DDL to first drop an existing database object, we would also need to select the corresponding dash under the Drop column. After making the appropriate selections, you would then select **Generate** from the Main Menu and then select **DDL, Selected**.

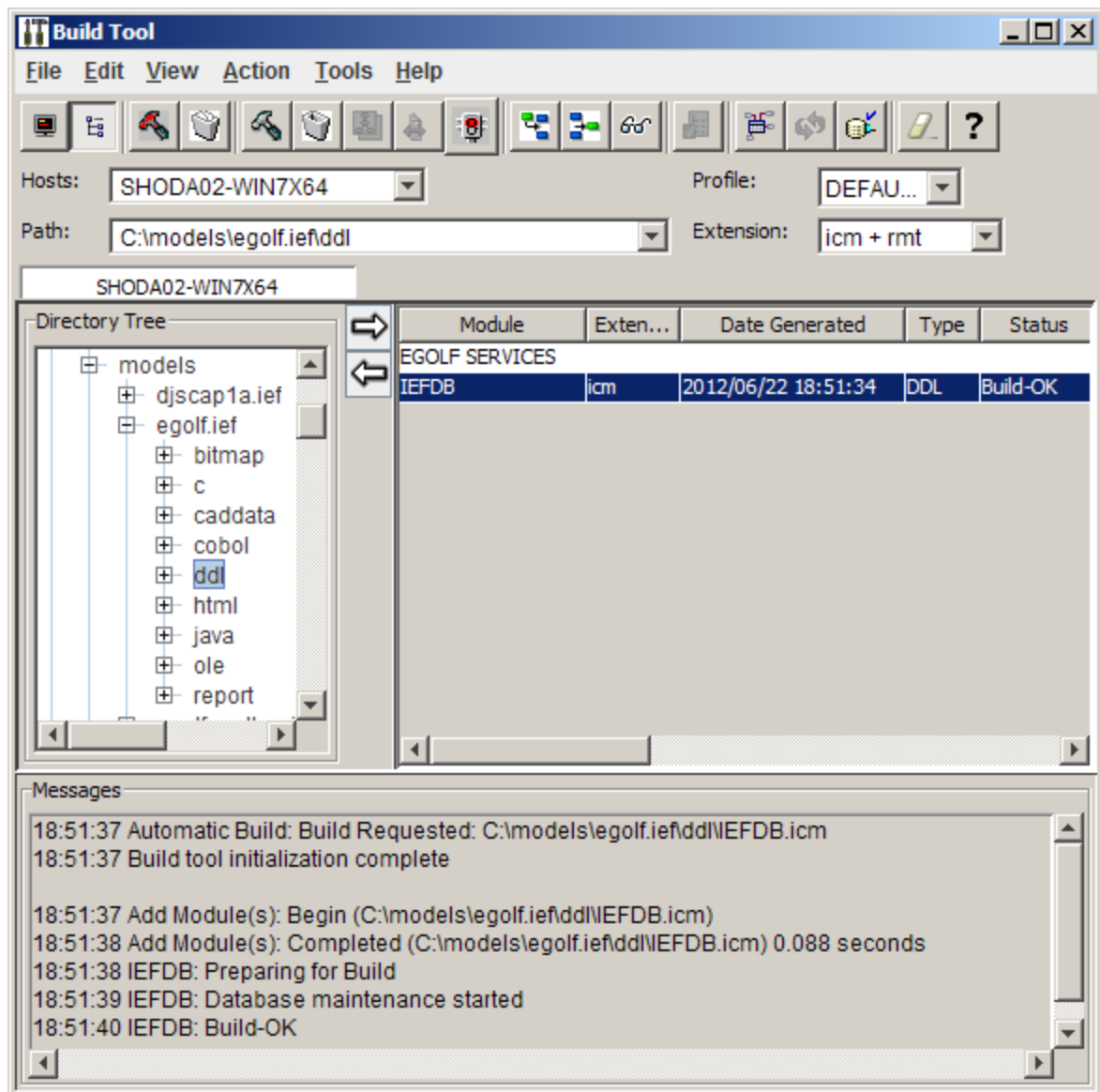
2. In our particular case, we want to generate and install all of the DDL. To do this, simply select the [generate] **DDL, All and Install**  icon from the Toolbar. The Generation Progress dialog shown below opens, and will close automatically if the DDL generation is successful.



If the generation is successful, in the DDL Generation diagram you will see that the list of database objects to be generated has been replaced with a count of the total number of lines of DDL generated. The number of lines generated can vary, depending on your settings.



The DDL generation happens in foreground, which means that you cannot use the Toolset while the generation is taking place. The installation happens in background by starting up a separate program called the Build Tool. If everything works successfully, the Build Tool displays the Build-OK status message as shown below.



3. You can review the results of the installation by selecting the database name in the Build Tool as shown above and then selecting the **Review** icon on the Build Tool toolbar.

Referential Integrity Triggers

The following sections deal with referential integrity triggers.

Lesson Objectives and Time Allotment

After this lesson you will understand:

- The purpose of RI Triggers
- How to generate the RI Triggers

Allow yourself approximately 15 minutes to complete this lesson.

Referential Integrity

Referential Integrity (RI) is a feature of relational databases that helps ensure the consistency of data related between two tables. For the most part, it is concerned with the effects of an Insert or a Delete of a row of data into one table, and how that action should influence (or be influenced by) its relationship to data in another table.

Referential Integrity is generally best left to the DBMS to support, simply because the DBMS can usually perform the RI more efficiently. However, every DBMS does not necessarily support every form of Referential Integrity. With CA Gen, you can supplement the inherent RI capabilities of your targeted DBMS with additional RI Constraints (not normally supported by that database) through the use of CA Gen-generated RI Triggers.

For example, suppose that you wanted to automatically delete all Dependents for an Employee when the Employee is deleted. This is commonly known as a Cascade Delete. While most DBMSs support Cascade Deletes, CA Gen can generate some additional code for you to automatically take care of this constraint if your targeted DBMS does not support it.

Another example is the cyclical Cascade Delete: deleting one Employee who manages another Employee who manages still another. Many DBMSs would not support deleting all three. CA Gen RI Triggers can extend the capabilities of your targeted DBMS to support this constraint as well. The CA Gen RI Triggers are essentially subroutines that are generated to enhance or extend the DBMS' native RI capabilities.

CA Gen provides RI coverage that is both fast and complete. It does this through a combination of generated RI Constraints, Triggers, Stored Procedures and embedded SQL, depending on the particular DBMS selected.

Lesson Activity




Regardless of whether or not you have chosen to have the Referential Integrity maintained by the DBMS (the default), or to have CA Gen maintain all of the Referential Integrity through the use of Triggers, or a combination of the two, the Referential Integrity library must be created and made available to the rest of the CA Gen generated application. These Referential Integrity triggers are commonly referred to by various names, including Triggers, Trigger Library (or Lib), RI Triggers, Cascade Triggers, and Cascade Lib, among others.

In this exercise, we are going to:

- Generate the Referential Integrity Triggers (Cascade.dll)

Generate the Referential Integrity Triggers

Follow these steps:

1. With the Generation diagram still open, select the **Window Code generation**  icon from the toolbar.

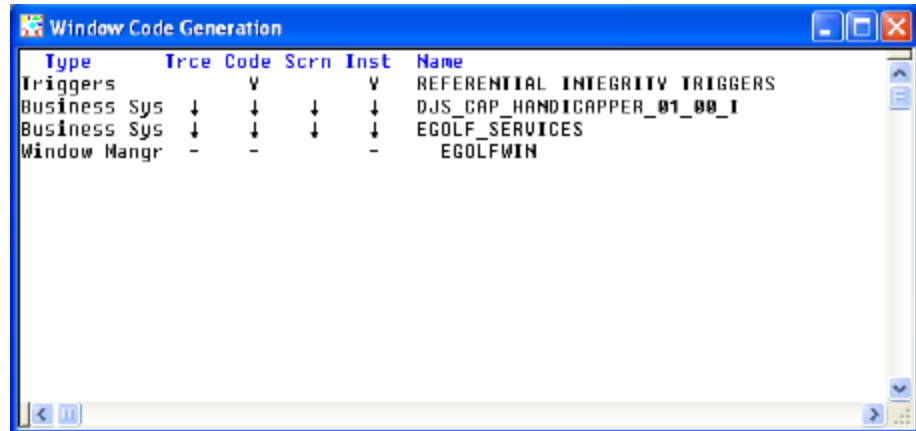
Notice in the Window Code Generation diagram the six columns:

- Type
- Trce
- Code
- Scrn
- Inst
- Name

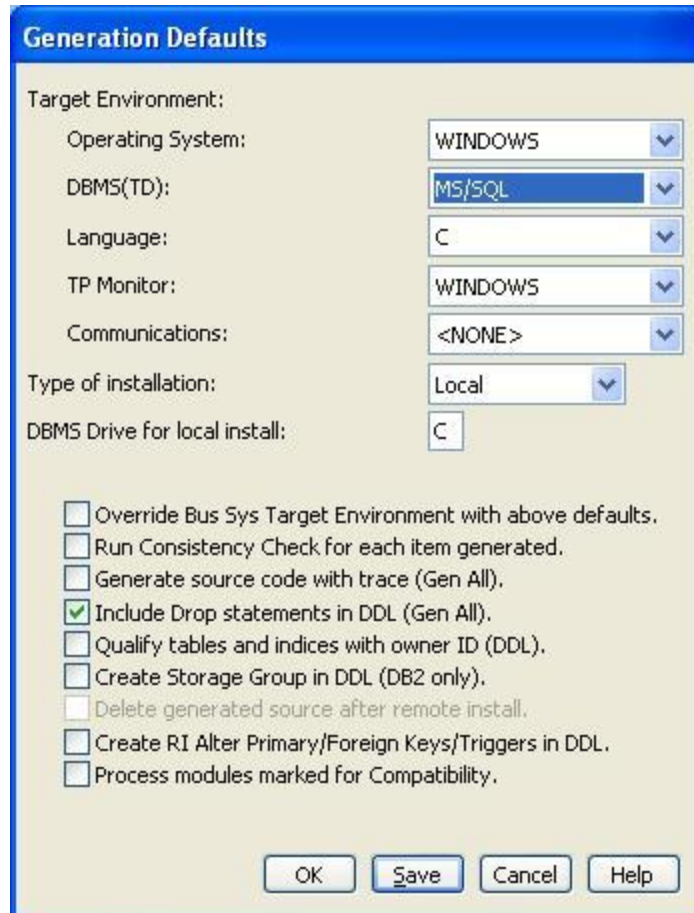
The Type column describes the type of object you are referring to, while the Name column gives the name of that object. For example, REFERENTIAL INTEGRITY TRIGGERS is a Triggers type object. These will make a little more sense when we get into the actual application code generation. As for the other columns, Trce is an abbreviation for trace, Code means source code, Scrn is an abbreviation for screen, and Inst is an abbreviation for install.


If we want to generate code for a particular object, we click the dash (–) under the Code column for that object. Generating the code for some object just generates the source code, such as COBOL, C, or Java. If we actually want the source code compiled, linked with other compiled code, and in some cases bound to a DBMS, we would also select the dash under the Inst column. If we were generating a Blockmode application, it would likely have a green screen associated with it. To generate the screen, we would select the dash under the Scrn column. Finally, CA Gen provides the capability to trace through the executing code to assist with debugging. To enable this capability, additional code needs to be added to the generated source code. To add this additional code to the generated source code, you select the dash under the Trce column. Triggers have no screen and cannot be traced; therefore there are no dashes under those columns.

2. Select the dashes under the Code and Inst columns for the Triggers.

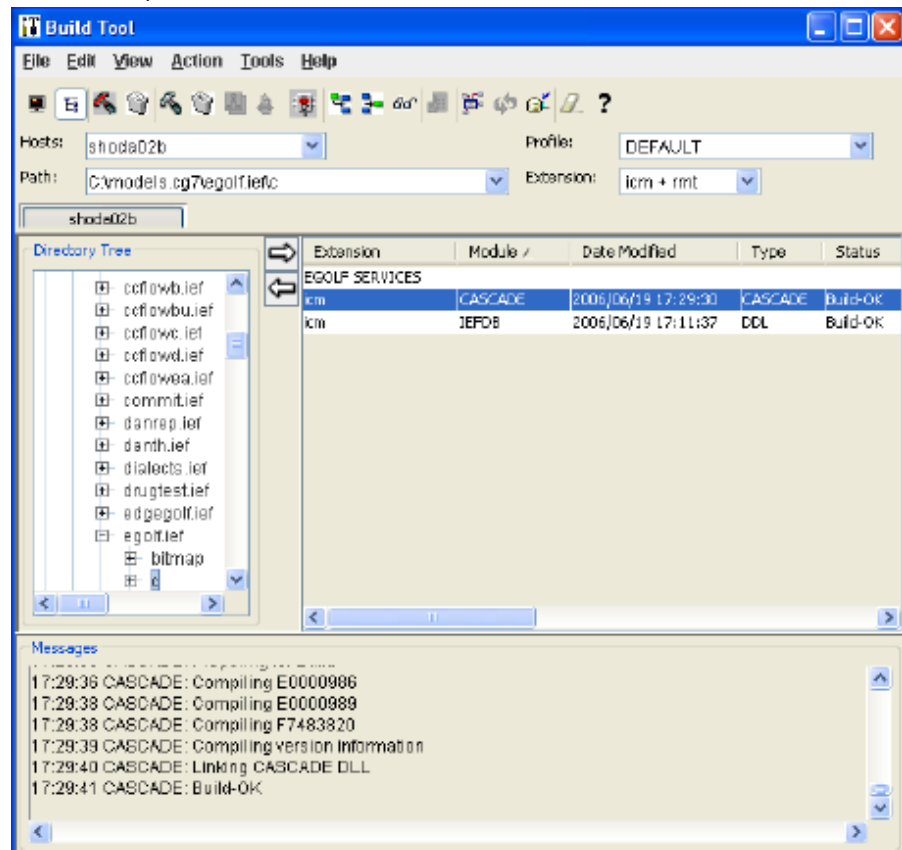


3. Select **Options** from the Main Menu, and then select **Generation Defaults**. Verify the targeted environment parameters are still as indicated below.



4. Select the [generate] **Code, Selected**  icon from the toolbar to initiate the code generation and compilation. You will very briefly see the Generation Progress dialog again, which closes automatically if the Trigger generation is successful.
5. The Trigger generation happens in foreground, which means you cannot use the Toolset while the generation is taking place. The installation happens in background by starting up the Build Tool. If everything works successfully, the Build Tool displays the Build-OK status message as shown below.

Note: If you left the Build Tool open after the successful database installation, you will still see the IEFDB line item as well. If you had closed the Build Tool, then you will now only see the Cascade line item.



To review, what we have done so far is generate the Database and the Referential Integrity Triggers. Unless there is a change to the logical or physical database design, then these two tasks would normally not be done again.

Packaging Procedures into Load Modules

The following section deals with packaging procedures into load modules.

Lesson Objectives and Time Allotment

After this lesson you will understand:

- The purpose of Load Module packaging
- How to package a Load Module

Allow yourself approximately 15 minutes to complete this lesson.

Load Module Packaging

In our model, we have documented the requirements and the detailed design for an application. So far all we have is the model. However, from this model, we are going to generate a lot of source code. And from this source code we are going to compile and link it into some form of an executable (dll, lib, exe) program. We need to specify to the Toolset how we want all this done. This is called Load Module Packaging.

The kinds of things we need to specify are:

- Load Module (executable program) names
- Trancodes to use with which load modules
- Source code member names
- Library names

For testing our application, we are going to initially generate everything and compile it into one load module (executable), and test it all locally in a Windows environment. However, for deploying to the web, we will need to repack the application into a client load module and a server load module. Client and Server packaging is known as Cooperative packaging. Other types of packaging include Online, Batch, and Component packaging.



Lesson Activity

In this exercise, we're going to:

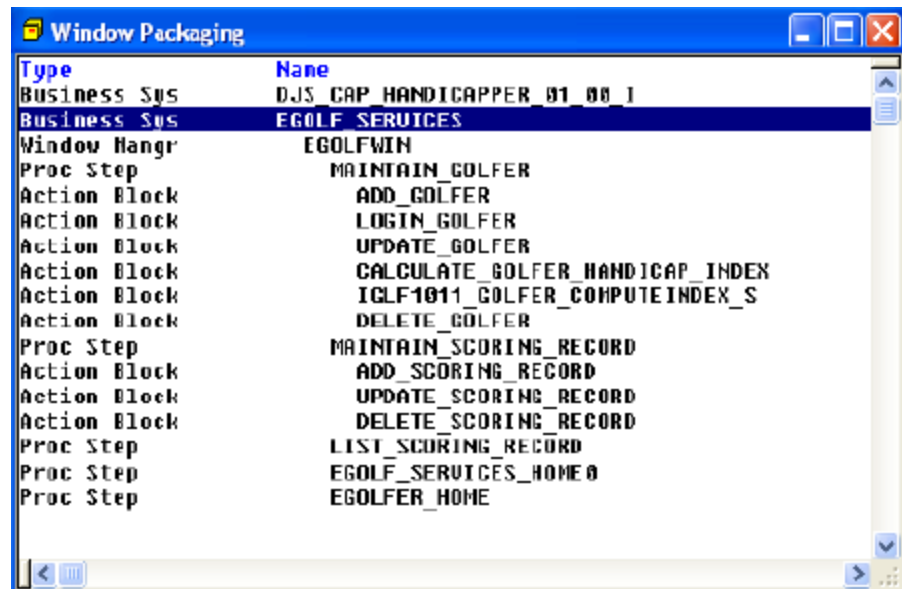
- Define the Window packaging by adding a load module called EGOLFWIN to the GOLF_SERVICES business system.
- Add all of the procedure steps to this one load module.
- Use the Complete option to complete the assignment of Trancodes and Source Code Member Names.

Define the Window Packaging


Follow these steps:

1. From the Tree Control, scroll down to the Construction folder and double-click **Packaging**.
2. From the Tool Palette, select the **Window Packaging**  icon.
3. In the Window Packaging diagram, select the **EGOLF_SERVICES** Business System.
4. From the Tool Palette, select the **Add Load Module...**  icon. In the (new object) Properties dialog, enter **egolfwin** and select the **Add Psteps...** push button. Select each procedure step (Proc Step) and, with all five-procedure steps highlighted, select the **OK** push button.
5. In the (new object) Properties dialog, select the **OK** push button. If you have multiple adds turned on, select the **Cancel** push button.
6. Notice that something called a Window Manager (Window Mangr) has been added below the Business System. Every load module (executable) in a Windows environment has a Window Manager associated with it. The Window Manager handles some of the window functionality, as well as things such as flows between other load modules.

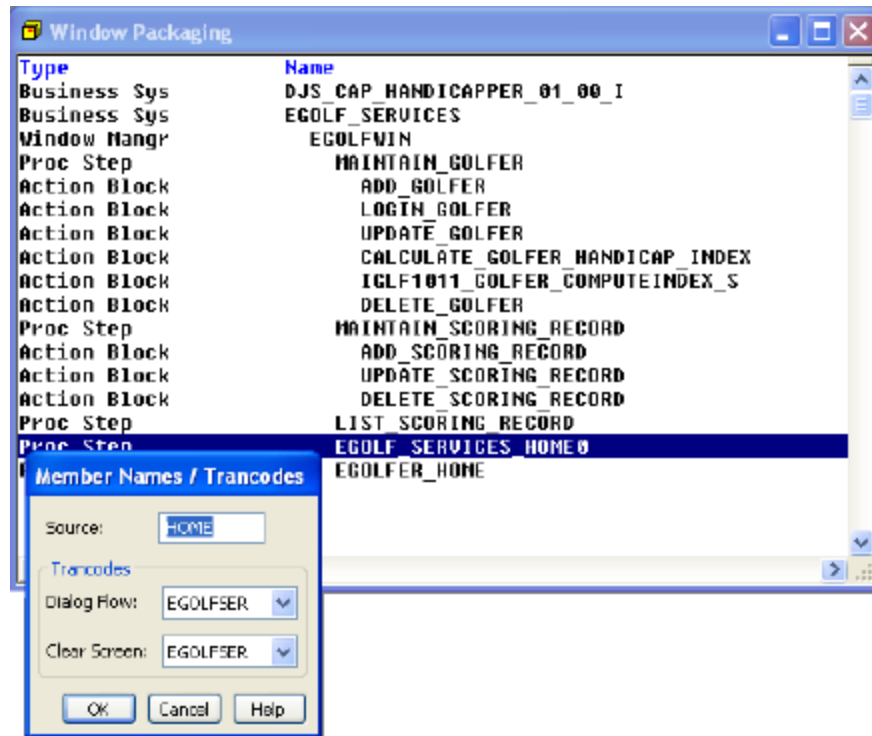
Select the **EGOLFWIN** Window Manager and then, from the Main Menu select **View**, then **Expand All**. Notice that each procedure step we added appears indented below the Window Manager name. And each action block called by each procedure step appears indented below the procedure step name. Thus, when we generate the code for all of these modules, they will all be linked into one executable called EGOLFWIN.



7. Before we can generate the code, however, we still need to complete the packaging by assigning source code member names to all of the modules and trancodes to the procedure steps. This one executable is going to have the equivalent of five entry points. Trancodes are used to specify a particular entry point. In a Mainframe environment, these trancodes would correspond to those registered with the TP Monitor. For other environments, they can be any unique name. We can assign trancodes and source member names manually, or we can let the Toolset complete the packaging for us. We will do the latter.

Select the Window Manager **EGOLFWIN**. Then, from the Tool Palette, select the **Complete**  [packaging] icon.

8. In the Window Packaging diagram, double-click the **EGOLF_SERVICES_HOME0** procedure step and make note of the Dialog Flow Trancode name. More than likely it is called **EGOLF5ER**. We will need to know this when we begin testing. Change the Source: from **P1** to **HOME** and select **OK**.



9. Save your model.

Generating the Application

The following sections deal with generating the application.

Lesson Objectives and Time Allotment

After this lesson you will understand:

- How to configure the Build Tool to support component linking
- How to define the targeted environment
- How to generate and install the application

Allow yourself approximately 45 minutes to complete this lesson.

Application Generation

Now that we have told CA Gen how we want everything generated, we need to specify the environment that we want the generated code targeted to. We need to specify a supported Operating System, DBMS, Source Code Language, Transaction Processing Monitor, Communications, and Installation Type. Depending on the choices you make for any one of these parameters, you may then be limited in your choices on others. Your options may further be limited depending on your licensing agreements.

And as a reminder, successfully completing this section of the *Tutorial* requires that you have a properly configured CA Gen Workstation, Build Tool, and Runtime environment, and that you have also installed and properly configured a supported DBMS, SQL precompiler, source language compiler, Web server, Java Runtime Environment, and Java Compiler. Instructions for setting up each of these surrounding technologies are outside the scope of this Tutorial.

Lesson Activity

In this exercise, we're going to:

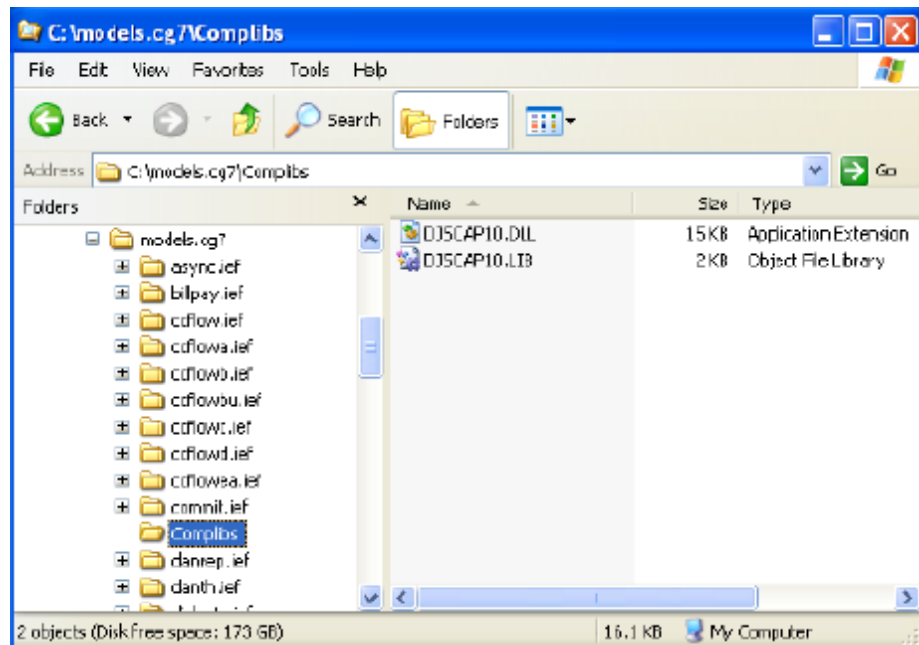
- Configure the Build Tool to support Component linking.
- Define the Windows environment.
- Generate the Windows-based application.

Configure the Build Tool to Support Component Linking

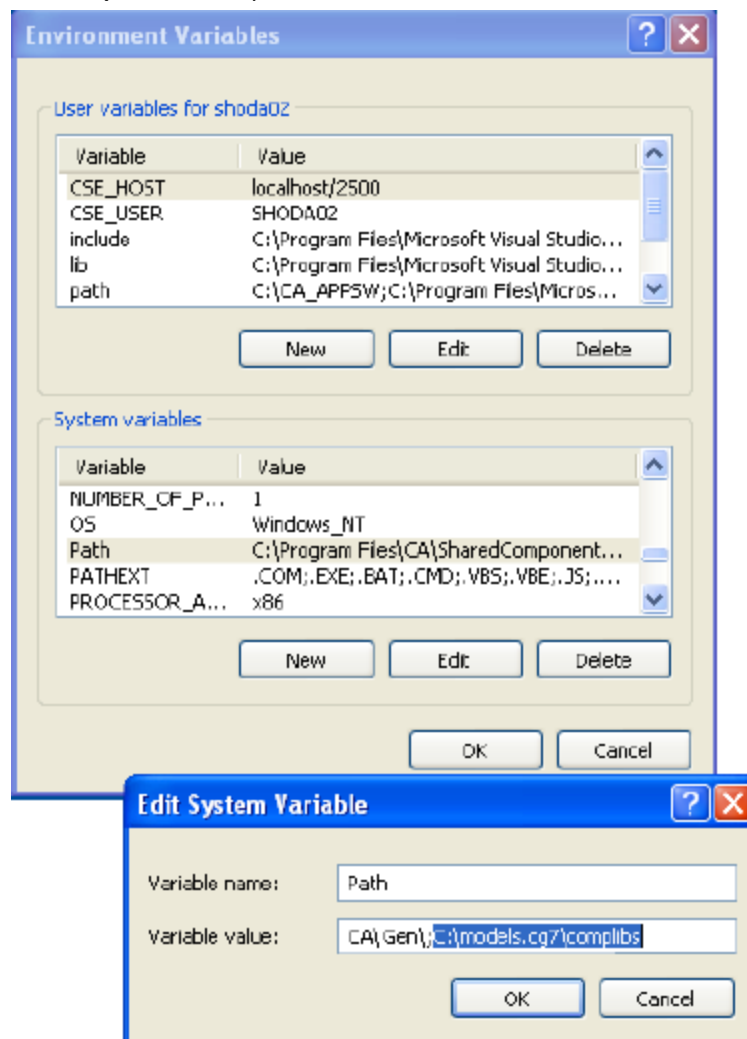
As part of the application generation, we need to link the code from the component `DJS_CAP_HANDICAPPER_01_00_I` into our generated application.

Follow these steps:

1. Using Windows Explorer or a Command Prompt, create a folder to contain all of your consumed components operations libraries. For example, in the directory containing your models, `C:\Models` in our case, create a folder called **Complibs**.
2. Copy the `DJS_CAP_HANDICAPPER_01_00_I` components operations **LIB** and **DLL** into this folder. The operations LIB and DLL are called `DJSCAP10`, and can be found in the `C` sub-directory within the components model (`djscap1a.ief`), which should be in your model directory.

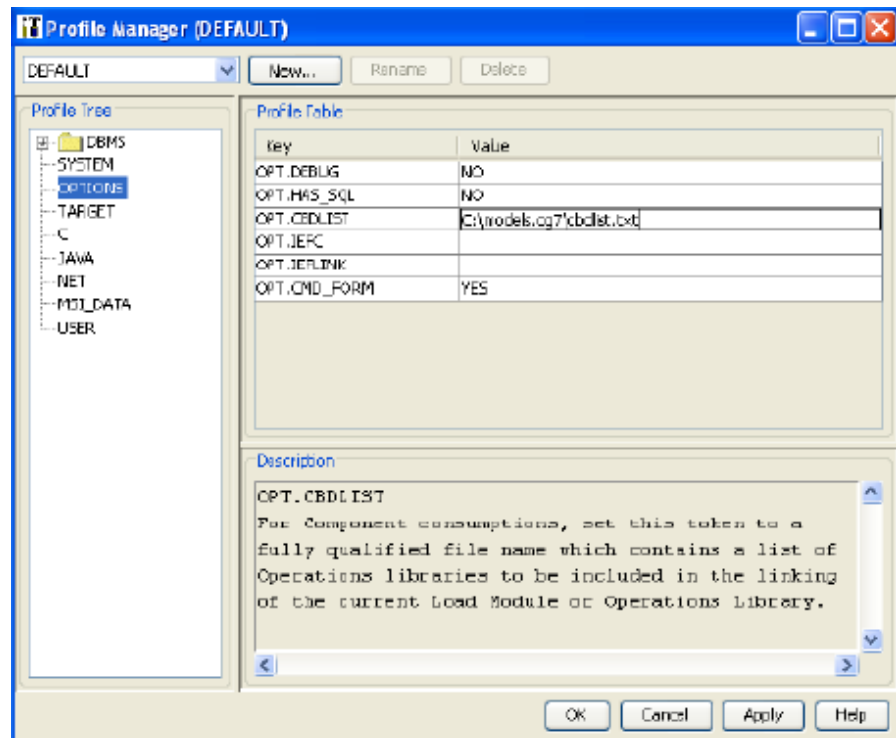


3. Update the System Environment Path variable to include the path to the Complibs directory so that the DLL can be found when it is needed. To do so, right-click **My Computer** on your desktop (if it is there) and select **Properties** from the pop-up menu. Then, from the System Properties dialog, select the **Advanced** tab, and then select the **Environment Variables** push button. Under the System variables select the **Path** variable, then select the **Edit** push button. Add the fully qualified path to the **Complibs** directory.



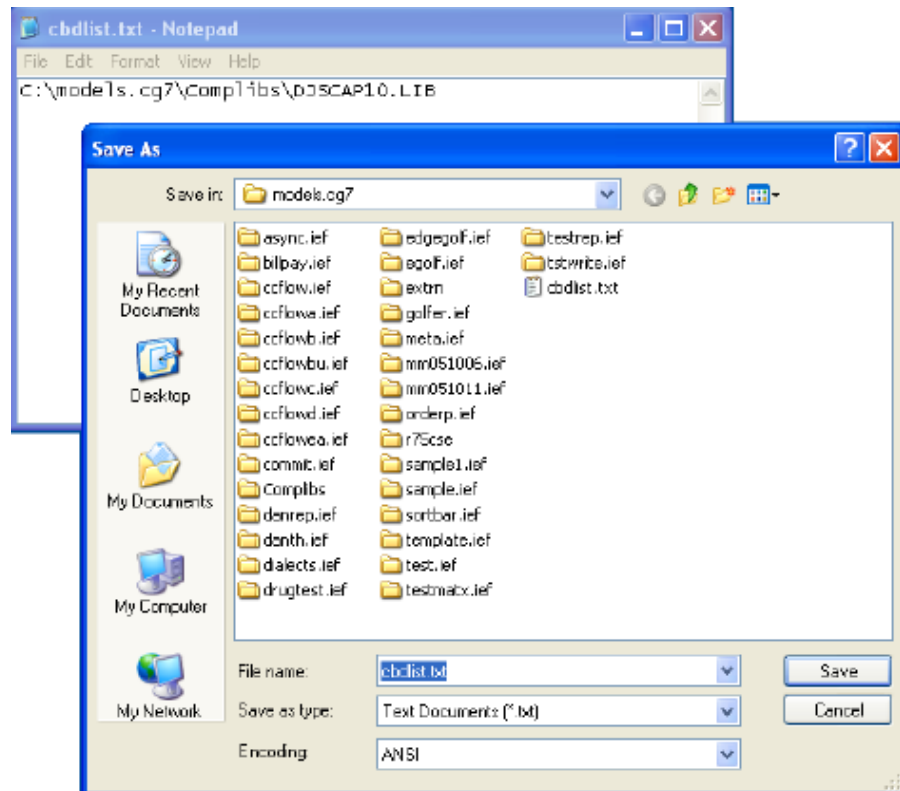
4. If you have already closed the Build Tool, you can reopen it again by going to the **Start** menu, selecting **Programs**, and then follow the path to your CA Gen installation. Most likely, this will be **CA, Gen rX** (where **X** is the release number), and then **Build Tool**.

From the Build Tools Main Menu, select **Tools** and then select **Profile Manager...** to open the appropriate Profile Manager, most likely the Default. Select **OPTIONS** from the Profile Tree on the left. Double-click the **Value** cell opposite the **OPT.CBDLIST** token and enter the fully qualified path to a text file which will contain a listing of all of the component libraries used by this application. In the example below, we have used **C:\Models\cbdlist.txt**.



5. Finally, we need to create the text file containing the list of components in the path we specified in the Build Tool. Open Windows **Notepad** and select **File**, select **Save**, and then select **New**.

6. In the new untitled Notepad text file, enter the fully qualified path to the components Operations Library. Continuing with our example, it would be **C:\Models\Complibs\DJSCAP10.LIB**. Then from Notepad's Main Menu, select **File**, select **Save As...** and save this file as **cbdlist.txt** in the path **C:\Models** that we specified in the Build Tool.

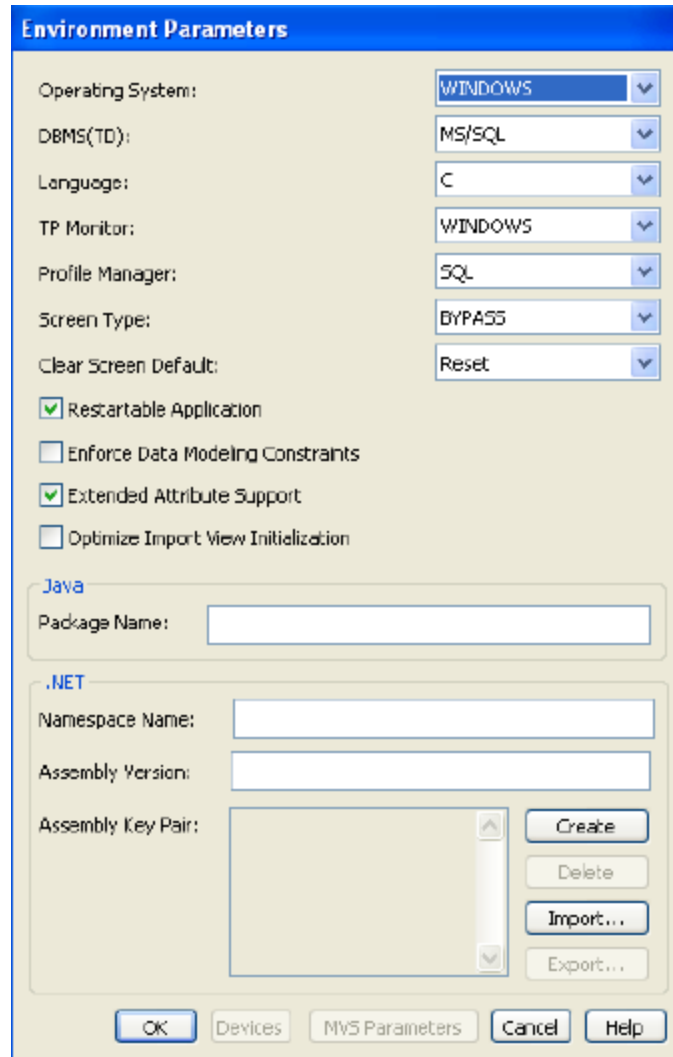


To summarize, in the Build Tool we have a token (OPT.CBDLIST) that points to a text file (C:\Models\cbdlist.txt). In this text file, we have a fully qualified listing of all the component libraries used by our application. In our particular case, it is only the one component and its LIB is in C:\Models\Complibs. Within the C:\Models\Complibs directory, we have copied the components LIB and DLL. And finally, we have identified the path to this directory to the operating system by updating the operating system's System Environment Path variable so it can pick up the DLL when it is needed. Normally this configuration is only done once. Any new components needed would be copied into the Complibs directory and the cbdlist.txt file would be updated.

Define the Windows Environment

Follow these steps:

1. From the Tree Control, scroll down to the **Construction** folder and double-click **Environment**.
2. In the Environment diagram, double-click the **EGOLF SERVICES** Business System. Verify your Environment Parameters are as indicated below or that the DBMS(TD) points to your specific database. If not, change them and select the OK push button.




The image shows the 'Environment Parameters' dialog box. It has a blue title bar and a light beige background. The dialog is divided into several sections. The top section contains seven dropdown menus: 'Operating System' (set to WINDOWS), 'DBMS(TD)' (set to MS/SQL), 'Language' (set to C), 'TP Monitor' (set to WINDOWS), 'Profile Manager' (set to SQL), 'Screen Type' (set to BYPASS), and 'Clear Screen Default' (set to Reset). Below these are four checkboxes: 'Restartable Application' (checked), 'Enforce Data Modeling Constraints' (unchecked), 'Extended Attribute Support' (checked), and 'Optimize Import View Initialization' (unchecked). The next section is titled 'Java' and contains a 'Package Name' text field. The final section is titled '.NET' and contains 'Namespace Name', 'Assembly Version', and 'Assembly Key Pair' text fields. To the right of the 'Assembly Key Pair' field are four buttons: 'Create', 'Delete', 'Import...', and 'Export...'. At the bottom of the dialog are five buttons: 'OK', 'Devices', 'MVS Parameters', 'Cancel', and 'Help'.

Environment Parameters	
Operating System:	WINDOWS
DBMS(TD):	MS/SQL
Language:	C
TP Monitor:	WINDOWS
Profile Manager:	SQL
Screen Type:	BYPASS
Clear Screen Default:	Reset
<input checked="" type="checkbox"/> Restartable Application	
<input type="checkbox"/> Enforce Data Modeling Constraints	
<input checked="" type="checkbox"/> Extended Attribute Support	
<input type="checkbox"/> Optimize Import View Initialization	
Java	
Package Name:	
.NET	
Namespace Name:	
Assembly Version:	
Assembly Key Pair:	
	Create
	Delete
	Import...
	Export...
OK Devices MVS Parameters Cancel Help	

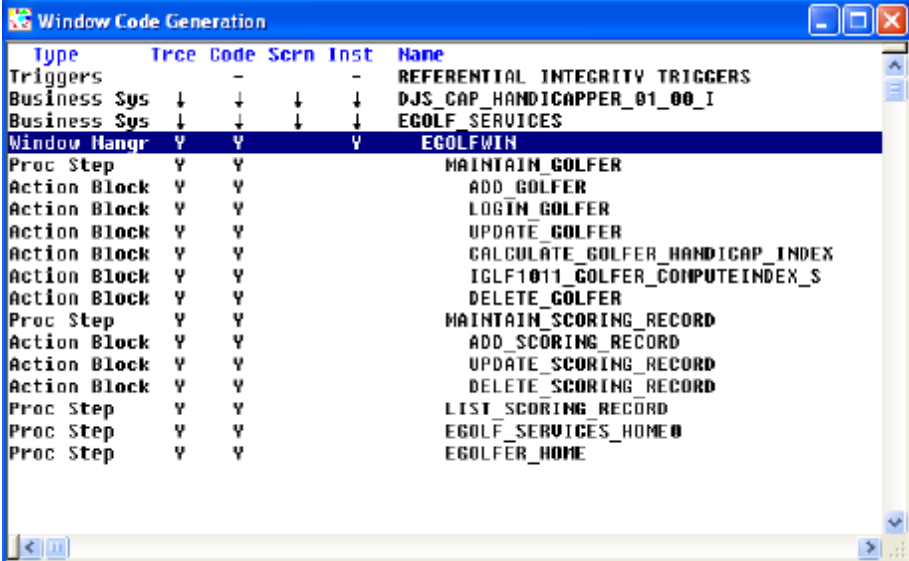
Generate the Windows Application

Follow these steps:


1. If necessary, open the Window Code Generation diagram again. To do so, from the Tree Control select **Generation** and then from the Tool Palette select the **Window Code [generation]**  icon.
2. In the Window Code Generation diagram, select **the EGOLF SERVICES Business System (Business Sys)** and from the Main Menu select **View** and then select **Expand All**.

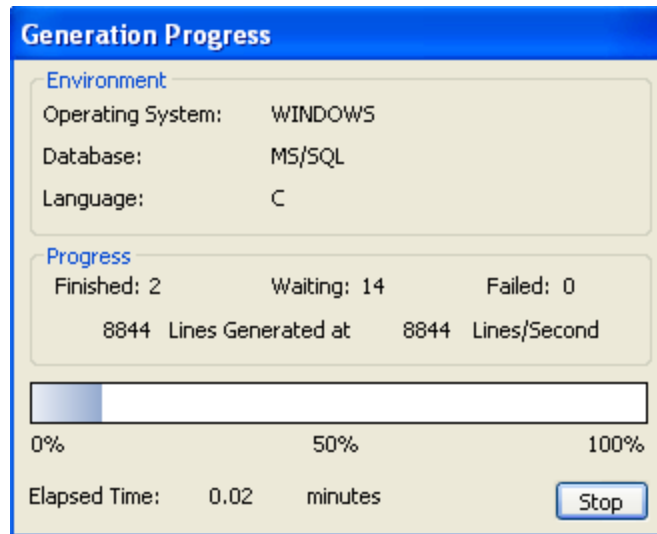
Note: A very common generation mistake is failing to fully expand a module for generation and, subsequently, not generating all of its sub-modules. However, you should strive to only generate the modules that require generation, and avoid falling into the bad habit of regenerating everything all the time.

3. All of the code needs to be generated. We can select each **dash (-)** under the **Code** column individually, or we can simply select the **down arrow** under the **Code** column to have all of the items selected at once. We also want to trace the application for testing, so we want to select the **down arrow** under **Trce** as well. Finally, selecting the Code and Trce columns only causes the source code to be generated. To have the source code compiled into an executable program, we need to select either the **dash** under the **Inst** column or the **down arrow** under the **Inst** column. The Window Code Generation diagram should look as follows:

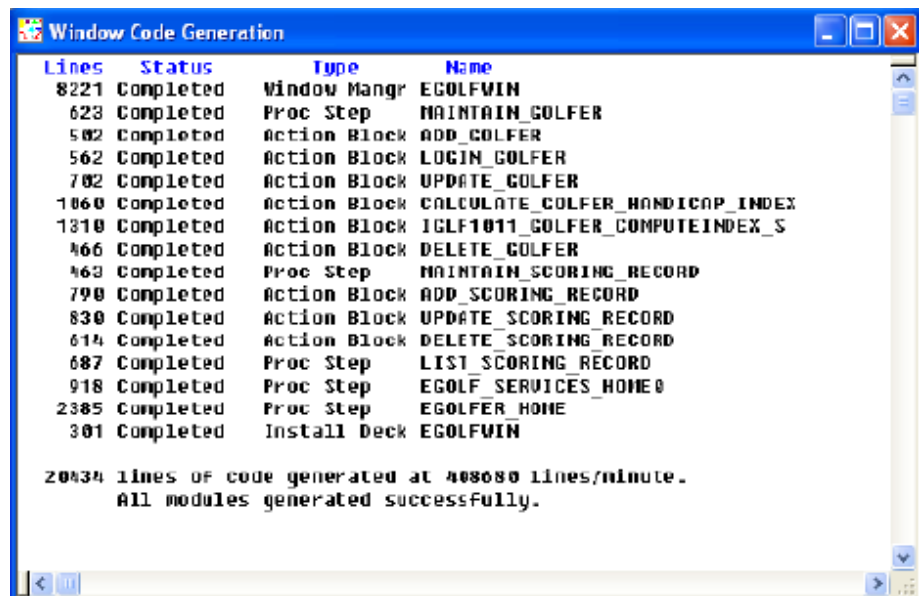


Type	Trce	Code	Scrn	Inst	Name
Triggers	-	-	-	-	REFERENTIAL INTEGRITY TRIGGERS
Business Sys	↓	↓	↓	↓	DJS_CAP_HANDICAPPER_01_00_I
Business Sys	↓	↓	↓	↓	EGOLF SERVICES
Window Mangr	Y	Y		Y	EGOLFWIN
Proc Step	Y	Y			MAINTAIN_GOLFER
Action Block	Y	Y			ADD_GOLFER
Action Block	Y	Y			LOGIN_GOLFER
Action Block	Y	Y			UPDATE_GOLFER
Action Block	Y	Y			CALCULATE_GOLFER_HANDICAP_INDEX
Action Block	Y	Y			IGLF1011_GOLFER_COMPUTEINDEX_S
Action Block	Y	Y			DELETE_GOLFER
Proc Step	Y	Y			MAINTAIN_SCORING_RECORD
Action Block	Y	Y			ADD_SCORING_RECORD
Action Block	Y	Y			UPDATE_SCORING_RECORD
Action Block	Y	Y			DELETE_SCORING_RECORD
Proc Step	Y	Y			LIST_SCORING_RECORD
Proc Step	Y	Y			EGOLF_SERVICES_HOME0
Proc Step	Y	Y			EGOLFER_HOME

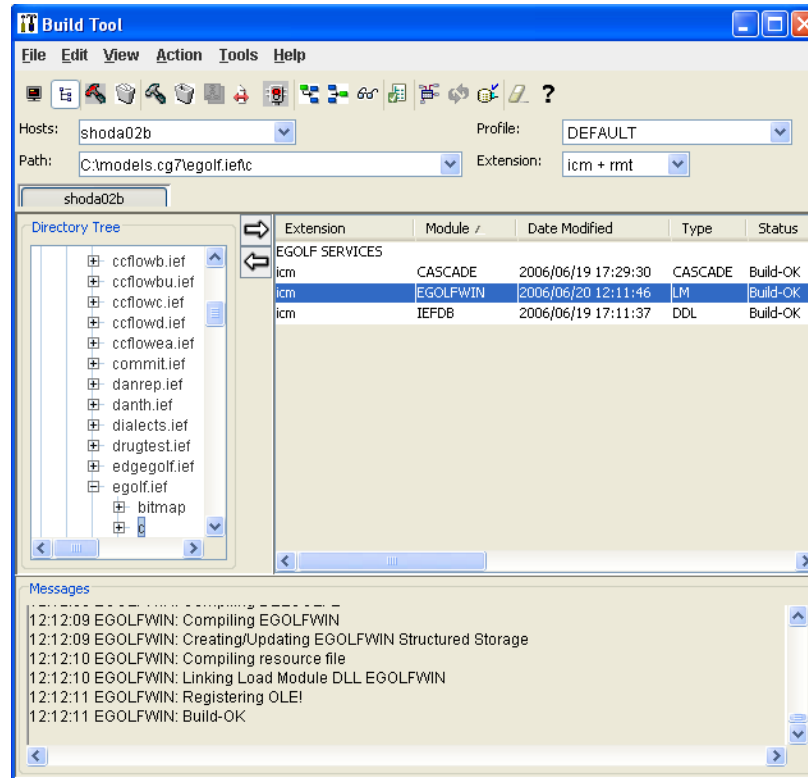
4. To start the code generation, from the Menu Bar select **Generate**, and then select **Code, Selected**. Alternately, simply select the [generate] **Code, Selected**  icon from the Tool Palette. Depending on the speed of your workstation, you will briefly see the Generation Progress dialog shown below, which will close automatically if the code generation is successful.




If the generation is successful, in the Window Code Generation diagram you will see that the list of modules to be generated has been replaced with a count of the total number of lines of source code generated for each module.



The code generation happens in foreground, which means you cannot use the Toolset while the generation is taking place. The installation happens in background by starting up the Build Tool. If everything works successfully, the Build Tool displays the Build-OK status message as shown below.



You can review the results of the installation by selecting the Load Module in the Build Tool as shown above and then selecting the **Review**  icon on the Toolbar.

Testing the Application

The following sections deal with testing the application.

Lesson Objectives and Time Allotment

After this lesson you will understand:

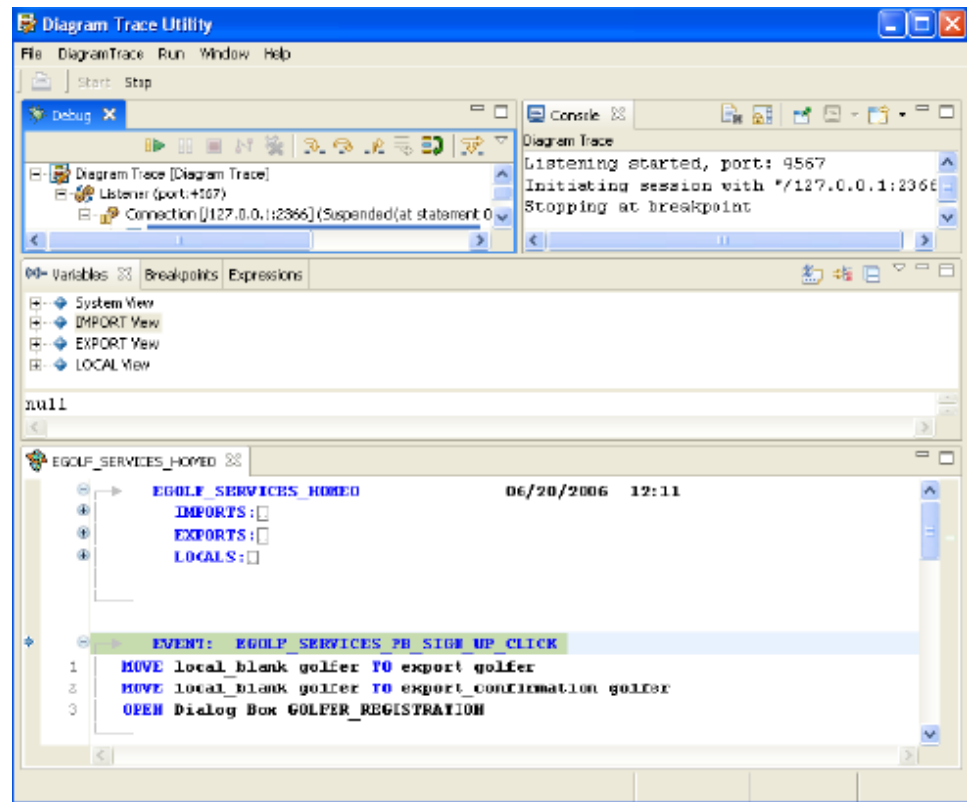
- How to use the Diagram Trace Utility to test your application

Allow yourself approximately 45 minutes to complete this lesson.

Application Test

CA Gen provides an interactive Diagram Trace Utility to assist with debugging an application. It lets you follow the execution of statements in the action diagram and examine and dynamically change the contents of views and system attributes. To use the Diagram Trace Utility, the application (or parts of the application) must have been generated with trace. The application should be regenerated without trace prior to deployment to a production environment.

When using the Diagram Trace Utility within a Windows environment, you are presented with the default perspective (window layout) as shown below.



The default perspective is split into five views (Debug view, Console view, Variables view, Breakpoints view, and Expressions view) and a PAD Code View editor which shows the action diagram being traced.

- The Debug view shows that the Diagram Trace Utility is started and listening on the configured port (in this case, the default port 4567).
- The Console view is a read-only view that displays standard output messages.

- The Variables view displays the system variables and import, export, local, and entity action view data.
- For information on the Breakpoints and Expressions views, see the *Diagram Trace Utility User Guide*.
- The PAD Code View editor is a read-only viewer that lets you view the action diagram statements as they are being executed. You can step through one or more statements of the action diagram at a time by clicking one of the icons from the Debug toolbar. The highlighted statement is the statement that will be executed on the next step through the action diagram. In other words, the highlighted statement has not yet been executed.

Lesson Activity

In this exercise, we're going to:

- Utilize the Windows Trace facility to trace through the application.

Trace the Application

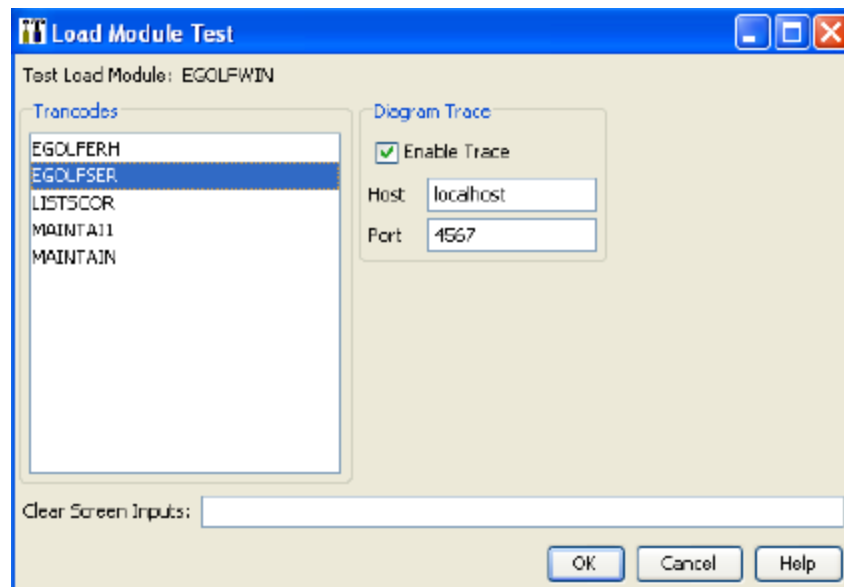
Follow these steps:

1. Start the Diagram Trace Utility by selecting **Start, All Programs, CA, Gen xx, Diagram Trace Utility**.

where, xx is the current CA Gen release number.

2. From the Build Tool, select the **EGOLFWIN** module, and then click the Test  icon on the toolbar.

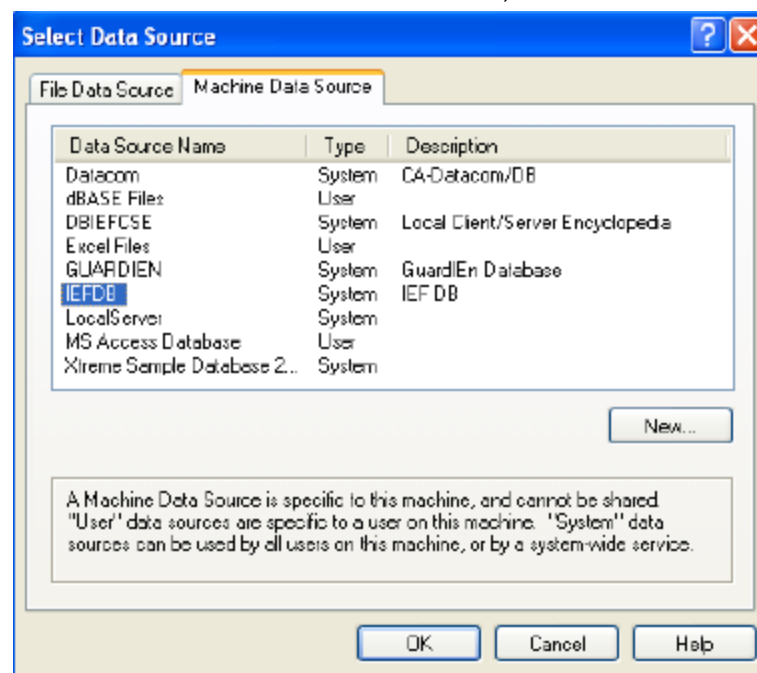
The Load Module Test dialog appears for the selected load module.



As mentioned earlier, this load module has the equivalent of five entry points. To enter at the eGolf Services Home page, select the tranocode **EGOLFSER** in the list of Trancodes. The application can be tested with or without actually tracing through the action diagram statements. To trace through the action diagram statements, ensure that the **Enable Trace** check box is checked, and then select the **OK** push button. The eGolf Services Home page should open.


If you get "Could not successfully verify that Diagram Trace Utility was started at: localhost:4567, ensure it is started and try again." error message, return to Step 1 and start the Diagram Trace Utility.

3. To use the application, we must first **sign up!** Select the sign up! push button. When presented with the Select Data Source window, select the **Machine Data Source** tab. Then select the **IEFDB** Data Source Name, and then select the **OK** push button.



4. If a DBMS login dialog opens, enter the appropriate User ID and Password. In the example below, we just need to enter the SQL (User) ID, and Password, and then select the **OK** push button.



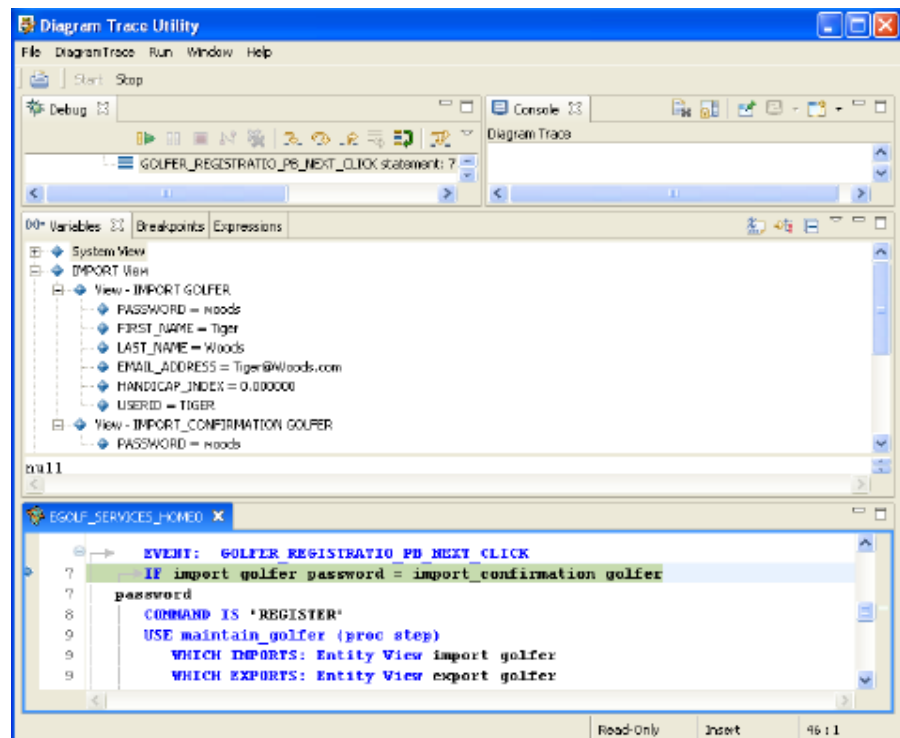
5. The Diagram Trace Utility main window shown earlier then opens. If you recall, in Design we assigned a click-event to the sign up! push button. We let the Toolset name the event GOLF_SERVICES_PB_SIGN_UP_CLICK. The Diagram Trace Utility shows that we are about to enter that event.
6. From the Debug view toolbar, click the **Step Into**  icon. The highlighting moves to line 1, MOVE local_blank golfer to export golfer. If we wanted to, we could go to the Variables view and expand the **LOCAL** View by selecting the **plus sign** next to it, and then expand the **LOCAL_BLANK_GOLFER**. But, there is nothing in that view, which is why it is the "blank" view. We are just using it to initialize the export golfer view prior to opening the GOLFER_REGISTRATION dialog. This prevents the information from a prior session from being redisplayed in the dialog.
Note: It has not actually moved the contents of the local_blank golfer to the export golfer yet, but it will when we select the StepInto icon again.
7. From the Debug view toolbar, select the **Step Into** icon again. The highlighting moves to line 2, Move local_blank golfer to export_confirmation golfer. So the action in line 1 has now been executed and the action in line 2 will be executed when we select StepInto again.
8. Select the **Step Into** icon again. The highlighting moves to line 3, OPEN Dialog Box GOLFER_REGISTRATION. So the action in line 2 has now been executed and the action in line 3 will be executed when we select StepInto again.
9. Select the **Step Into** icon again. The highlighting moves to the bottom of the event action. When we select Step Into again, the eGolf Golfer Registration dialog will open.
10. Select the **Step Into** icon again. The Registration dialog opens. It may be underneath the Diagram Trace Utility. Control is indicated on the Windows task bar by the active task bar button.

Note: Since we are testing this application in a Windows GUI environment, we will not see any of the animation in our GIFs. That is because we are actually looking at the associated BMPs.

When we go to the Web, we will see the animation.

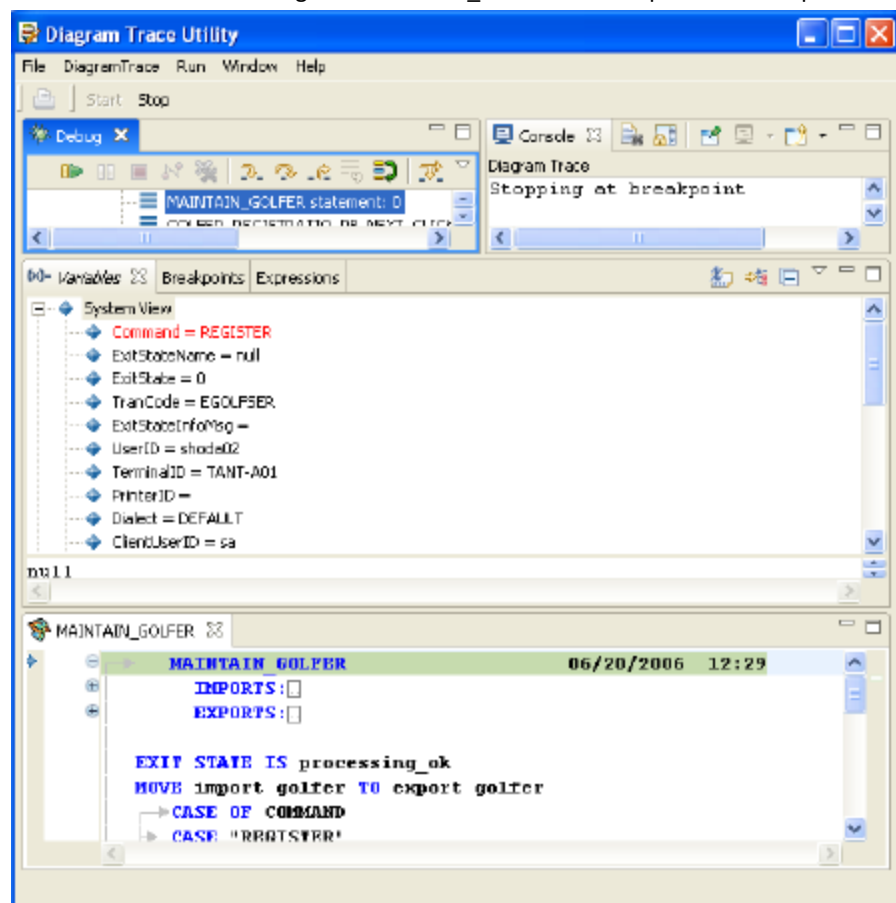
11. In the application's eGolf Golfer Registration dialog, enter the required information for a new user and select the **Next>** push button. In the Diagram Trace Utility, the highlighting is positioned at the `GOLFER_REGISTRATIO_PB_NEXT_CLICK` click-event that we have associated with that push button.
12. Select the **Step Into** icon again. The highlighting moves to line 7, which verifies that the two passwords entered match. This statement has not been executed, but will be executed the next time we select Step Into. If necessary, expand the **IMPORT** View in the Variables view. Select the + sign next to View **IMPORT GOLFER**, expanding the attribute views. Select the + sign next to View **IMPORT_CONFIRMATION GOLFER**, expanding the single attribute view.

In the example below the two passwords are identical, so we would expect the highlighting to proceed to the statement setting the Command to REGISTER. If the passwords were not the same, we would expect the highlighting to drop down to the statement setting the Exit State to password mismatch. Or, we can right-click either of the two passwords to change the values to be the same or different for testing purposes.



13. Select **Step Into** icon again. As predicted, the highlighting moves to the setting of the Command system variable. However, it has not yet set the Command. We can verify this by expanding the **System View** variables. Next to the attribute Command, there is no value.
14. Select **Step Into** icon again. The highlighting now moves to the USE maintain_golfer (proc step) statement. Notice that under the System View variables, the Command has now been set to REGISTER. When we select Step Into again, we will effectively leave the client procedure for the maintain_golfer server procedure step. When we return from the server procedure step, we will return to the IF EXITSTATE statement directly below the currently highlighted USE (proc step) statement.
15. Select the **Step Into** icon again.

Note: We are now entering the MAINTAIN_GOLFER server procedure step.



16. Select the **Step Into** icon again.

Note: In the Variables System View notice that the ExitStateInfoMsg has not been entered. We are getting ready to initialize the Exit State.

17. Select **Step Into** icon again. The ExitStateInfoMsg has now been initialized to Processing OK. We are now ready to move the import views to the export views. Expand the Import View Variables and expand the IMPORT GOLFER view.

Note: The golfer information was successfully passed from the client to the server procedure.

18. Now, select the **EXPORT View** Variables and expand the **EXPORT GOLFER** view. Currently, its values are blank.

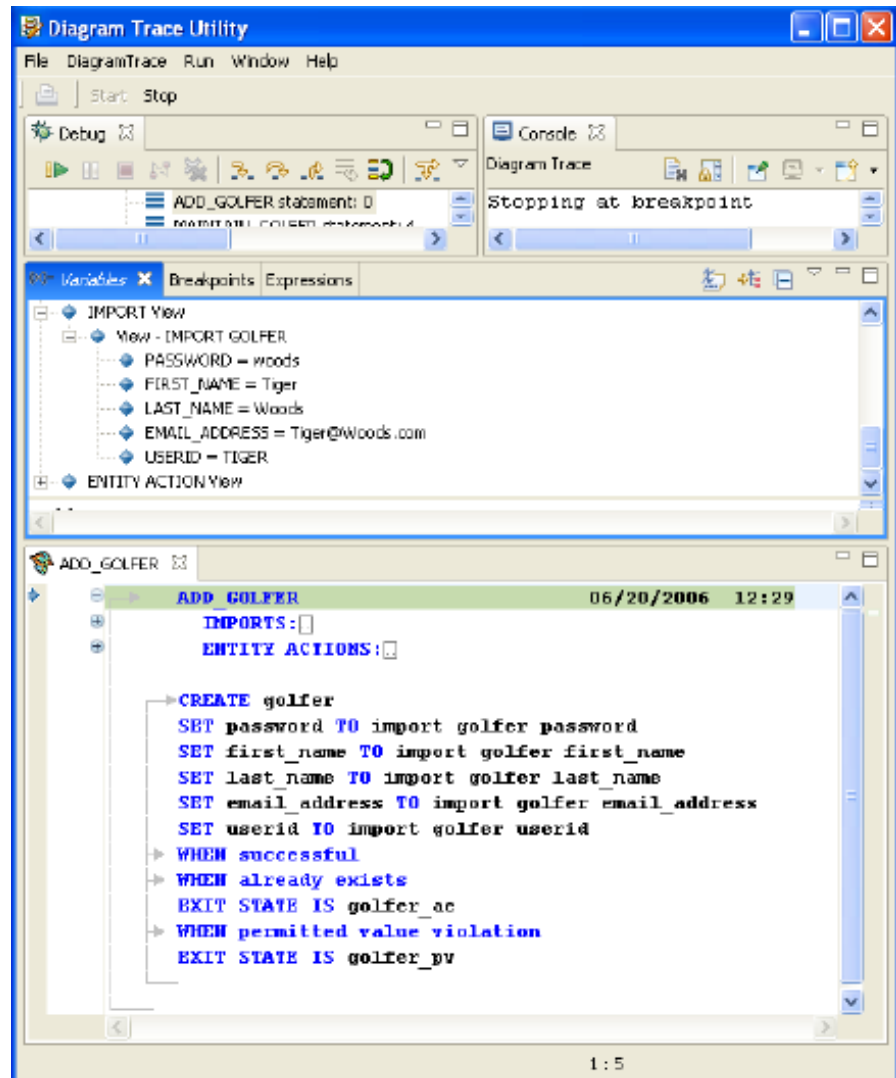
19. Select **Step Into** icon again.

Note: The export views have now been populated from the import views. As various action diagrams would be called below, any information they exported would overlay the matched view in the procedure steps export view. Only the login_golfer and calculate_golfer_handicap_index action blocks export information.

Since the system command is REGISTER, we would expect the next statement to be highlighted would be the CASE "REGISTER" statement.

20. Select the **Step Into** icon again. The CASE "REGISTER" statement is highlighted.
21. Select the **Step Into** icon again. The USE add_golfer statement is highlighted. When we select the Step Into icon again, we will be entering the add_golfer action block. Upon completion of that action block, we will be returned to the bottom of this server procedure step, since there are no other statements to be executed.

22. Select the **Step Into** icon again. We are now entering the ADD_GOLFER action block. Selecting Step Into will move us to the CREATE golfer statement.



23. Select the **Step Into** icon again. The CREATE golfer statement is highlighted.

Note: The Create action operates on the ENTITY ACTION View. We are going to create an occurrence of a golfer in our database, setting the Entity Action views to the values passed to us in the import views.

24. Expand the **ENTITY ACTION View** and expand View **GOLFER**. Currently, it is empty.

25. Select the **Step Into** icon again. The WHEN successful statement is highlighted.

Note: This indicates we did not receive an already exists or a permitted value violation. Expand View **GOLFER** once again and see what values were used to create the new occurrence of the golfer. As expected, they are the same values from the import views.

26. Select the **Step Into** icon again. This returns you to the bottom of this action diagram.
27. Select **Step Into** icon again. This returns us to the bottom of the server procedure step
28. Select the **Step Into** icon again. This returns us to the client procedure step just after the call to the server procedure step. After the server call, we will check the value of the ExitState system attribute to verify that everything worked okay. If so, we will change the ExitState value to the value that will cause us to flow to the golfer's home page, so our golfers can begin to enter their scoring records.
29. Select the **Step Into** icon again. The processing was OK, so we are now positioned to set the ExitState to the value which will cause us to flow to the golfer's home page.
30. Select the **Step Into** icon again. We have set the appropriate ExitState value to initiate the flow to the other client procedure step.
31. Select the **Step Into** icon again. We have executed the statement to close the GOLFER_REGISTRATION dialog and now we are at the bottom of the GOLFER_REGISTRATIO_PB_NEXT_CLICK event.
32. Select the **Step Into** icon again. In the background, you can see that we have switched to the eGolfer Home page. We have an open event associated with this window to pre-populate it with the list of scoring records for the golfer. This is a new golfer, so there will not be any scoring records.
33. Continue to Step through the open event for this client procedure. It will:
- Set the system command to LIST.
 - Call the list_scoring_record server procedure step that will attempt to read each scoring record for the golfer that was passed to it.
 - Return to the bottom of the client procedure step.
 - Display the eGolfer Home window with the golfer's first name and handicap index displayed. Keep in mind that it takes a minimum of five scoring records to establish a Handicap Index.

34. Continue to test and trace the application. If you would like, you can enter the sample test data shown in the table below.

- To stop tracing completely, click the Stop button on the Diagram Trace Utility main toolbar.
- For additional information, review the *Diagram Trace Utility User Guide*.

Note: Since this application is intended to be a Web Application, we have omitted the system menu from the Design, which makes it somewhat difficult to close the application. To close or kill the eGolf application, you will need to go to the Task Manager and from the Applications tab perform an End Task.

Date	Time	Score	Rating	Slope	Note
03/07/02	1300	90	70.1	116	Chase Oaks
03/05/02	1300	91	70.1	116	Chase Oaks
02/24/02	1300	94	72.3	123	Willow Bend
02/20/02	0900	88	70.1	116	Chase Oaks
01/18/02	0900	89	70.1	116	Chase Oaks
01/17/02	1200	90	72.3	123	Willow Bend
01/16/02	1300	91	72.3	123	Willow Bend
12/12/01	1200	91	70.1	116	Chase Oaks
12/10/01	1100	91	70.1	116	Chase Oaks
11/08/01	0900	86	68.7	105	Lonesome Dove
11/04/01	0900	90	70.1	116	Chase Oaks
11/01/01	1200	92	72.3	123	Willow Bend
10/24/01	1300	85	68.0	107	Goofy Golf
10/16/01	1400	78	68.7	105	Lonesome Dove
10/12/01	1500	82	70.1	116	Chase Oaks
10/02/01	1000	84	70.1	116	Chase Oaks
09/14/01	1000	94	72.3	123	Willow Bend
09/05/01	1200	93	72.3	123	Willow Bend
09/04/01	1700	89	72.3	123	Willow Bend
09/01/01	1800	88	70.1	116	Chase Oaks

Web Generation

The following sections deal with web generation.

Lesson Objectives and Time Allotment

After this lesson you will understand:

- How to package the application for Cooperative Processing (Client/Server)
- How to Generate the application for the Web
- How to deploy the application to the Web



Lesson Activity

In this exercise, we're going to:


- Package our application for Cooperative Processing.
- Generate the application for the Web.
- Deploy the application.

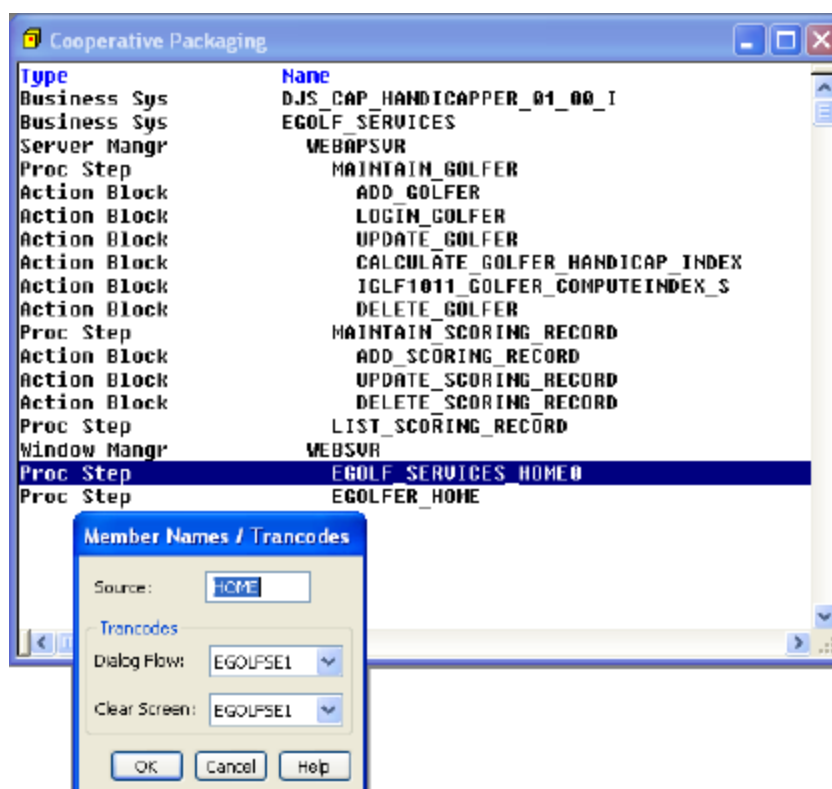
Package Our application for Cooperative Processing

Follow these steps:

1. From the Tree Control, scroll down to the **Construction** folder and double-click **Packaging**.
2. From the Tool Palette, select the **Cooperative Packaging**  icon.
3. In the Cooperative Packaging diagram, select the **EGOLF_SERVICES** Business System.
4. From the Tool Palette, select the **Add Load Module...**  icon. In the (new object) Properties dialog, enter **webapsvr** for the Load Module name, select **Server Mangr** for the Type of load module, and then select the **Add Psteps...** push button.

Notice that while we have five procedure steps in our model, we can only select three. That is because these were the only procedure steps defined as servers.

5. Select each procedure step (Proc Step), and with all three highlighted, select the **OK** push button. Then select **OK** on the (new object) Properties dialog to add the Server Manager to the Business System. Select the **Add Load Module** icon again. In the (new object) Properties dialog enter **websvr** for the load module, change the Type of load module to **Window Mangr**, and then select the **Add Psteps...** push button. This time we are just presented with the two client procedure steps. Select both procedure steps, and then select the **OK** push button. Then select **OK** again to add the new Window Manager to the Business System.
6. To complete the packaging, select each load module in the Cooperative Packaging diagram and then, from the Tool Palette, select the **Complete**  icon.
7. Double-click the **EGOLF_SERVICES_HOME0** procedure step and make note of the Dialog Flow Trancode name. More than likely it will be called EGOLFSE1.
8. Save your model.



Generate the Application for the Web

Follow these steps:

1. In order for our Web application to load the component, we need to copy the component's JAR file to the Web Application Server's lib folder. Copy the **DJS_CAP_HANDICAPPER_01_00_I** components operations JAR file into the **JBoss_Tomcat\server\default\lib** folder. The operations JAR file is called **djscap10.jar**, and can be found in the Java sub-directory within the components model (djscap1a.ief).


Note: The directory where you place the JAR file will be different depending on your version of Tomcat or on your application server.

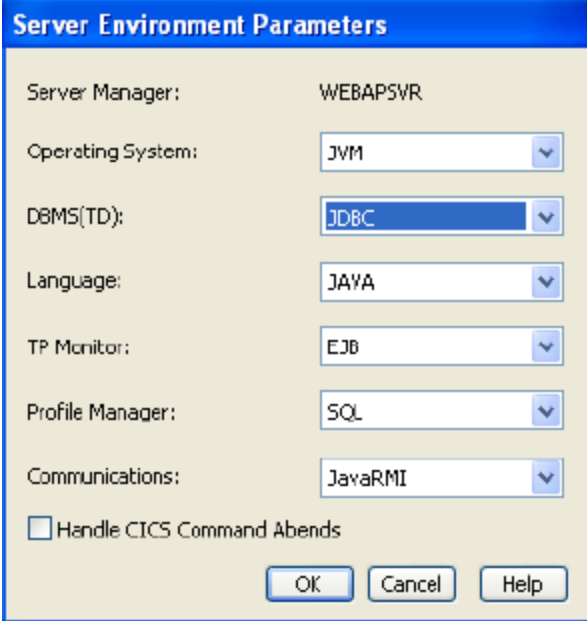
2. From the Tree Control, scroll down to the **Construction** folder and double-click **Generation**.

3. From the Tool Palette, select **Cooperative Code**  generation.

4. For the Web, we are going to be targeting two slightly different environments. One is for the clients targeting the Web server. The other is for the servers targeting a Web Application Server. We can define both of these environments from the Cooperative Code Generation diagram.

Select the Server Manager (Server Mangr) **WEBAPSVR**. From the Toolbar, select the

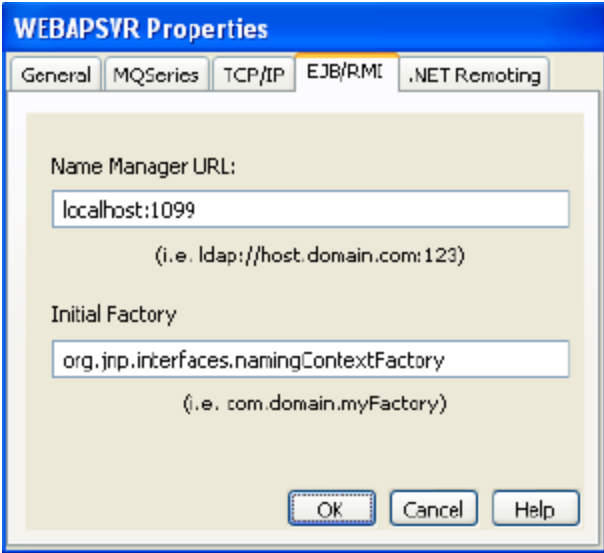
Server Environment...  icon. In the Server Environment Parameters dialog, set the selections as indicated below and select the OK push button.




The dialog box titled "Server Environment Parameters" contains the following settings:

Server Manager:	WEBAPSVR
Operating System:	JVM
DBMS(TD):	JDBC
Language:	JAVA
TP Monitor:	EJB
Profile Manager:	SQL
Communications:	JavaRMI
<input type="checkbox"/> Handle CICS Command Abends	
OK Cancel Help	

5. Double-click the Server Manager (Server Mangr) **WEBAPSVR** to open its properties dialog, then select the **EJB/RMI** tab and make the following entries.



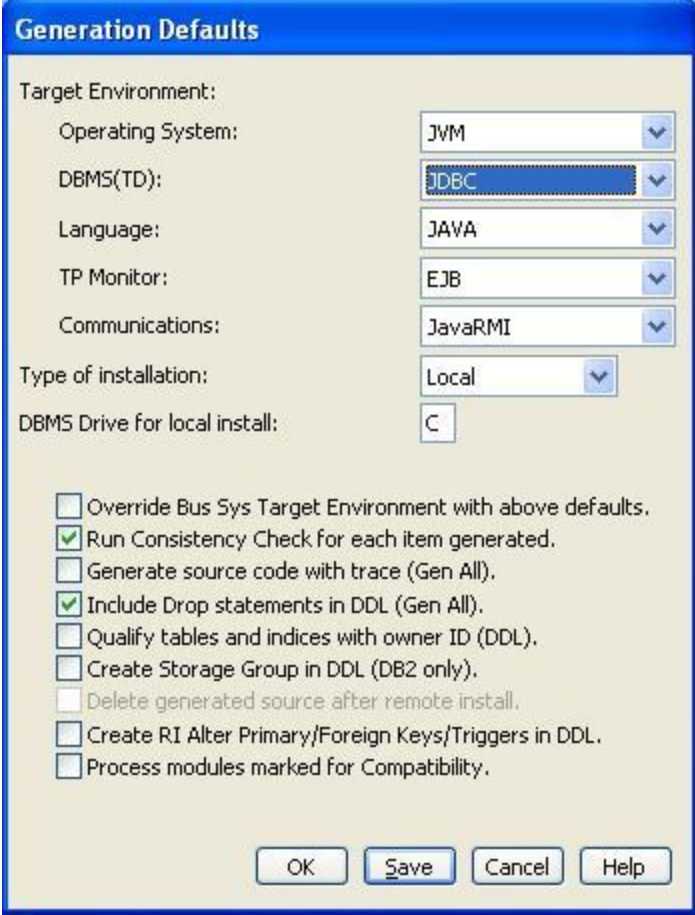
The screenshot shows the 'WEBAPSVR Properties' dialog box with the 'EJB/RMI' tab selected. The 'Name Manager URL' field contains 'localhost:1099' with a hint '(i.e. ldap://host.domain.com:123)'. The 'Initial Factory' field contains 'org.jnp.interfaces.namingContextFactory' with a hint '(i.e. com.domain.myFactory)'. At the bottom are 'OK', 'Cancel', and 'Help' buttons.

6. Select the Window Manager (Window Mangr) **WEBSVR**. From the Toolbar, select the **Client Environment...**  icon. In the Window Environment Parameters dialog, set the selections as indicated below and select the OK push button.



The screenshot shows the 'Window Environment Parameters' dialog box. The 'Window Manager' is set to 'WEBSVR'. The 'Operating System' is set to 'JVM'. The 'DBMS(TD)' is set to '<NONE>'. The 'Language' is set to 'JAVA'. The 'TP Monitor' is set to 'INTERNET'. The 'Profile Manager' is set to 'SQL'. There is an unchecked checkbox for 'Handle CICS Command Abends'. At the bottom are 'OK', 'Cancel', and 'Help' buttons.

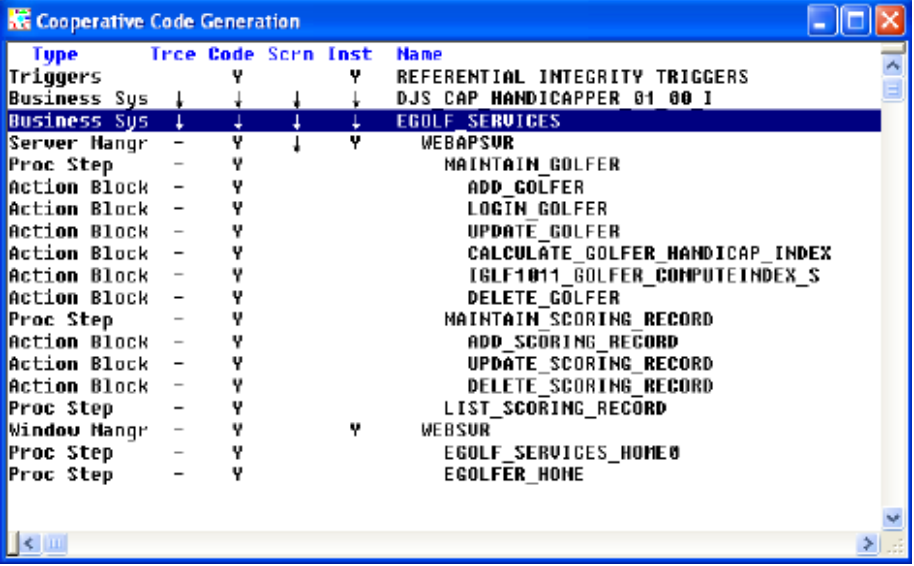
7. Finally, we need to specify the environment for the Triggers. From the Menu Bar, select **Options** and then **Generation Defaults**. Set the generation defaults as indicated below.



The image shows a 'Generation Defaults' dialog box with a blue title bar. It contains several configuration options for a target environment. The 'Target Environment' section includes dropdown menus for Operating System (JVM), DBMS(TD) (DB2), Language (JAVA), TP Monitor (EJB), and Communications (JavaRMI). Below these are 'Type of installation' (Local) and 'DBMS Drive for local install' (C). A list of checkboxes follows, with 'Run Consistency Check for each item generated' and 'Include Drop statements in DDL (Gen All)' checked. At the bottom are 'OK', 'Save', 'Cancel', and 'Help' buttons.

Generation Defaults	
Target Environment:	
Operating System:	JVM
DBMS(TD):	DB2
Language:	JAVA
TP Monitor:	EJB
Communications:	JavaRMI
Type of installation:	Local
DBMS Drive for local install:	C
<input type="checkbox"/> Override Bus Sys Target Environment with above defaults.	
<input checked="" type="checkbox"/> Run Consistency Check for each item generated.	
<input type="checkbox"/> Generate source code with trace (Gen All).	
<input checked="" type="checkbox"/> Include Drop statements in DDL (Gen All).	
<input type="checkbox"/> Qualify tables and indices with owner ID (DDL).	
<input type="checkbox"/> Create Storage Group in DDL (DB2 only).	
<input type="checkbox"/> Delete generated source after remote install.	
<input type="checkbox"/> Create RI Alter Primary/Foreign Keys/Triggers in DDL.	
<input type="checkbox"/> Process modules marked for Compatibility.	
OK Save Cancel Help	

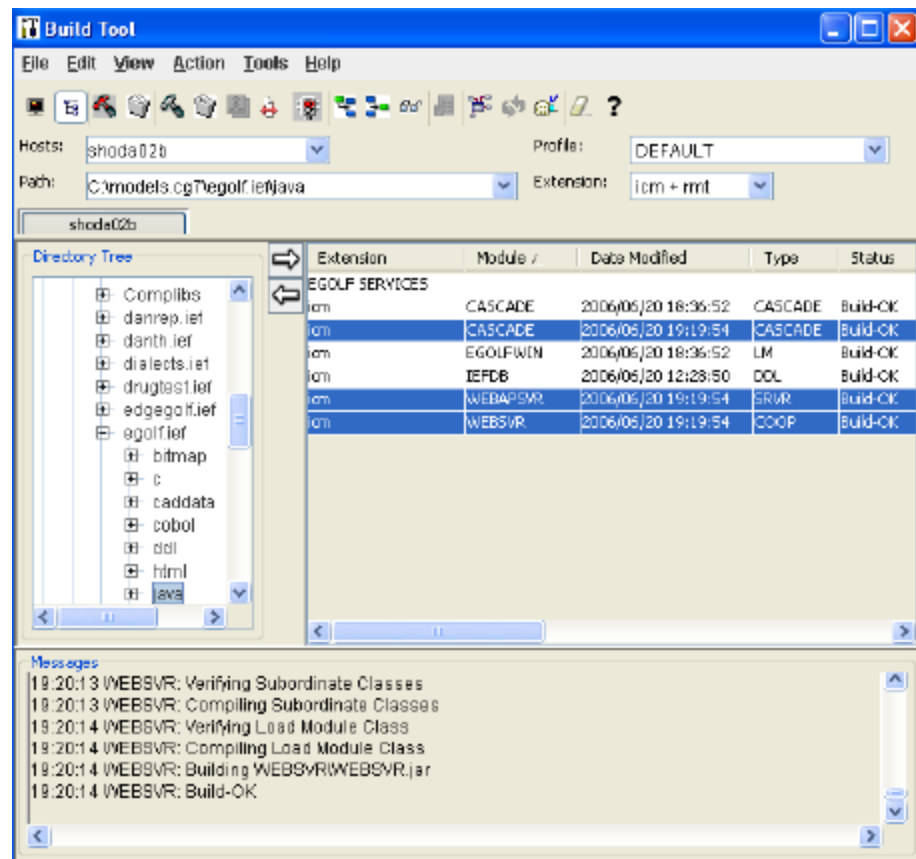
8. Generate all Cooperative Code and Triggers as shown below. See the earlier generation exercise if you need help in submitting these.




The screenshot shows a window titled "Cooperative Code Generation" with a table of generated code items. The table has five columns: Type, Trce, Code, Scrn, Inst, and Name. The items are listed in a hierarchical manner, with some items having sub-items indicated by arrows.

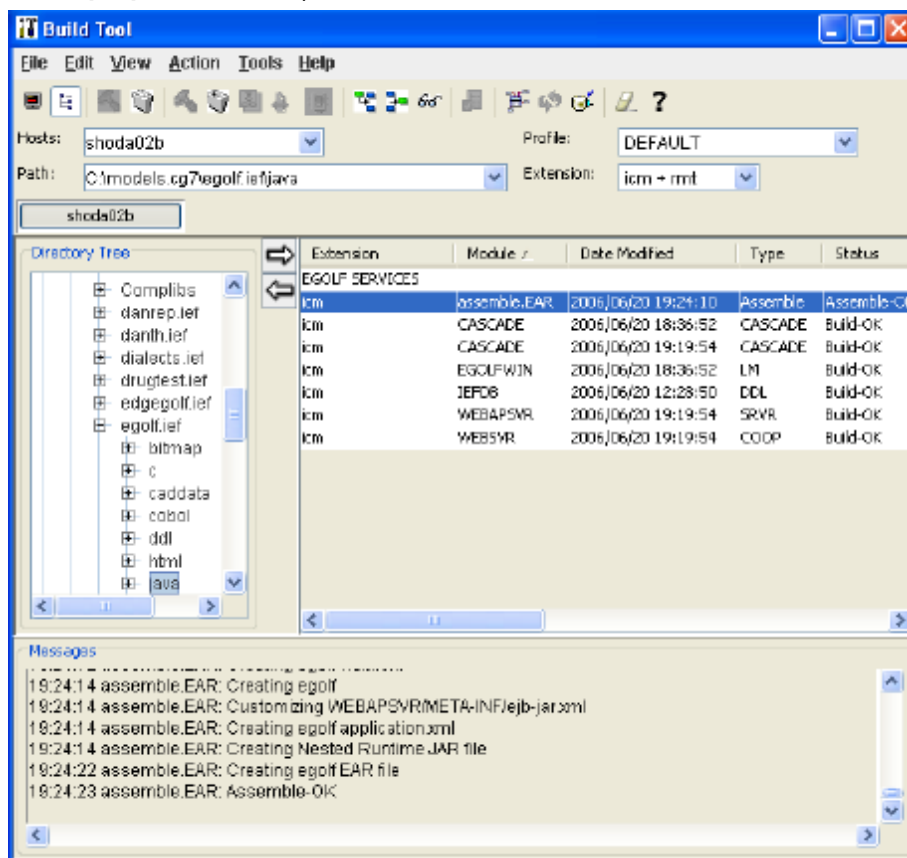
Type	Trce	Code	Scrn	Inst	Name
Triggers		Y		Y	REFERENTIAL INTEGRITY TRIGGERS
Business Sys	↓	↓	↓	↓	DJS CAP HANDICAPPER 01 00 I
Business Sys	↓	↓	↓	↓	EGOLF SERVICES
Server Hangr	-	Y	↓	Y	WEBAPSUR
Proc Step	-	Y			MAINTAIN_GOLFER
Action Block	-	Y			ADD_GOLFER
Action Block	-	Y			LOGIN_GOLFER
Action Block	-	Y			UPDATE_GOLFER
Action Block	-	Y			CALCULATE_GOLFER_HANDICAP_INDEX
Action Block	-	Y			ISGLF1011_GOLFER_COMPUTEINDEX_S
Action Block	-	Y			DELETE_GOLFER
Proc Step	-	Y			MAINTAIN_SCORING_RECORD
Action Block	-	Y			ADD_SCORING_RECORD
Action Block	-	Y			UPDATE_SCORING_RECORD
Action Block	-	Y			DELETE_SCORING_RECORD
Proc Step	-	Y			LIST_SCORING_RECORD
Window Hangr	-	Y		Y	WEBSUR
Proc Step	-	Y			EGOLF_SERVICES_HOME0
Proc Step	-	Y			EGOLFER_HONE

Remember, the source code generation happens in foreground, while the compilation happens in background. When complete, you should see the three new modules listed in the Build Tool as shown below.



9. Now we need to deploy the application to the Web Server. In the Build Tool, select (highlight) just the load modules **WEBAPSVR** and **WEBSVR**, and not the CASCADE line item. Select the **Assemble**  push button.

10. In the EAR File Deployment Details dialog, uncheck the check box **Package runtime in EAR** [file]. Select the **OK** push button.



11. After receiving the Assemble-OK status message as shown above, locate the **egolf services ear** file in your model's `java\deploy.j2ee` directory. Copy it into the `JBoss_Tomcat\server\default\deploy` folder.
12. Since we did not package the Runtime with the EAR file, we want to copy it to the Web Application Server's lib folder, so it will be available to both the component and the application. Having the Runtime in more than one location can cause problems. Locate the **Runtime jar** file (**genrt.##.jar**) in your model's `java\deploy.j2ee` directory and copy it into the `JBoss_Tomcat\server\default\lib` folder.
13. To start the JBoss Web Application Server and the Tomcat Web Server, navigate to the `JBoss_Tomcat\bin` directory and execute the **run.bat** file.
14. To test the application, open Internet Explorer and enter the following URL:
http://localhost:8080/egolf/egolfse1.

Appendix A: BLOB

This section contains the following topics:

[Audience](#) (see page 313)

[Objectives](#) (see page 314)

[Software Required to Complete this Appendix](#) (see page 315)

[Time Allotment](#) (see page 316)

[Analysis](#) (see page 317)

[Design](#) (see page 331)

[Construction and Test](#) (see page 374)

Audience

This appendix is intended to assist someone in implementing Binary Large Objects (BLOB) attributes into a CA Gen application.

Note: While BLOB attributes can be implemented in most CA Gen Environments, the Client side visualization of the BLOB contents in this example makes use of a third-party Windows OLE Control Extension (OCX) control. Because there are limitations when using OCX controls in a Java web environment, our Clients design can then only be generated targeting a Windows environment. See the Web Generation Guide for more details.

Objectives

After completing this Appendix, you will learn how to:

- Implement a BLOB attribute
- Use Process Synthesis
- Implement an OCX control (Chestysoft csXImageTrial)
- Use a Work Set
- Create/Use External Action Blocks (EABs)
- Perform an Incremental Transformation

In addition to the preceding topics, you will have an opportunity to reinforce the following concepts. Refer back to the appropriate chapter in the Tutorial if you need help with completing any of these tasks:

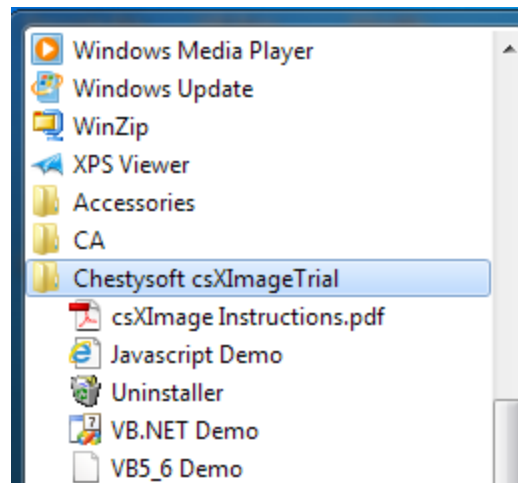
- Analysis (Requirements)
 - Data Modeling
 - Activity Modeling
 - Action Diagramming
- Design
 - Server Design
 - Client Design
- Construction and Test
 - Database Generation
 - Logical to Physical Data Model Transformation
 - DDL Generation
 - RI Triggers Generation
- Load Module Packaging
- Code Generation
- Testing the application

Software Required to Complete this Appendix

In addition to the software that was required to complete the Tutorial, to complete this appendix you need the following additional software:

- OCX Control

You can download a trial version of the Chestyssoft csXImage OCX control at <http://www.chestyssoft.com/ximage/download.asp>. Follow their instructions for installation. After the installation, you will see an entry in your Windows Start Menu similar to the following



- External Action Blocks

The OCX control displays images from a file on your file system. So we need to make use of two external action blocks, one that reads a file into a BLOB attribute view and another that writes the contents of a BLOB attribute view to a file. The external action blocks are available in the CA Gen installation `\Samples\Models\Tutorial Models\Starter Models\extrn` directory. Copy the extrn directory to your models directory. The model directory used in the examples is `C:\Models`.

You will be given instructions later in this appendix as to how to use them.

Time Allotment

Total - 3 hours 30 minutes

■ **Analysis (Requirements)**

45 minutes

- Case Study (aka Business Requirements)
5 minutes
- Data Modeling
20 minutes
- Activity Modeling
5 minutes
- Process Synthesis
5 minutes
- Action Diagramming
10 minutes

■ **Design**

1 hour 45 minutes

- Server Design
15 minutes
- Client Design
1 hour 30 minutes

■ **Construction and Test**

1 hour 0 minutes

- Database Generation
 - Logical to Physical Data Model Transformation
10 minutes
 - DDL Generation
10 minutes
- RI Triggers Generation
5 minutes
- Load Module Packaging

10 minutes

- Code Generation

15 minutes

- Testing the application

10 minutes

Analysis

Case Study

We received many requests from golfers that they would like the ability to save an image of their score card. This would allow them to immortalize that once-in-a-lifetime hole in one, that pro-am game with one of the golfing greats, or that holiday playing The Old Course at St Andrews Links. To satisfy this request, we will add the ability to save an image of their score card to our application. Golfers will then be able to scan an image of their score card, or take a picture of it with their smart phones, and save it in our database.

There are at least two ways to satisfy this requirement. The first might be to simply add a BLOB attribute to our existing Scoring Record entity type. And while someone could probably make a strong argument that this is where it goes, placing it there would have repercussions through much of our completed work. Therefore, for this exercise we will instead add another entity type to our data model containing the BLOB attribute. Ultimately we will end up making just about the same amount of changes or additions to our model, but by implementing it this way we will be able to better isolate and focus on those changes from the work we have already accomplished.

Data Model

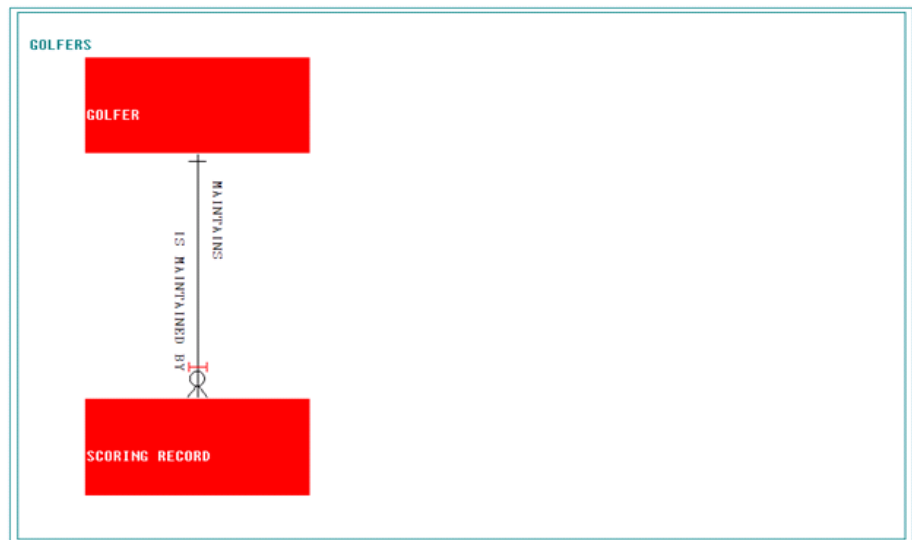
In the data model the following steps will be performed:

- Resize the Golfers subject area.
- Add the Score Card entity type.
- Add the BLOB attribute to the Score Card entity type.
- Create a relationship between the Scoring Record and Score Card entity types.
- Define an Identifier for the Score Card entity type.
- Perform a Consistency Check on the data model.

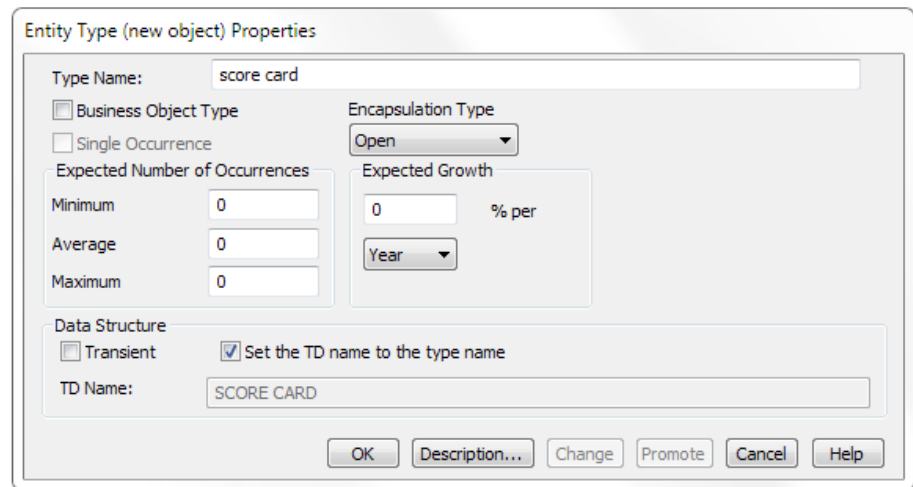
Follow these steps:

1. Open the Data Model. Click the **Golfers** subject area and expand it if necessary.
2. Resize the subject area to make room for adding the new entity type to the right of SCORING RECORD.

It should look similar to the following:



3. Click the **Golfers** subject area to gain focus on it and add a new entity type. Specify the properties as follows:



Entity Type (new object) Properties

Type Name:

☐ Business Object Type Encapsulation Type:

☐ Single Occurrence

Expected Number of Occurrences: Minimum Average Maximum

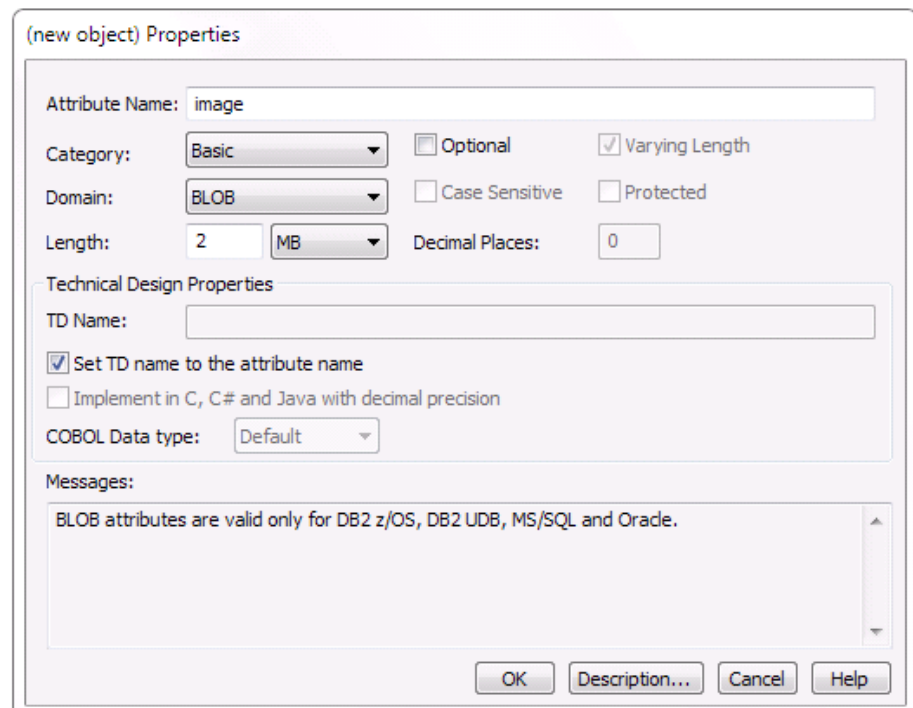
Expected Growth: % per

Data Structure: ☐ Transient ☒ Set the TD name to the type name

TD Name:

OK Description... Change Promote Cancel Help

4. Select the **SCORE CARD** entity type and add a BLOB attribute to it. Set the properties as follows:



(new object) Properties

Attribute Name:

Category: ☐ Optional ☒ Varying Length

Domain: ☐ Case Sensitive ☐ Protected

Length: Decimal Places:

Technical Design Properties

TD Name:

☒ Set TD name to the attribute name

☐ Implement in C, C# and Java with decimal precision

COBOL Data type:

Messages:

BLOB attributes are valid only for DB2 z/OS, DB2 UDB, MS/SQL and Oracle.

OK Description... Cancel Help

5. Select the **SCORING RECORD** entity type, then holding the Ctrl-key select the **SCORE CARD** entity type. Both entity types must be highlighted. Join the two with the following relationship:

Relationship Properties

Each: SCORING_RECORD
Sometimes is documented with
One SCORE_CARD

☒ Display relationship name
☐ Transferable Relationship

ASSOCIATE is
☒ Modifying
☐ Referencing

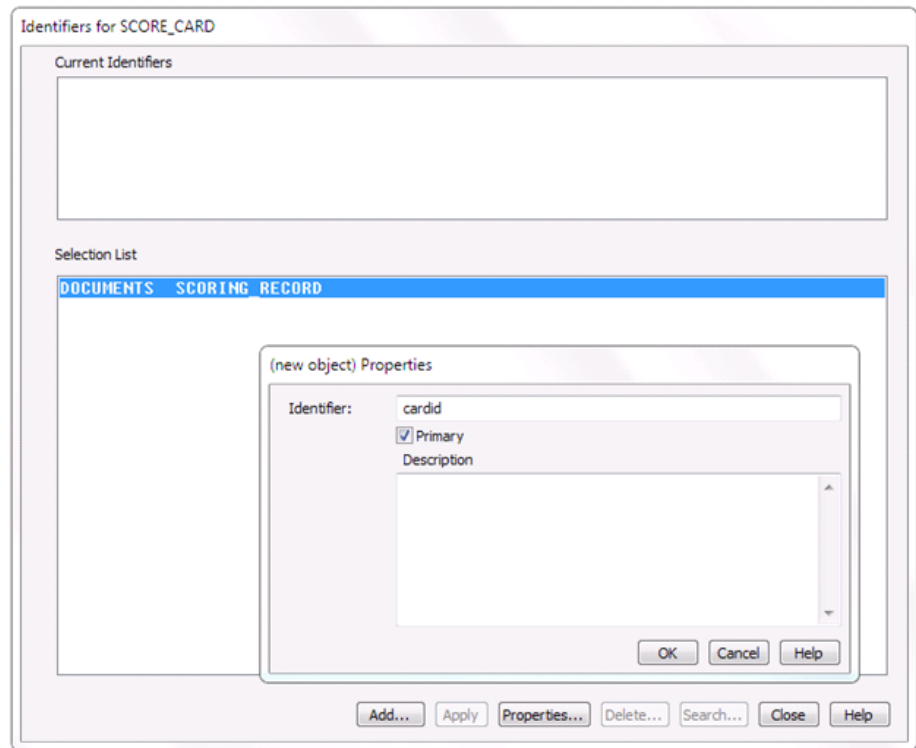
Each: SCORE_CARD
Always documents
One SCORING_RECORD

☒ Display relationship name
☐ Transferable Relationship

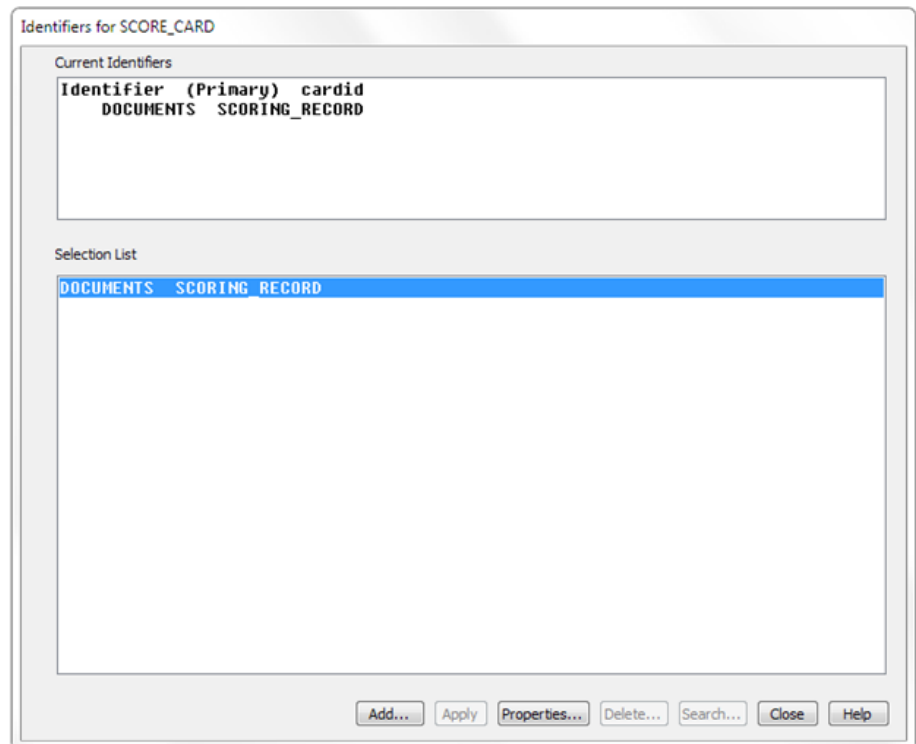
☐ Protected ☐ Transient

OK Source Properties... Destination Properties... Cancel Help

6. Because the relationship between SCORING RECORD and SCORE CARD is a one-to-one relationship, the Identifier for the SCORE CARD can be the relationship to its SCORING RECORD. Select the **SCORE CARD** entity type. From the Menu Bar select **Detail, Identifiers** to bring up the list of identifiers currently defined for SCORE CARD. Currently there are none. Select the relationship **DOCUMENTS SCORING_RECORD** and click the **Add** push button to add an identifier with the properties as follows:



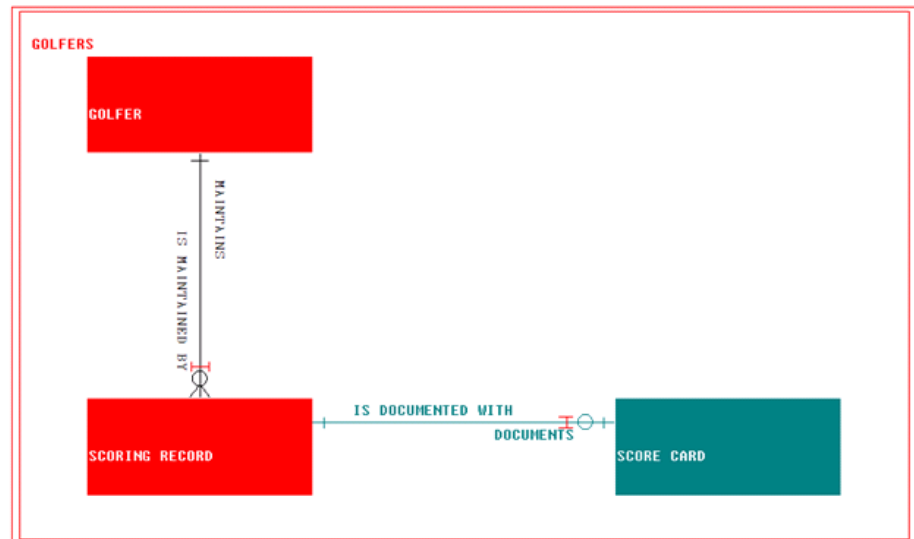
7. When you are finished, the Identifiers for SCORE_CARD will look as follows:



8. Click the **Close** push button.

The completed Data Model should now look as follows.

9. Select the new entity type and relationship and run a Consistency Check on them.
You should receive no Errors.
10. If you have not done so lately, save the model.



Activity Model

In the activity model, the following steps will be performed:

- Update the Activity Hierarchy Diagram to account for the elementary processes required to maintain the new entity type.
- Utilize Process Synthesis to create a first cut at the logic for each process action diagram.
- Review and modify each generated action diagram as necessary.
- Perform a Consistency Check on the activity model.

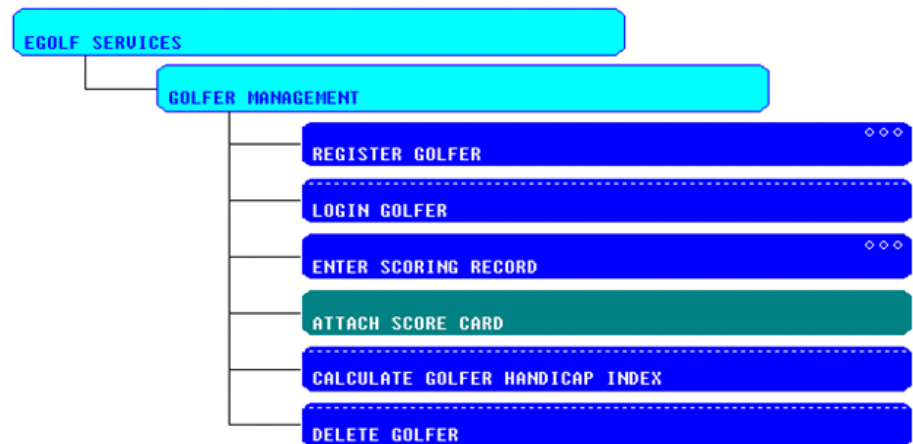
Follow these steps:

1. Open the Activity Hierarchy Diagram. Expand the **GOLFER MANAGEMENT** function. With **GOLFER MANAGEMENT** selected, add a new **Process** with the properties as follows:

The screenshot shows a dialog box titled "(new object) Properties". It contains the following fields and controls:

- Process Name:** A text field containing "attach score card".
- Elementary:** A checkbox that is currently unchecked.
- Repetitive:** A checkbox that is currently unchecked.
- Suggested Mechanism:** A group box containing four radio buttons: "Online" (selected), "Manual", "Batch", and "Other".
- Operation Properties:** A section containing:
 - Operation of:** A dropdown menu showing "<<NONE>>".
 - Choose Entity Type...** A button.
 - Protected Operation:** A checkbox that is currently unchecked.
 - Instance Operation:** A radio button that is currently unselected.
 - Type Operation:** A radio button that is currently selected.
- Description:** A large text area for entering a description.
- Buttons:** "OK", "Cancel", and "Help" buttons at the bottom right.

2. Select and drag the new **ATTACH SCORE CARD** process positioning it in the hierarchy as follows:



3. We need elementary processes to create, update, and delete SCORE CARDS. But rather than add them manually, we are going to use Process Synthesis to create them. Process Synthesis is much like Action Block Synthesis that we used back in Chapter Two of the Tutorial, but Process Synthesis will create multiple process action diagrams at the same time.
4. Select the ATTACH SCORE CARD process. From the Menu Bar select Generate, Process Synthesis.
5. In the Process Synthesis dialog select **SCORE_CARD**. Clear the selection of **Review Default Actions** and **Consistency Check on Entity Type**. Rename **CREATE_SCORE_CARD** to **ADD_SCORE_CARD**. Clear the selection of the **Populate** checkboxes for the last two Candidate Processes. Your completed dialog must look as follows:

Process Synthesis

Entity

Subject Area	EGOLF_SERVICES
Subject Area	CONSUMED_COMPONENTS
Subject Area	SPECIFICATION
Subject Area	EGOLF_SERVICES_APPLICATION
Subject Area	GOLFERS
Entity Type	GOLFER
Entity Type	SCORE_CARD
Entity Type	SCORING_RECORD

Action Dialog Option

☐ Review Default Actions

☐ Consistency Check on Entity Type

☒ Automatic Attribute View Selections

Candidate Process List

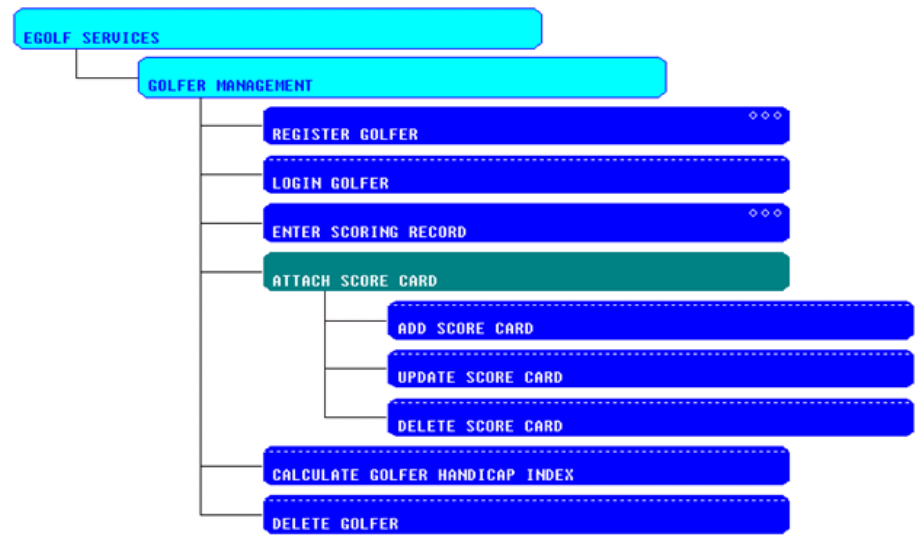
ADD_SCORE_CARD	<input checked="" type="checkbox"/>
UPDATE_SCORE_CARD	<input checked="" type="checkbox"/>
DELETE_SCORE_CARD	<input checked="" type="checkbox"/>
READ_SCORE_CARD	<input type="checkbox"/>
LIST_SCORE_CARD	<input type="checkbox"/>

Populate:

OK Cancel Help

Click the OK button.

- Elementary Processes have been created to add, update, and delete SCORE_CARDS. Expand the ATTACH SCORE CARD process. Your hierarchy will now look as follows:



7. The generated process action diagrams need a little cleanup. Open the ADD SCORE CARD action diagram. Expand the views. While all of the import views are required, the export views contain no new information and can be removed. However, before removing the views we need to remove the statements referencing the views. Highlight the two MOVE statements and delete them as follows.

```

ADD_SCORE_CARD
IMPORTS:
  Entity View import score_card (mandatory,transient,import only)
    image (mandatory)
  Entity View import golfer (mandatory,transient,import only)
    userid (mandatory)
  Entity View import scoring_record (mandatory,transient,import only)
    date (mandatory)
    time (mandatory)
EXPORTS:
  Entity View export score_card (transient,export only)
    image
  Entity View export scoring_record (transient,export only)
    date
    time
LOCALS:
ENTITY ACTIONS:
  Entity View score_card
    image
  Entity View golfer
    userid
  Entity View scoring_record
    date
    time

  READ scoring_record
    WHERE DESIRED scoring_record date IS EQUAL TO import scoring_record date
    AND DESIRED scoring_record time IS EQUAL TO import scoring_record time
    AND DESIRED scoring_record is_maintained_by SOME golfer
    AND THAT golfer userid IS EQUAL TO import golfer userid
  WHEN successful
    MOVE scoring_record TO export scoring_record
    CREATE score_card
    ASSOCIATE WITH scoring_record WHICH is_documented_with IT
    SET image TO import score_card image
    WHEN successful
    MOVE score_card TO export score_card
    WHEN already exists
    EXIT STATE IS score_card_ae
    WHEN permitted value violation
    EXIT STATE IS score_card_pv
  WHEN not found
    EXIT STATE IS scoring_record_nf

```

8. Delete the export views.

The completed ADD SCORE CARD action diagram will look as follows:

```

ADD SCORE_CARD
IMPORTS:
  Entity View import score_card (mandatory,transient,import only)
    image (mandatory)
  Entity View import golfer (mandatory,transient,import only)
    userid (mandatory)
  Entity View import scoring_record (mandatory,transient,import only)
    date (mandatory)
    time (mandatory)
EXPORTS:
LOCALS:
ENTITY ACTIONS:
  Entity View score_card
    image
  Entity View golfer
    userid
  Entity View scoring_record
    date
    time

  READ scoring_record
    WHERE DESIRED scoring_record date IS EQUAL TO import scoring_record date
    AND DESIRED scoring_record time IS EQUAL TO import scoring_record time
    AND DESIRED scoring_record is_maintained_by SOME golfer
    AND THAT golfer userid IS EQUAL TO import golfer userid

  WHEN successful
    CREATE score_card
    ASSOCIATE WITH scoring_record WHICH is_documented_with IT
    SET image TO import score_card image
  WHEN successful
  WHEN already exists
    EXIT STATE IS score_card_ae
  WHEN permitted value violation
    EXIT STATE IS score_card_pv

  WHEN not found
    EXIT STATE IS scoring_record_nf

```

9. Open the UPDATE SCORE CARD action diagram and expand the views. Again, the export views are not necessary. Delete the MOVE statement and then delete the export view. The completed diagram will look as follows:

```

UPDATE_SCORE_CARD
IMPORTS:
  Entity View import score_card (mandatory,transient,import only)
    image (mandatory)
  Entity View import golfer (mandatory,transient,import only)
    userid (mandatory)
  Entity View import scoring_record (mandatory,transient,import only)
    date (mandatory)
    time (mandatory)
EXPORTS:
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
    userid
  Entity View scoring_record
    date
    time
  Entity View score_card
    image

  READ score_card
    WHERE DESIRED score_card documents SOME scoring_record
    AND THAT scoring_record date IS EQUAL TO import scoring_record date
    AND THAT scoring_record time IS EQUAL TO import scoring_record time
    AND THAT scoring_record is_maintained_by SOME golfer
    AND THAT golfer userid IS EQUAL TO import golfer userid

  WHEN successful
    UPDATE score_card
    SET image TO import score_card image
  WHEN successful
  WHEN not unique
  EXIT STATE IS score_card_nu
  WHEN permitted value violation
  EXIT STATE IS score_card_pv

  WHEN not found
  EXIT STATE IS score_card_nf

```

10. Finally, open the DELETE SCORE CARD action diagram and expand the views. Once again, the export view is not necessary and should be removed. Delete the MOVE statement and then delete the export view. The completed diagram will look as follows:

```
DELETE_SCORE_CARD
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
    userid (mandatory)
  Entity View import scoring_record (mandatory,transient,import only)
    date (mandatory)
    time (mandatory)
EXPORTS:
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
    userid
  Entity View scoring_record
    date
    time
  Entity View score_card
    image

  READ score_card
    WHERE DESIRED score_card documents SOME scoring_record
    AND THAT scoring_record date IS EQUAL TO import scoring_record date
    AND THAT scoring_record time IS EQUAL TO import scoring_record time
    AND THAT scoring_record is_maintained_by SOME golfer
    AND THAT golfer userid IS EQUAL TO import golfer userid
  WHEN successful
  DELETE score_card
  WHEN not found
  EXIT STATE IS score_card_nf
```

11. Save the model. Select the **EGOLF SERVICES** function and perform a Consistency Check.

You may receive several Warnings and Errors this time. The errors will be referencing the fact that the new entity type, attribute, and relationship are not yet in the Technical Design. This will be fixed in a later section.

Design

Server Design

We need to create a server procedure to support the SCORE CARD elementary processes. So in the Window Navigation Diagram we need to:

- Select the appropriate Window Navigation Diagrams.
- Add the server procedure.
- Add logic to the server procedure to call the appropriate elementary processes.

Follow these steps:

1. Open the Window Navigation Diagram. From the Menu Bar select **Diagram, Open**. Open the EGOLF APPLICATION SERVER diagram. It will look similar to the image shown as follows:



2. Add a new server procedure step with the properties shown as follows:

The screenshot shows the 'Server/Procedure Step Properties' dialog box. It has a title bar 'Server/Procedure Step Properties'. Inside, there are three sections: 'Business Systems' with a dropdown menu showing 'EGOLF_SERVICES'; 'Server Name' with a text field containing 'Maintain Score Card'; and 'Description' with a large empty text area. At the bottom right, there are three buttons: 'OK', 'Close', and 'Help'.

3. The diagram should now look as follows:



4. We now want to save this “new” diagram. From the Menu Bar select **Diagram, Create Diagram**. and click **EGOLF_APPLICATION_SERVER** from the Create Diagram dialog. Click **Save** and then click **Yes** when prompted if you want to replace it.
5. Open the MAINTAIN SCORE CARD procedure step action diagram. It will be empty. In addition to supporting the ADD, UPDATE, and DELETE SCORE CARD processes, it needs to support displaying a score card as well. We will use Action Block Synthesis to create the READ action, then go back and modify the diagram to include the ADD, UPDATE, and DELETE elementary processes.

While inside the empty action diagram, from the Menu Bar click **Generate, Action Block Synthesis**. In the Action Block Synthesis dialog select the **SCORE CARD** entity type. Clear the selection of the **Review Default Actions** and **Consistency Check on Entity Types** check boxes. Change the Mode to **Read**. The dialog will look as follows. Click **OK**.

Action Block Synthesis

Entity Type:

Subject Area	EGOLF_SERVICES
Subject Area	CONSUMED_COMPONENTS
Subject Area	SPECIFICATION
Subject Area	EGOLF_SERVICES_APPLICATION
Subject Area	GOLFERS
Entity Type	GOLFER
Entity Type	SCORE CARD
Entity Type	SCORING_RECORD

Action Dialog Option

☐ Review Default Actions

☐ Consistency Check on Entity Type

☒ Automatic Attribute View Selections

Mode: Read ▼

OK Cancel Help

6. Expand the views. The MAINTAIN SCORE CARD procedure step will now look as follows:


```

MAINTAIN SCORE_CARD
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
  userid (mandatory)
  Entity View import scoring_record (mandatory,transient,import only)
  date (mandatory)
  time (mandatory)
EXPORTS:
  Entity View export score_card (transient,export only)
  image
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
  userid
  Entity View scoring_record
  date
  time
  Entity View score_card
  image

  READ score_card
    WHERE DESIRED score_card documents SOME scoring_record
    AND THAT scoring_record date IS EQUAL TO import scoring_record date
    AND THAT scoring_record time IS EQUAL TO import scoring_record time
    AND THAT scoring_record is_maintained_by SOME golfer
    AND THAT golfer userid IS EQUAL TO import golfer userid
  WHEN successful
  MOVE score_card TO export score_card
  WHEN not found
  EXIT STATE IS score_card_nf

```

7. For procedure steps, particularly maintenance type server procedure steps, we typically need the import views and export views to match. The import view contains two views that are not in the export, and the export contains one view that is not in the import. So before we add the remaining statements that this action diagram needs, we want to modify the views such that they match. So all we want to do is copy the two imports to the export and the single export view to the import, and rename them in the process.

Click and drag the cursor over the import views highlighting both of them, select the **F8 key** or from the Menu Bar click **Edit, Copy** and select the **EXPORTS:** label. When prompted, in the Copy View dialog enter **EXPORT** for the New Names.

Then do the same for the **export score_card** copying it to the **IMPORTS** views and giving it the new name **IMPORT**. When finished, you will have an import golfer, import scoring_record, and import score_card and an export golfer, export scoring_record, and export score_card. The diagram will look as follows.

Note: You can move the views around (F7 key) if you would like them to match what is shown as follows:

```
MAINTAIN_SCORE_CARD
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
    userid (mandatory)
  Entity View import scoring_record (mandatory,transient,import only)
    date (mandatory)
    time (mandatory)
  Entity View import score_card (optional,transient,import only)
    image (optional)
EXPORTS:
  Entity View export golfer (transient,export only)
    userid
  Entity View export scoring_record (transient,export only)
    date
    time
  Entity View export score_card (transient,export only)
    image
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
    userid
  Entity View scoring_record
    date
    time
  Entity View score_card
    image

  READ score_card
    WHERE DESIRED score_card documents SOME scoring_record
    AND THAT scoring_record date IS EQUAL TO import scoring_record date
    AND THAT scoring_record time IS EQUAL TO import scoring_record time
    AND THAT scoring_record is maintained_by SOME golfer
    AND THAT golfer userid IS EQUAL TO import golfer userid
  WHEN successful
  MOVE score_card TO export score_card
  WHEN not found
  EXIT STATE IS score_card_nf
```

8. Next add the CASE OF COMMAND statement to CREATE, UPDATE, DELETE, and DISPLAY the SCORE CARD. Then move the READ score_card statement to the DISPLAY section of the CASE statement. The diagram will look as follows:

```

MAINTAIN_SCORE_CARD
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
    userid (mandatory)
  Entity View import scoring_record (mandatory,transient,import only)
    date (mandatory)
    time (mandatory)
  Entity View import score_card (optional,transient,import only)
    image (optional)
EXPORTS:
  Entity View export golfer (transient,export only)
    userid
  Entity View export scoring_record (transient,export only)
    date
    time
  Entity View export score_card (transient,export only)
    image
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
    userid
  Entity View scoring_record
    date
    time
  Entity View score_card
    image

CASE OF COMMAND
CASE CREATE
CASE UPDATE
CASE DELETE
CASE DISPLAY
  READ score_card
    WHERE DESIRED score_card documents SOME scoring_record
    AND THAT scoring_record date IS EQUAL TO import scoring_record date
    AND THAT scoring_record time IS EQUAL TO import scoring_record time
    AND THAT scoring_record is_maintained_by SOME golfer
    AND THAT golfer userid IS EQUAL TO import golfer userid
  WHEN successful
    MOVE score_card TO export score_card
  WHEN not found
    EXIT STATE IS score_card_nf
OTHERWISE

```

9. Next we can add the USE statements to the ADD SCORE CARD, UPDATE SCORE CARD, and DELETE SCORE CARD elementary processes, matching the Import views of the process action diagrams to the Import views of the procedure step action diagram. The CASE OF COMMAND statement should now look as follows:

```

CASE OF COMMAND
CASE CREATE
  USE add_score_card
    WHICH IMPORTS: Entity View import score_card TO Entity View import score_card
                  Entity View import golfer TO Entity View import golfer
                  Entity View import scoring_record TO Entity View import scoring_record
CASE UPDATE
  USE update_score_card
    WHICH IMPORTS: Entity View import score_card TO Entity View import score_card
                  Entity View import golfer TO Entity View import golfer
                  Entity View import scoring_record TO Entity View import scoring_record
CASE DELETE
  USE delete_score_card
    WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
                  Entity View import scoring_record TO Entity View import scoring_record
CASE DISPLAY
  READ score_card
    WHERE DESIRED score_card documents SOME scoring_record
    AND THAT scoring_record date IS EQUAL TO import scoring_record date
    AND THAT scoring_record time IS EQUAL TO import scoring_record time
    AND THAT scoring_record is_maintained_by SOME golfer
    AND THAT golfer userid IS EQUAL TO import golfer userid
  WHEN successful
    MOVE score_card TO export score_card
  WHEN not_found
    EXIT STATE IS score_card_nf
OTHERWISE

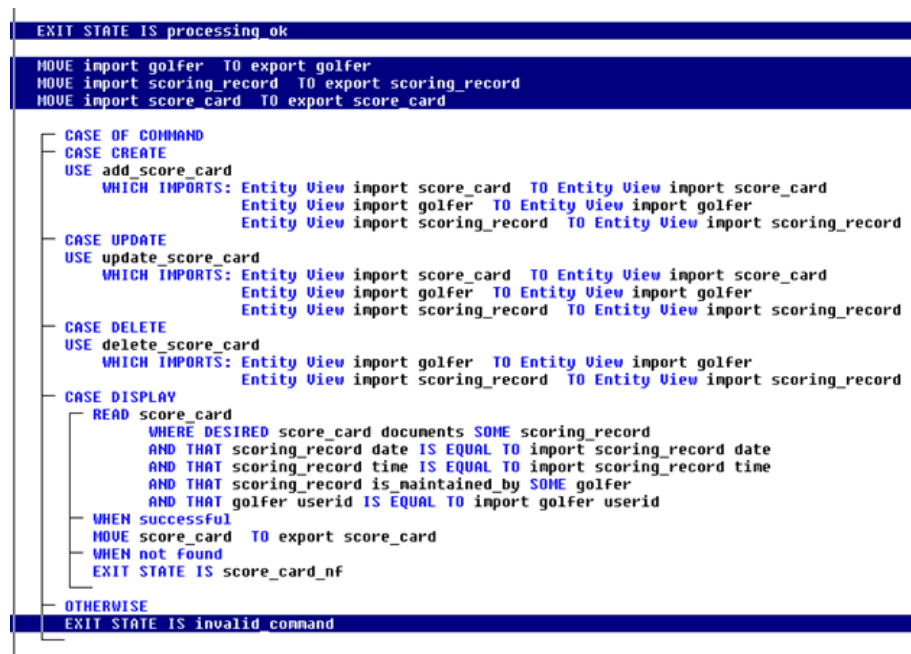
```

10. There are just a few more things we need to do to finish the server procedure step. We need to set an “initializing” exit state of PROCESSING OK at the beginning of the diagram, move all import views to export views, and set an INVALID COMMAND exit state for the OTHERWISE condition on the CASE OF COMMAND statement. The completed diagram will now look as follows:

```

MAINTAIN_SCORE_CARD
IMPORTS:
  Entity View import golfer (mandatory,transient,import only)
    userid (mandatory)
  Entity View import scoring_record (mandatory,transient,import only)
    date (mandatory)
    time (mandatory)
  Entity View import score_card (optional,transient,import only)
    image (optional)
EXPORTS:
  Entity View export golfer (transient,export only)
    userid
  Entity View export scoring_record (transient,export only)
    date
    time
  Entity View export score_card (transient,export only)
    image
LOCALS:
ENTITY ACTIONS:
  Entity View golfer
    userid
  Entity View scoring_record
    date
    time
  Entity View score_card
    image

```



11. Save the model.

Client Design

We need to create a client procedure to support the visualization and maintenance of the SCORE CARD. So in the Window Navigation Diagram the following steps will be performed:

- Add the client procedure.
- Add the OCX control to the client procedure's primary dialog box.
- Add logic to the client procedure to read/display the score card image as well as call the server procedure with the appropriate Commands.
- Modify the eGolfer Home Page to Link to this new dialog in order to attach the score card.

Review the client processing

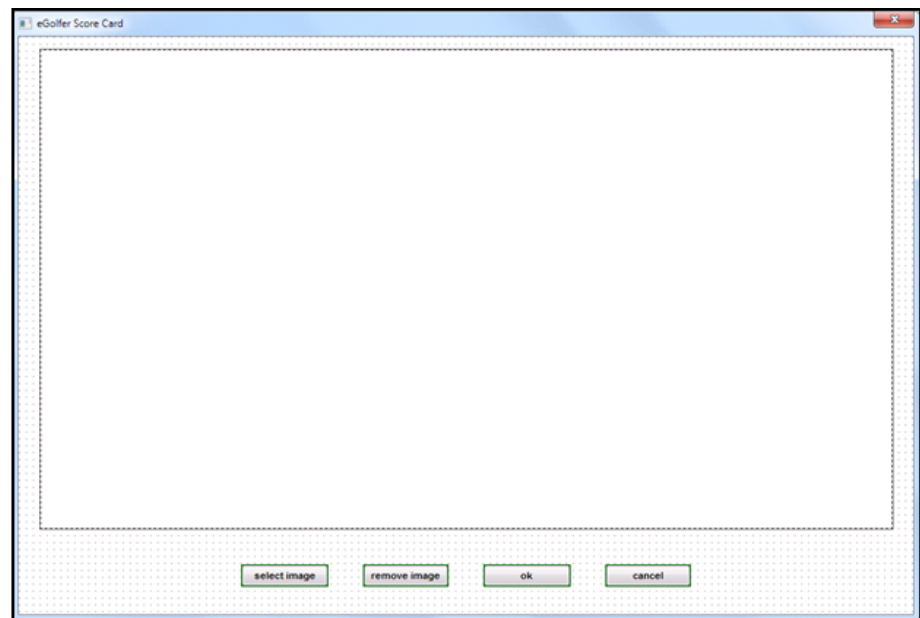
A new push button will be added to the eGolfer Home page with the label “score card”. It will be associated to a new click-event. When a Scoring Record is highlighted the push button will be enabled. Clicking it will allow the click-event logic to determine the highlighted row and to pass that information to the new Score Card client.

The modified eGolfer Home page will appear as follows:

The screenshot shows the eGolfer Home page. At the top is a green banner with the text "Place Your Ad Here" in yellow. Below this is the "Personal Profile" section, which includes a "Welcome" message and "Your Handicap Index is ZZ9.9". To the left of the profile information are buttons for "update" and "remove". Below the profile section is the "Scoring Record" section, which contains a table with columns: Date, Time, Score, Rating, Slope, and Note. The table has 15 rows of placeholder data (MM/DD/YY, HH:MM, ZZZ, ZZ.Z, ZZZ, XXXXXX). To the left of the table are buttons for "add", "update", "score card" (circled in red), and "remove". At the bottom right of the page is a "logout" button.

The new eGolfer Score Card page will contain an OCX control area and four push buttons. The buttons will allow you to select a new image, remove an image, and click ok to save your changes, or cancel your changes. Clicking ok or cancel will return you to the eGolfer Home page.

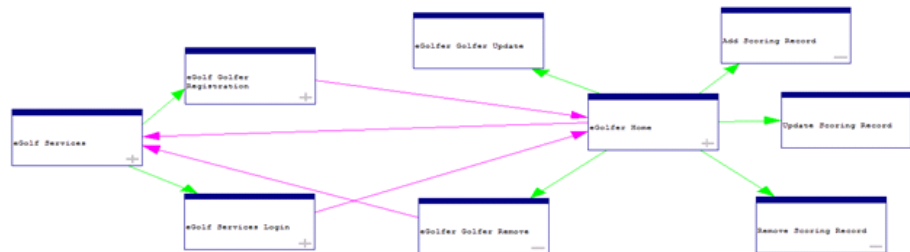
The new eGolfer Score Card page will appear as follows:



Add the client procedure

Follow these steps:

1. Open the Window Navigation Diagram.
2. From the Menu Bar click **Diagram, Open**. Open the **EGOLF WEB SERVER** diagram. It will look similar to the following image:



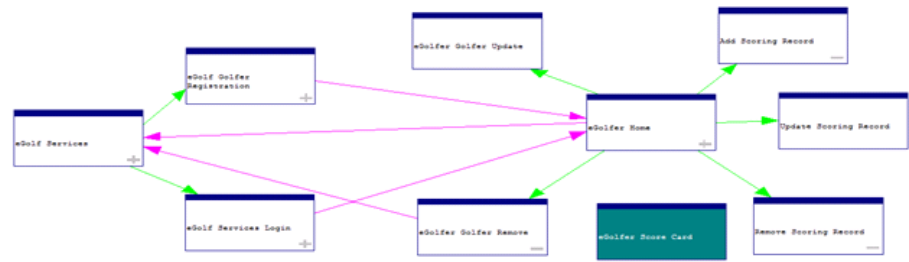
3. Add a Window to the Navigation Diagram, below the eGolfer Home window. Set the properties as follows:

Note: Be sure to select the Type of **Primary Dialog Box**.

Window / Dialog Box Properties

Type <input type="radio"/> Primary Window <input checked="" type="radio"/> Primary Dialog Box <input type="radio"/> Dialog Box <input type="radio"/> Help About Box	Modality <input checked="" type="radio"/> Application Modal <input type="radio"/> Modeless, Single Instance <input type="radio"/> Modeless, Multiple Instances	Initial Position <input type="radio"/> Designed Position <input type="radio"/> Mouse Alignment <input checked="" type="radio"/> System Placed
Background <input type="radio"/> Tiled Bitmap: <NONE> <input checked="" type="radio"/> Scaled <input type="radio"/> Centered HTML Extension:	Style <input checked="" type="checkbox"/> System Menu <input type="checkbox"/> Minimize Button <input type="checkbox"/> Maximize Button <input checked="" type="checkbox"/> Dialog Border	
Title: eGolfer Score Card Icon File: Name: egolfer score card Dialect: DEFAULT		
<input checked="" type="checkbox"/> Copy Default Window on Create Current Business System EGOLF_SERVICES		
OK Bitmaps... Description... Cancel Help		

4. The Window Navigation Diagram will now look as follows:

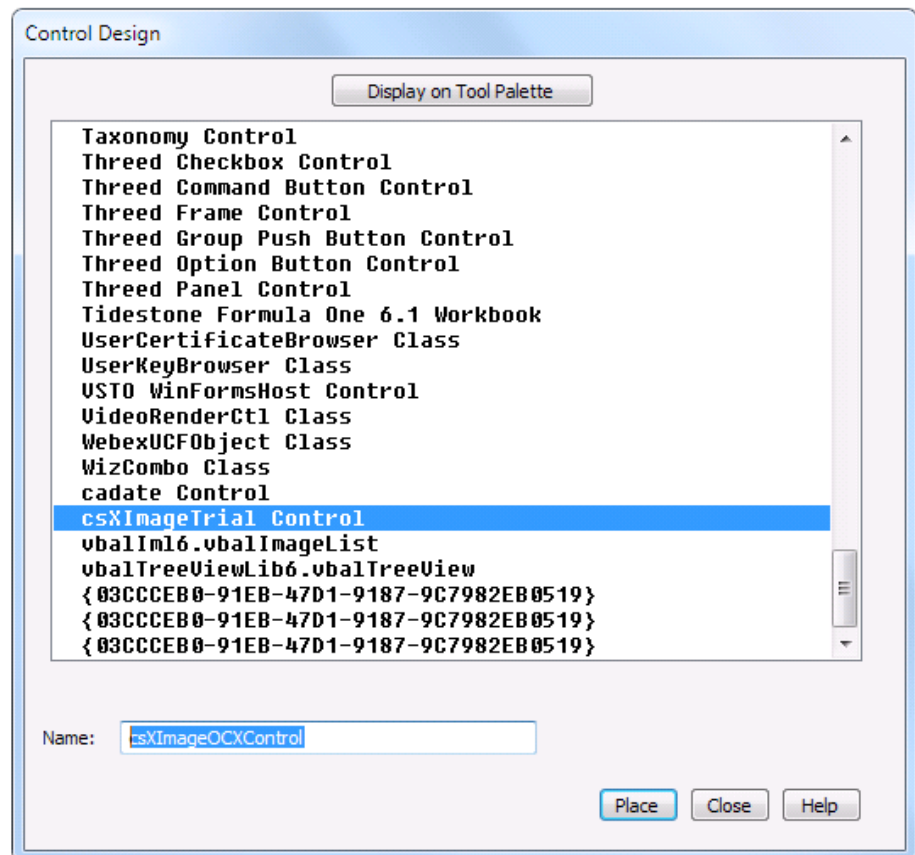


5. We now want to save this “new” diagram. From the Menu Bar click **Diagram**, **Create Diagram** and select **EGOLF_WEB_SERVER** from the Create Diagram dialog. Click **Save** and then click **Yes** when prompted if you want to replace it.

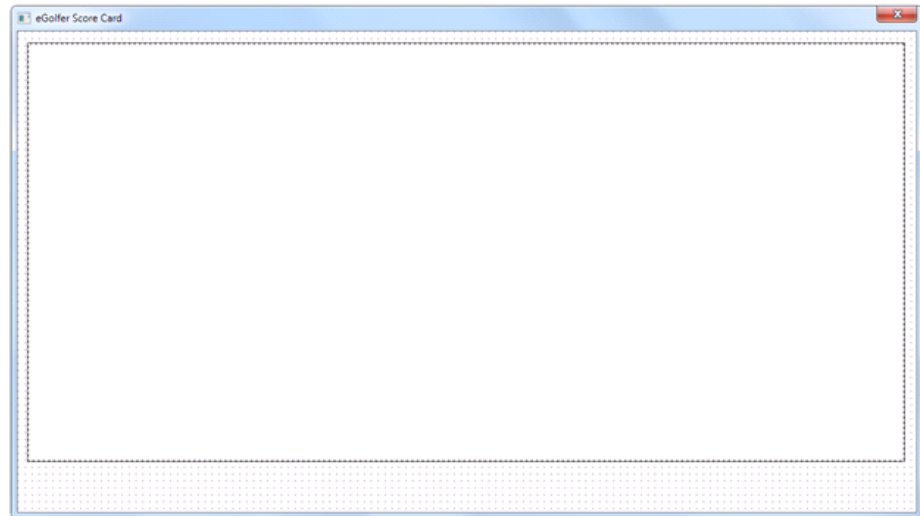
Layout the window design

Add the OCX control

1. Double-click the **eGolfer Score Card** window in the Window Navigation Diagram to open the window designer.
2. Click the **eGolfer Score Card** titlebar to gain focus on the window. From the Menu Bar click **Edit, Control Design**.
3. In the Control Design dialog scroll down and select the **csXImageTrial Control** and change the default name for the control to **csXImageOCXControl** as follows:



4. Select the **Place** push button and place it on the window. Resize the window borders and the control borders so that the window looks somewhat like the following example. Leave room at the bottom for adding the four push buttons shown in the example earlier.

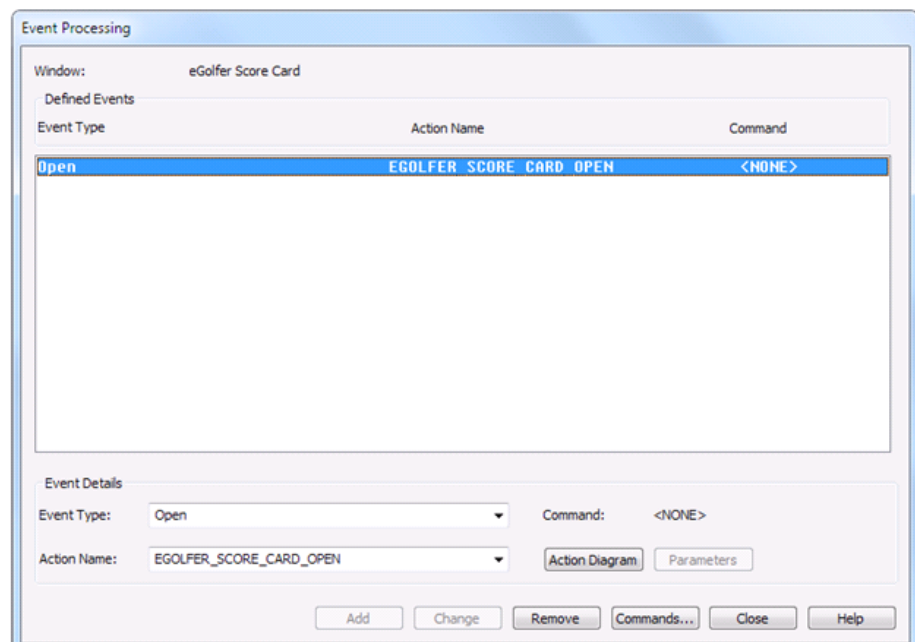


Add the window open event

Follow these steps:

When this window opens, if there is a SCORE CARD already associated with the SCORING RECORD, we want it to be displayed. To make this happen we will add an OPEN event to the window. When the window is opened, it will trigger the OPEN event and inside that event we will place the logic necessary to go and get the image if it exists.

1. Select the **eGolfer Score Card** title bar to get focus on the window.
2. From the Menu Bar select **Detail, Events**. In the Event Processing dialog change the Event Type dropdown to **Open**. Click **Add**. The event is created and added to the list of Event Types associated with this window. The Event Processing dialog will look as follows:



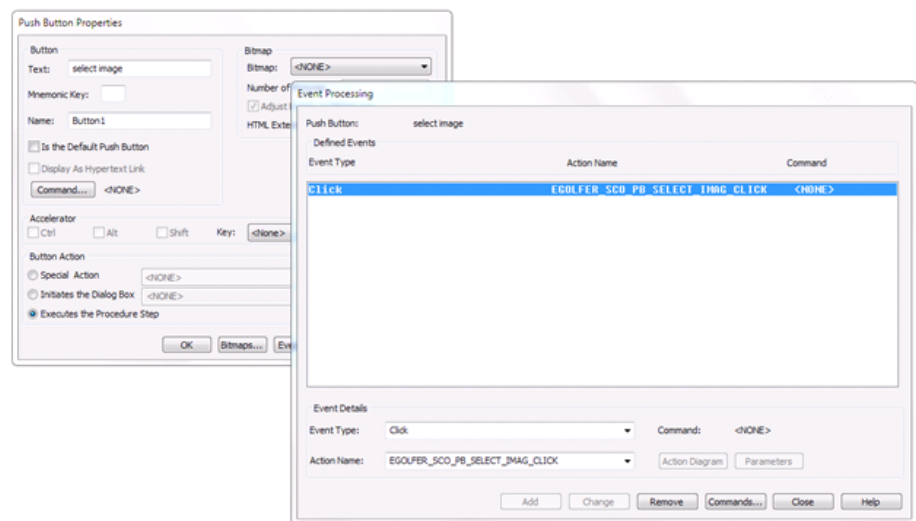
3. On the Event Processing dialog click **Close**.

Add the pushbuttons and define their actions

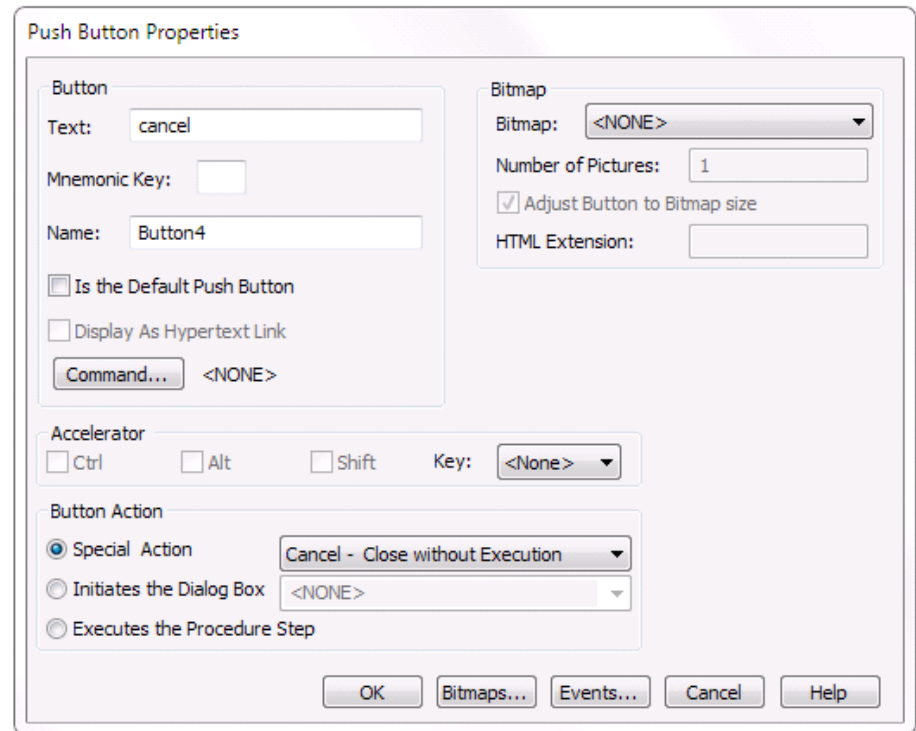
Follow these steps:

Next we will add the four push buttons. Three of the four push buttons will have click events associated with them. The fourth push button, cancel, will use a special action.

1. From the Menu Bar, click **Add, Push Button**.
2. In the Button Text field enter **select image**. By default, it Executes the Procedure Step. Click **Events**.
3. In the Event Processing dialog click **Add** to have it create the Click event. It will be added to the list of events associated with this push button as follows:



4. Click **Close** on the Event Processing dialog and then **OK** on the Push Button Properties dialog and place the push button roughly in the location desired.
5. Repeat the steps above for the “**remove image**” push button.
6. Repeat the steps above for the “**ok**” push button.
7. For the “**cancel**” push button, on the Push Button Properties dialog for the Button Action property select **Special Action** and in the dropdown select **Cancel – Close without Execution**. The push button properties will look as follows:

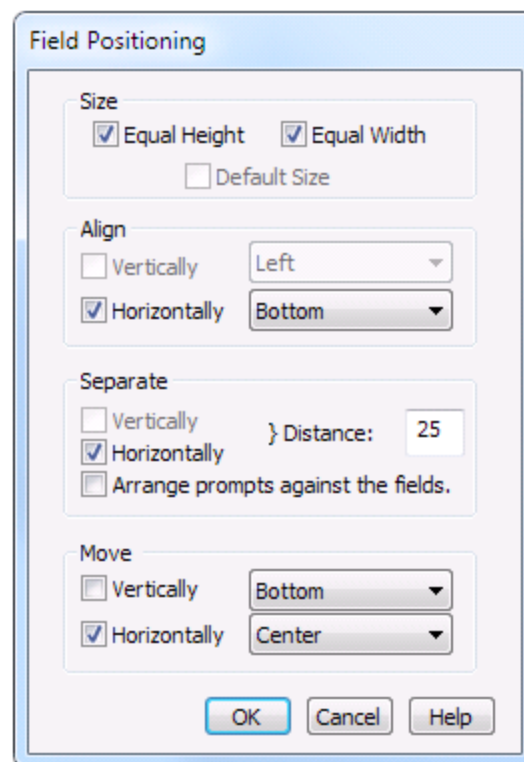


8. Select the **OK** push button and place it appropriately.

Position the pushbuttons

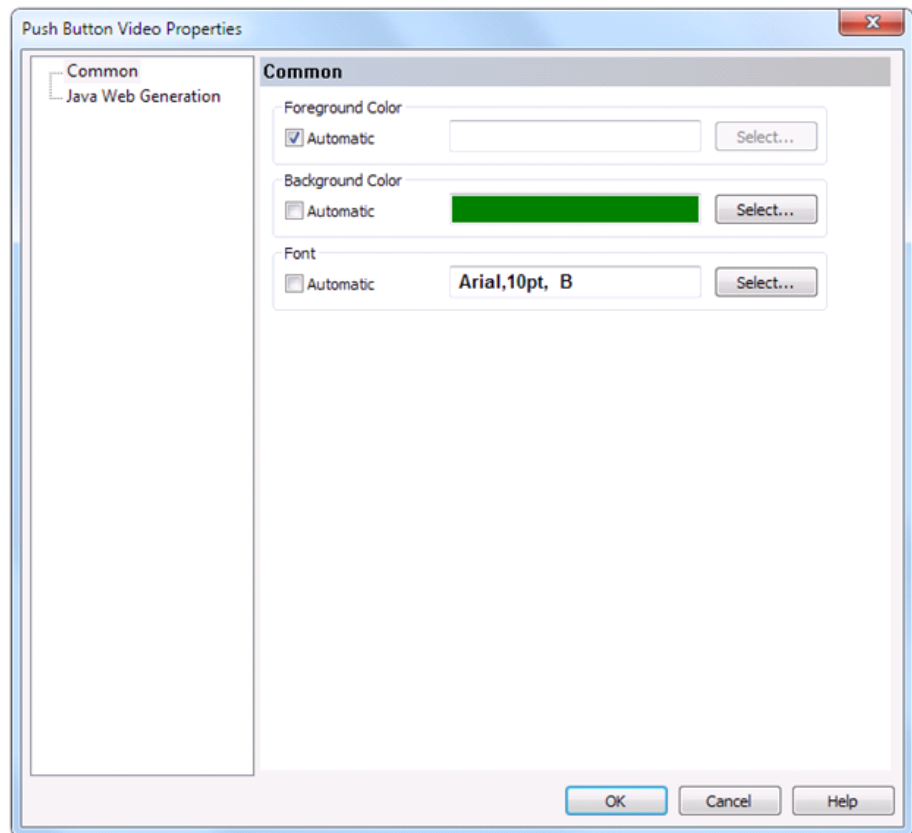
Follow these steps:

1. You will now have four push buttons along the bottom of the window. We want to make them look a little nicer and distribute them evenly across the bottom of the window. Select the button on the bottom left, which should be the **select image** push button. By default, its height is about 32. You can look in the bottom right side of the Window Navigation Diagram and it gives the coordinates along with its size. Resize the height to about 26.
2. With focus still on that push button, use the **Ctrl** key and click the other 3 push buttons. All four should show they are selected. From the Menu Bar click **Edit, Position**.
3. In the Field Positioning dialog select **Equal Height** and **Equal Width**, **Align Horizontally** (along the **Bottom**), **Separate Horizontally** and enter the Distance of **25**, and **Move Horizontally** and select **Center** from the dropdown. The Field Positioning dialog will look as follows:



4. Click **OK**.

The four push buttons will still be highlighted, sized equally, and placed evenly across the bottom of the window. With the four push buttons still highlighted, from the Menu Bar click **Detail, Video Properties** and clear the selection of the Automatic options for both **Background Color** and **Font**. The Video Properties dialog will look as follows:



5. Click **OK**.

Your window will now look similar to the following image:



Add the client logic

Rename the events

Follow these steps:

1. Open the eGolfer Score Card action diagram. The diagram will look as follows:

```

EGOLFER_SCORE_CARD
IMPORTS:
EXPORTS:
LOCALS:
ENTITY ACTIONS:

EVENT ACTION egolfer_score_card_open
EVENT ACTION egolfer_sco_pb_select_imag_click
EVENT ACTION egolfer_sco_pb_remove_imag_click
EVENT ACTION egolfer_sco_pb_ok_click

```

The procedure step action diagram is empty and it is followed by four event actions. The first is the Open event, and it will contain the actions we want to take when the window is opened. The other three are the Click events and they will contain the actions we want to take when someone clicks on the “select image”, “remove image”, and “ok” push buttons respectively. There is no event associated with the “cancel” push button as it performs the Special Action Cancel – Close without Execution.

2. We suggest that you rename the event names to make them more readable. Double-click each event action and rename them as follows:

```

EGOLFER_SCORE_CARD
IMPORTS:
EXPORTS:
LOCALS:
ENTITY ACTIONS:

EVENT ACTION score_card_open
EVENT ACTION score_card_pb_select_image_click
EVENT ACTION score_card_pb_remove_image_click
EVENT ACTION score_card_pb_ok_click

```

Add logic to the “select image” click event

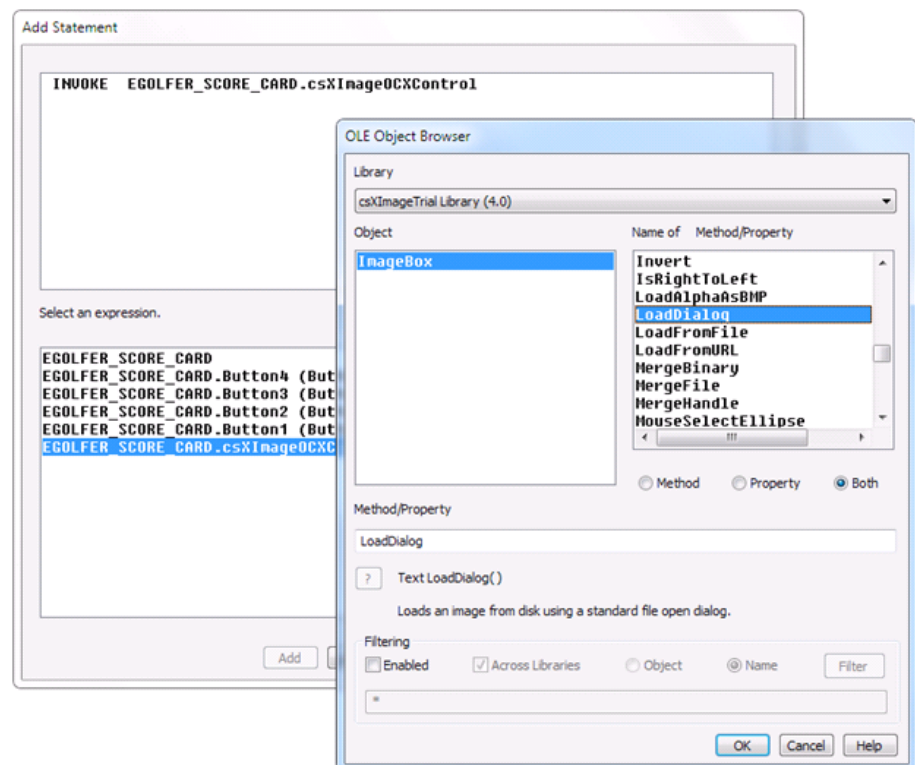
Follow these steps:

1. In addition to interacting with our Maintain Score Card server procedure step, much of what we are going to be doing here is interacting with the OCX Control in order to visualize the images. The OCX Control provides “methods” for us to do that. Two of our events require very little work on our part except to invoke the appropriate OCX Control method, so we will complete those first.

For selecting an image from a file and displaying it in the OCX Control, we need only invoke its LoadDialog method. It will then present a standard file dialog in which someone can select a supported image file from their file system in order to display it in the OCX Control.

In the eGolfer Score Card action diagram, select the **score_card_pb_select_image_click** event.

2. From the Menu Bar, click **Edit, Add Statement, Invoke**.
3. In the Add Statement dialog select **Window Object** and then **EGOLFER_SCORE_CARD.csXImageOCXControl (OLEControl)**. The OLE Object Browser dialog opens.
4. In the list of Methods scroll down and select the **LoadDialog** method as follows:



5. Click OK and then click the Add Statement Add push button. The statement will be added to the diagram.

Add logic to the “remove image” click event

Follow these steps:

1. To remove an image from the display, we need only invoke the OCX Controls Clear method.

In the eGolfer Score Card action diagram, select the **score_card_pb_remove_image_click** event.

2. Add the **Invoke** statement.
3. In the Add Statement dialog select **Window Object** and then **EGOLFER_SCORE_CARD.csXImageOCXControl (OLEControl)**. You will be presented with the OLE Object Browser dialog.
4. In the List of Methods scroll down and select the **Clear** method.
5. Then click **OK** and then click the Add Statement **Add** push button. Your diagram will now look as follows:

```

EGOLFER_SCORE_CARD
  IMPORTS:
  EXPORTS:
  LOCALS:
  ENTITY ACTIONS:

  EVENT ACTION score_card_open

  EVENT ACTION score_card_pb_select_image_click
  INVOKE EGOLFER_SCORE_CARD.csXImageOCXControl.LoadDialog()

  EVENT ACTION score_card_pb_remove_image_click
  INVOKE EGOLFER_SCORE_CARD.csXImageOCXControl.Clear()

  EVENT ACTION score_card_pb_ok_click
  
```

Add logic to the window “open” event

Follow these steps:

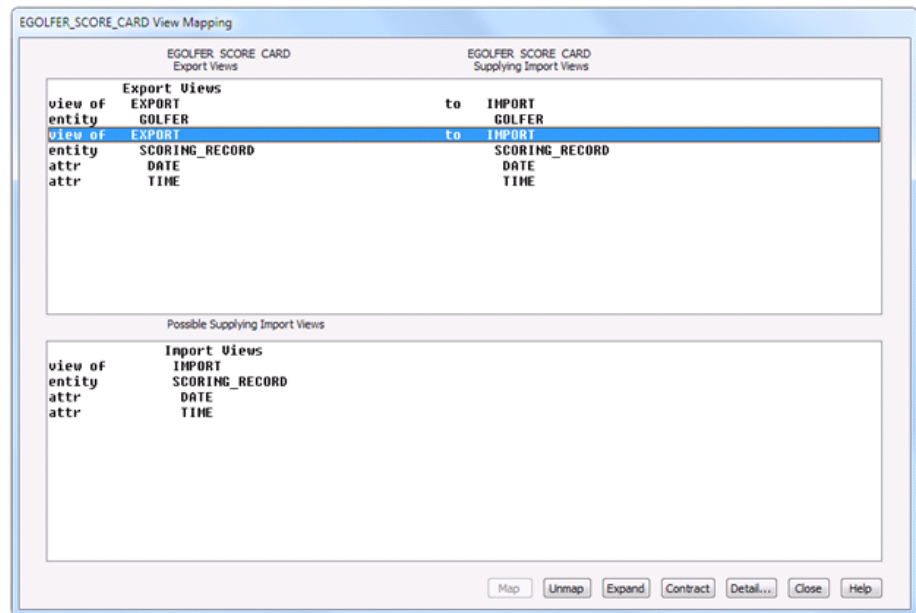
1. The next event we want to complete is the window Open event. As you may recall, this procedure step will be linked to from the eGolfer Home procedure step. The golfer will select a scoring record and click the Score Card push button which will cause a Link flow to this procedure step. As part of that flow, the golfer userid and the scoring record date and time are passed to this procedure step in order for it to read and display the score card, if it exists. So the import and export views of this procedure step need to contain entity views of the golfer and scoring record.

Add **IMPORTS** views to this procedure step containing the **golfer userid** and the **scoring record date** and **time**. Name the views **Import**. When complete, copy them to the **EXPORTS** views naming the new export views **Export**. Your diagram will now look as follows:

```
EGOLFER_SCORE_CARD
IMPORTS:
  Entity View import golfer (optional,transient,import only)
    userid (optional)
  Entity View import scoring_record (optional,transient,import only)
    date (optional)
    time (optional)
EXPORTS:
  Entity View export golfer (transient,export only)
    userid
  Entity View export scoring_record (transient,export only)
    date
    time
LOCALS:
ENTITY ACTIONS:

EVENT ACTION score_card_open
EVENT ACTION score_card_pb_select_image_click
  INVOKE EGOLFER_SCORE_CARD.csXImageOCXControl.LoadDialog()
EVENT ACTION score_card_pb_remove_image_click
  INVOKE EGOLFER_SCORE_CARD.csXImageOCXControl.Clear()
EVENT ACTION score_card_pb_ok_click
```

2. Now that we have the required import and export views, we want to make sure they are properly mapped. Return to the window design for the eGolfer Score Card. From the Menu Bar, click **Detail, Mapping**. The EGLOFER_SCORE_CARD View Mapping dialog is displayed. In the top panel select **EXPORT GOLFER**. In the bottom panel select **IMPORT GOLFER** and click the **Map** push button. Do the same for **EXPORT SCORING_RECORD**. When complete, the dialog will look as follows:



3. On the view mapping dialog click **Close**.
4. We will be displaying the score card image and normally that would imply an export view, but Gen does not provide UI support for BLOB attributes, which is why we are using the OCX control. But the OCX control still needs the BLOB data. So we will use a local view of the Score Card Image.

Back in the EGOLFER_SCORE_CARD procedure step action diagram create a LOCALS view of **Score Card Image** and name the view **Local**. Your diagram will now look as follows:

```

EGOLFER_SCORE_CARD
IMPORTS:
  Entity View import golfer (optional,transient,import only)
  userid (optional)
  Entity View import scoring_record (optional,transient,import only)
  date (optional)
  time (optional)
EXPORTS:
  Entity View export golfer (transient,export only)
  userid
  Entity View export scoring_record (transient,export only)
  date
  time
LOCALS:
  Entity View local score_card
  image
ENTITY ACTIONS:

EVENT ACTION score_card_open

EVENT ACTION score_card_pb_select_image_click
  INVOKE EGOLFER_SCORE_CARD.csXImageOCXControl.LoadDialog()

EVENT ACTION score_card_pb_remove_image_click
  INVOKE EGOLFER_SCORE_CARD.csXImageOCXControl.Clear()

EVENT ACTION score_card_pb_ok_click

```

5. Now we are ready to call the Maintain_Score_Card server procedure step. Select the **score_card_open** event and add the **Command Is** statement. From the Add Statement dialog select **command value** and select the **DISPLAY** command. Select the Add Statement **Add** push button to add the statement to the event handler.
6. Next add a **Procedure Step Use** statement, selecting the **maintain_score_card** procedure step. In the Import View Matching dialogs that follow, match the **import golfer** in the server MAINTAIN SCORE CARD to the **import golfer** from the client EGOLFER SCORE CARD.

Do the same for the import **scoring record** and click the **Close** button. In the Export View Matching dialog, you only need to match the **export score_card** from the server MAINTAIN SCORE CARD to the **local score_card** of the client EGOLFER SCORE CARD, then click the **Close** button. The Open event will now look as follows:

Note: Make sure the EXPORTS Entity View local score_card is matched from the Entity View export score_card as shown. If you need to adjust any of the view matching, double-click the view name that you want to change to bring up the view matching dialog again.

```
EVENT ACTION score_card_open
COMMAND IS DISPLAY
USE maintain_score_card (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
                Entity View import scoring_record TO Entity View import scoring_record
                <none> TO Entity View import score_card
  WHICH EXPORTS: <none> FROM Entity View export golfer
                <none> FROM Entity View export scoring_record
                Entity View local score_card FROM Entity View export score_card
```

7. If the server successfully reads an existing score_card image we want to display it using the OCX control. But we only want to try to display it if one was found. So we need to verify that the server completed successfully. Server logic usually sets the EXIT STATE variable to a known value, like PROCESSING_OK at the start of the processing and then sets it to another value if needed be when encountering a problem. So we want to check to see if the EXIT STATE is still equal to PROCESSING_OK.

Select the **USE** statement to highlight the entire USE statement. Add an **IF** action statement checking the value of the **EXIT STATE**. The Open event will now look as follows:

```
EVENT ACTION score_card_open
COMMAND IS DISPLAY
USE maintain_score_card (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
                Entity View import scoring_record TO Entity View import scoring_record
                <none> TO Entity View import score_card
  WHICH EXPORTS: <none> FROM Entity View export golfer
                <none> FROM Entity View export scoring_record
                Entity View local score_card FROM Entity View export score_card
IF EXITSTATE IS EQUAL TO processing ok
```

8. Now that we can determine if we successfully read a score_card image, actually visualizing it in the User Interface with the OCX Control is a two-step process. We first have to write the BLOB data out to a file, and then invoke the OCX Control passing it the file information.

Writing the BLOB data to a file requires the use of an External Action Block (EAB). An External Action Block is used to interface Gen generated code with developer handwritten code. An External Action Block is similar to every other action block in that it too can have import and export views, but the only action diagram statement allowed is EXTERNAL. The Gen application then “uses” the External Action Block like it would use any other action block, matching views to the External Action Blocks import and export views. The External Action Block is packaged and generated like other action blocks, but the only thing generated for it is a “stub” or “shell” of a program, containing definitions for the arguments used in the interface.

The developer would then add his own handwritten code to the stub, and then compile the code like he would any other handwritten code. If there were many EABs, the developer could decide to include them into a library. Finally, the location for the external action blocks would be added to a Build Tool profile so they could be located and linked into the Gen application as appropriate.

For writing our BLOB to a file, there are two pieces of information the EAB has to have, the BLOB data and the file information. So we are going to create an EAB that has two import views. There is probably information that should be returned to us as well, but for this example we will ignore that. So for this example our external action block will not have any export views.

For the two import views, the required BLOB data can be represented or defined by a view of the Score Card image. But we also need to pass file information, and we do not have anything defined in our model yet which represents file information. So we will create a new Work Set. A Work Set is similar to an entity type, in that it is a collection of “attributes”, but work sets do not appear in the data model, and subsequently do not get generated as data base tables. Essentially, Work Sets are lists of pre-defined arguments that are used in action diagrams to provide consistency in the parameter definitions.

To create a Work Set, from the Menu Bar click **Tool, Analysis, Work Set List**. A list of the current Work Sets in the model is displayed. Every model contains at least two work sets that are added automatically when you create the model. They are the ASYNC_REQUEST and IEF_SUPPLIED work sets. The IEF_SUPPLIED work set in particular is useful in that it contains definitions for many parameters commonly used in Gen applications, but there is nothing in it to represent our file information. So we will create a new work set.

9. From the Menu Bar click **Edit, Add Work Set**. Enter the name **eab workset** as follows and click **OK**.

The image shows a dialog box titled "(new object) Properties". It has two main sections: "Work Set" and "Description". The "Work Set" section has a text input field containing the text "eab workset". The "Description" section has a large, empty text area with a vertical scrollbar on the right. At the bottom right of the dialog box, there are three buttons: "OK", "Cancel", and "Help".

10. Next, select the **EAB_WORKSET** in the list and from the Menu Bar click **Edit, Add Attribute**. Enter the attribute name of **path**, the domain of **Text**, Length of **255**, and select the **Optional** checkbox as follows. Then click the **OK** button.

11. You will now be able to see your new Work Set added to the Work Set List.

Type	Name
Work Set	ASYNC_REQUEST ...
Work Set	IEF_SUPPLIED ...
Work Set	EAB_WORKSET
Attribute	PATH (Text, 255, Optional)

Close the Work Set List and you will be returned to the EGOLFER_SCORE_CARD action diagram.

12. Now that we are back in the action diagram, we want to create a **LOCALS** view of the **eab_workset path**. Name the local view **local**. Your local views will appear as follows:

```

EGOLFER_SCORE_CARD
IMPORTS: ...
EXPORTS: ...
LOCALS:
  Work View  local eab_workset
    path
  Entity View local score_card
    image
ENTITY ACTIONS:

```

13. Finally we can set the local view to the file location that we intend to have the EAB write the BLOB data to. Inside the **IF** statement, **set** the **local eab_workset path** to the **character string C:\temp\egolfblob.bin**. The Open event will look as follows:

```

EVENT ACTION score_card_open
COMMAND IS DISPLAY
USE maintain_score_card (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
                Entity View import scoring_record TO Entity View import scoring_record
                <none> TO Entity View import score_card
  WHICH EXPORTS: <none> FROM Entity View export golfer
                <none> FROM Entity View export scoring_record
                Entity View local score_card FROM Entity View export score_card
IF EXITSTATE IS EQUAL TO processing_ok
SET local eab workset path TO "c:\temp\egolfblob.bin"

```

14. Now we need to create the External Action Block. From the Menu Bar click **Diagram, New**. Name the new diagram “**eab write blob to file**” as shown in the following dialog. Click **OK**.

(new object) Properties

Action Block (BSD)

eab write blob to file

Description

Current Business System

EGOLF_SERVICES

OK Cancel Help

15. In the empty EAB_WRITE_BLOB_TO_FILE action diagram, select **IMPORTS** and add a work view of the **EAB_WORKSET PATH**. In the dropdown, mark the view as **Always** used as input. Name the view **Import**. Click **OK** to add the view to the diagram.
16. Add another **IMPORTS** view containing an entity view of the **SCORE_CARD IMAGE**. Mark the view as **Always** used as input. Name the view **Import**. Click **OK** to add the view to the diagram.
17. Expand the import views if necessary and double-click each attribute view and mark it as **Always** used as input.
18. Finally, add the single action diagram statement of **EXTERNAL**. The completed EAB must look as follows.

Note: It is important that the order of the Import views is exactly as shown below. That is because we have already written, compiled, and provided a library containing the external code and they need to match. If they are not, move them so that they are.

```

EVENT ACTION score_card_open
COMMAND IS DISPLAY
USE maintain_score_card (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
                Entity View import scoring_record TO Entity View import scoring_record
                <none> TO Entity View import score_card
  WHICH EXPORTS: <none> FROM Entity View export golfer
                <none> FROM Entity View export scoring_record
                Entity View local score_card FROM Entity View export score_card
  IF EXITSTATE IS EQUAL TO processing_ok
  SET local eab_workset path TO "c:\temp\egolfblob.bin"
  USE eab_write_blob_to_file
    WHICH IMPORTS: Entity View local score_card TO Entity View import score_card
                  Work View local eab_workset TO Work View import eab_workset
  INVOKE EGOLFER_SCORE_CARD.csXImageOCXControl.LoadFromFile(local eab_workset path)

```

19. Close the EAB_WRITE_BLOB_TO_FILE diagram and if you have not done so lately, save the model.
20. When we created the new external action block, the EGOLFER_SCORE_CARD procedure step action diagram was likely closed. If so, reopen the EGOLFER_SCORE_CARD procedure step action diagram.
21. In the Open event, now that we have called the server to return the BLOB into a local view, and set another local view to the path for the file, we can call the new EAB to write the BLOB data to the file. So after the SET statement, add a **USE EAB_WRITE_BLOB_TO_FILE** statement matching the EAB import views to the procedure steps local views.
22. The final thing we need to do is invoke the LoadFromFile method of the OCX Control to display a particular file. Add the **Invoke** statement.
23. In the Add Statement dialog select **Window Object** and then **EGOLFER_SCORE_CARD.csXImageOCXControl (OLEControl)**. The OLE Object Browser dialog opens.
24. In the list of Methods scroll down and select the **LoadFromFile** method and click **OK**.
25. Back in the Add Statement dialog select **character view, local eab_workset**, the **<Complete>** action, and the Add Statement **Add** push button. Your diagram will now look as follows.

```

EVENT ACTION score_card_open
COMMAND IS DISPLAY
USE maintain_score_card (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
                Entity View import scoring_record TO Entity View import scoring_record
                <none> TO Entity View import score_card
  WHICH EXPORTS: <none> FROM Entity View export golfer
                <none> FROM Entity View export scoring_record
                Entity View local score_card FROM Entity View export score_card
  IF EXITSTATE IS EQUAL TO processing_ok
  SET local eab_workset path TO "c:\temp\egolfblob.bin"
  USE eab_write_blob_to_file
    WHICH IMPORTS: Entity View local score_card TO Entity View import score_card
                  Work View local eab_workset TO Work View import eab_workset
  INVOKE EGOLFER_SCORE_CARD.csXImageOCXControl.LoadFromFile(local eab_workset path)

```

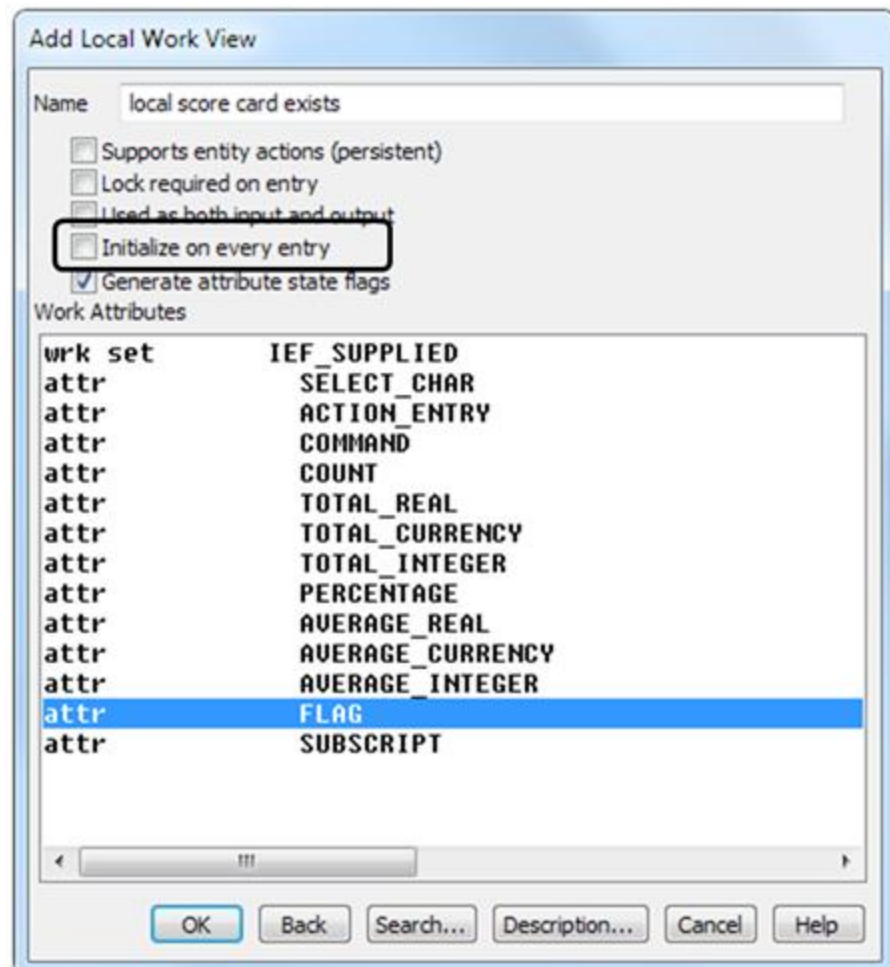
26. So to recap, the Open event gets triggered when this dialog opens. It will call the server procedure step to retrieve a score card image if it exists for the provided scoring record. If it is successful, we call the external action block to write the retrieved BLOB data to a file, and then pass that same file information to the OCX Control to display the image.

Add logic to the “ok” click event

Follow these steps:

1. The final event (in this diagram) is the OK push button click event. This event is a bit more complicated in that when the OK push button is clicked, they may be trying to add an image when none existed, update an image that did exist, or remove an image that they no longer want to keep. All from one button press. To help us figure this out, we need to know whether the Open event was successful in retrieving a scorecard image in the first place. If it was, we are either updating or deleting an existing image. If it was not, we are creating an image. So back in the Open event, we need to set a flag indicating whether or not it was successful. But first we need to define the flag.

Add a **LOCALS** work set view of the **IEF_SUPPLIED** flag. Name the view “**local score card exists**” and clear the selection of the **Initialize on every entry** checkbox as follows:



2. It is important to clear the selection of the Initialize on every entry checkbox because normally local views are initialized (set to spaces in this case) on every execution of the procedure step. But we do not want that to happen. When we set this flag to some value, we want it to remain set until we close this dialog. Click **OK**.
3. So back in the Open event, for the first statement inside the **IF EXITSTATE IS EQUAL TO processing_ok** statement, we want to set this flag to Y. Select the IF statement and add a SET statement, setting the attribute view **local_score_card_exists ief_supplied flag** to a character string of Y.
4. If it was not successful, we want to set it to N. So still in the Open event, select the last statement in the IF block of statements, the INVOKE statement, and add an **ELSE** statement after it.
5. With the ELSE statement highlighted, add another **SET** statement, setting the **attribute view local_score_card_exists ief_supplied flag** to a character string of N. Now the completed Open event will look as follows:

```

EVENT ACTION score_card_open
COMMAND IS DISPLAY
USE maintain_score_card (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
                Entity View import scoring_record TO Entity View import scoring_record
                <none> TO Entity View import score_card
  WHICH EXPORTS: <none> FROM Entity View export golfer
                <none> FROM Entity View export scoring_record
                Entity View local score_card FROM Entity View export score_card
IF EXITSTATE IS EQUAL TO processing_ok
  SET local score card exists ief supplied flag TO "Y"
  SET local eab_workset path TO "c:\temp\egolfblob.bin"
  USE eab_write_blob_to_file
    WHICH IMPORTS: Entity View local score_card TO Entity View import score_card
                  Work View local eab_workset TO Work View import eab_workset
  INVOKE EGOLFER_SCORE_CARD.csXImageOCXControl.LoadFromFile(local eab_workset path)
ELSE
  SET local score card exists ief supplied flag TO "N"

```

6. So now we are back to the OK event. The first thing we want to know is if the last thing they did before clicking OK was to select an image, or clear an image. We can determine that by calling a method on this OCX Control that will return to us the path to the last file loaded. If in fact the last thing done was to clear an image, then the path will be spaces. So we want to set our local eab_workset path to the OCX Control LastFileName method.

Select the **EVENT ACTION score_card_pb_ok_click**. Add a **SET** statement selecting attribute view, **local eab_workset path**, **TO** expression, Window Object, **EGOLFER_SCORE_CARD.csXImageOCXControl (OLEControl)**, in the OLE Object Browser the method **LastFileName**. Click the **OK** button, and then the Add Statement **Add** button. The OK click event will look as follows:

```

EVENT ACTION score_card_pb_ok_click
SET local eab_workset path TO EGOLFER_SCORE_CARD.csXImageOCXControl.LastFileName

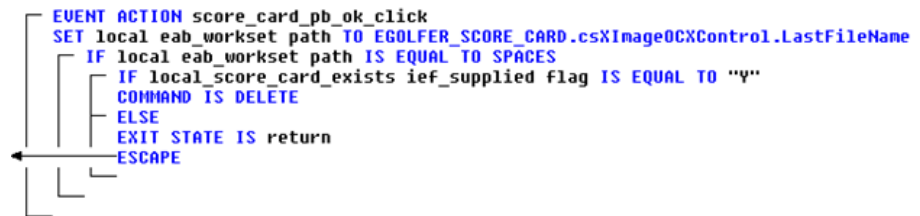
```

7. So now if the path is equal to spaces then the last thing they did was to clear the image. If they cleared the image and then clicked OK, the implication is that they want to delete whatever they had before. If they did not have anything before then there is nothing to delete and then the implication is that they simply want to flow back to the EGOLFER_HOME page.

So we will check to see if the local `eab_workset` path is equal to spaces. If it is, we will check to see if a `score_card` exists. If it does, we will set the `COMMAND` to `DELETE`, in anticipation of calling the `maintain_score_card` server. Else we assume they simply want to return, so we will set the exit state value which will initiate the return flow, and escape completely out of the client procedure step.

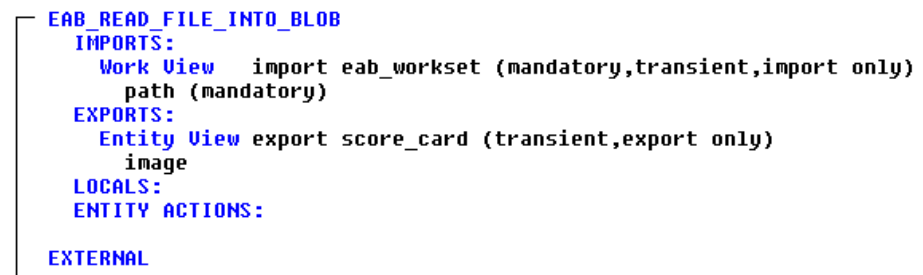
Add the statements required to support the logic described above. When finished, the OK click event will look as follows:

Note: Make sure the arrowhead on the `ESCAPE` clears the vertical line associated with the `EVENT ACTION`. If it does not, double-click the `ESCAPE` statement and try positioning the hand pointer further to the left.



8. The other option is that the path is not equal to spaces, which means the last thing done prior to clicking the OK button was to view something. So the implication is that they either want to create a new `score_card` if none existed, or update the `score_card` if it did. In either case we need to read the file into a BLOB view so that we can update the database. This will require another External Action Block. This EAB will import the `eab_workset` path, and export the `score_card` image.

Create an external action block and name it `eab_read_file_into_blob`. Add an IMPORTS work view of the `eab_workset` path and name it `Import`, and add an EXPORTS entity view of the `score_card` image and name it `Export`. Refer to the prior instructions for adding an EAB if you need assistance. The completed EAB will look as follows:



9. Close the new External Action Block and if necessary re-open the `EGOLFER_SCORE_CARD` procedure step action diagram. Back in the OK click event, we called the OCX Control method to return the last file opened. If the local `eab_workset` path is not equal to spaces, we want to use the new EAB to read that file into the local `score_card` image. Then if a `score_card` already exists in our database, we can set the `COMMAND` to `UPDATE` in anticipation of calling the `maintain_score_card` server procedure step. If a `score_card` does not exist, we can set the `COMMAND` to `CREATE`.

In the Open event, select the line directly below the `ESCAPE` statement as follows:

```

EVENT ACTION score_card_pb_ok_click
SET local eab_workset path TO EGOLFER_SCORE_CARD.csXImageOCXControl.LastFileName
IF local eab_workset path IS EQUAL TO SPACES
  IF local_score_card_exists ief_supplied flag IS EQUAL TO ""
    COMMAND IS DELETE
  ELSE
    EXIT STATE IS return
  ESCAPE

```

Add the ELSE statement followed by the **USE eab_read_file_into_blob**, matching both the import and export views to the equivalent local views, and then add the IF statement for setting the appropriate COMMAND. When finished, the OK click event will look as follows:

```

EVENT ACTION score_card_pb_ok_click
SET local eab_workset path TO EGOLFER_SCORE_CARD.csXImageOCXControl.LastFileName
IF local eab_workset path IS EQUAL TO SPACES
  IF local_score_card_exists ief_supplied flag IS EQUAL TO ""
    COMMAND IS DELETE
  ELSE
    EXIT STATE IS return
  ESCAPE
ELSE
  USE eab_read_file_into_blob
  WHICH IMPORTS: Work View local eab_workset TO Work View import eab_workset
  WHICH EXPORTS: Entity View local score_card FROM Entity View export score_card
  IF local_score_card_exists ief_supplied flag IS EQUAL TO ""
    COMMAND IS UPDATE
  ELSE
    COMMAND IS CREATE

```

10. Now that we have the necessary information, the only thing left is to call the `maintain_score_card` server and pass that information to it. And if the server completes its processing successfully, we can set the exit state value to return to the `EGOLFER_HOME` page.

Select the line that represents the end of the first IF statement as follows:

```

EVENT ACTION score_card_pb_ok_click
SET local eab_workset path TO EGOLFER_SCORE_CARD.csXImageOCXControl.LastFileName
IF local eab_workset path IS EQUAL TO SPACES
  IF local_score_card_exists ief_supplied flag IS EQUAL TO ""
    COMMAND IS DELETE
  ELSE
    EXIT STATE IS return
  ESCAPE
ELSE
  USE eab_read_file_into_blob
  WHICH IMPORTS: Work View local eab_workset TO Work View import eab_workset
  WHICH EXPORTS: Entity View local score_card FROM Entity View export score_card
  IF local_score_card_exists ief_supplied flag IS EQUAL TO ""
    COMMAND IS UPDATE
  ELSE
    COMMAND IS CREATE

```

11. Add a **procedure step use** statement calling the `maintain_score_card` server procedure step.

Match the import golfer and import scoring_record to the import views of the `EGOLFER_SCORE_CARD` procedure step and the import score_card to the local score_card.

You can ignore the export views and just select the Close button on the Match Exports dialog.

12. If the processing was successful, set the EXIT STATE to the value that will be required to return to the EGOLFER_HOME page. The completed OK click event will look as follows:

```

EVENT ACTION score_card_pb_ok_click
SET local eab_workset path TO EGOLFER_SCORE_CARD.csXImage0CXControl.LastFileName
IF local eab_workset path IS EQUAL TO SPACES
  IF local_score_card_exists ief_supplied flag IS EQUAL TO "Y"
    COMMAND IS DELETE
  ELSE
    EXIT STATE IS return
  ESCAPE
ELSE
  USE eab_read_file_into_blob
  WHICH IMPORTS: Work View local eab_workset TO Work View import eab_workset
  WHICH EXPORTS: Entity View local score_card FROM Entity View export score_card
  IF local_score_card_exists ief_supplied flag IS EQUAL TO "Y"
    COMMAND IS UPDATE
  ELSE
    COMMAND IS CREATE
  USE maintain_score_card (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
  Entity View import scoring_record TO Entity View import scoring_record
  Entity View local score_card TO Entity View import score_card
  WHICH EXPORTS: <none> FROM Entity View export golfer
  <none> FROM Entity View export scoring_record
  <none> FROM Entity View export score_card
  IF EXITSTATE IS EQUAL TO processing_ok
    EXIT STATE IS return

```

13. If you have not done so lately, save the model. The completed EGOLFER_SCORE_CARD procedure step action diagram will look as follows:

```

EGOLFER_SCORE_CARD
IMPORTS:
  Entity View import golfer (optional,transient,import only)
  userid (optional)
  Entity View import scoring_record (optional,transient,import only)
  date (optional)
  time (optional)
EXPORTS:
  Entity View export golfer (transient,export only)
  userid
  Entity View export scoring_record (transient,export only)
  date
  time
LOCALS:
  Work View local_score_card_exists ief_supplied
  flag
  Work View local eab_workset
  path
  Entity View local score_card
  image
ENTITY ACTIONS:

```

```

EVENT ACTION score_card_open
COMMAND IS DISPLAY
USE maintain_score_card (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
                Entity View import scoring_record TO Entity View import scoring_record
                <none> TO Entity View import score_card
  WHICH EXPORTS: <none> FROM Entity View export golfer
                <none> FROM Entity View export scoring_record
                Entity View local score_card FROM Entity View export score_card
  IF EXITSTATE IS EQUAL TO processing_ok
  SET local_score_card_exists ief_supplied flag TO "Y"
  SET local_eab_workset_path TO "c:\temp\egolfblob.bin"
  USE eab_write_blob_to_file
    WHICH IMPORTS: Entity View local score_card TO Entity View import score_card
                  Work View local eab_workset TO Work View import eab_workset
  INVOKE EGOLFER_SCORE_CARD.csXImageOCXControl.LoadFromFile(local_eab_workset_path)
  ELSE
  SET local_score_card_exists ief_supplied flag TO "N"

EVENT ACTION score_card_pb_select_image_click
INVOKE EGOLFER_SCORE_CARD.csXImageOCXControl.LoadDialog()

EVENT ACTION score_card_pb_remove_image_click
INVOKE EGOLFER_SCORE_CARD.csXImageOCXControl.Clear()

EVENT ACTION score_card_pb_ok_click
SET local_eab_workset_path TO EGOLFER_SCORE_CARD.csXImageOCXControl.LastFileName
  IF local_eab_workset_path IS EQUAL TO SPACES
  IF local_score_card_exists ief_supplied flag IS EQUAL TO "Y"
  COMMAND IS DELETE
  ELSE
  EXIT STATE IS return
  ESCAPE
  ELSE
  USE eab_read_file_into_blob
    WHICH IMPORTS: Work View local eab_workset TO Work View import eab_workset
                  WHICH EXPORTS: Entity View local score_card FROM Entity View export score_card
  IF local_score_card_exists ief_supplied flag IS EQUAL TO "Y"
  COMMAND IS UPDATE
  ELSE
  COMMAND IS CREATE

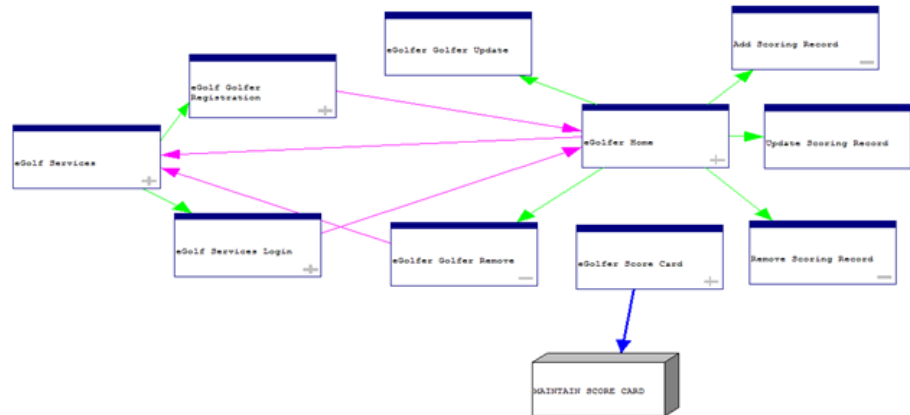
USE maintain_score_card (procedure step)
  WHICH IMPORTS: Entity View import golfer TO Entity View import golfer
                Entity View import scoring_record TO Entity View import scoring_record
                Entity View local score_card TO Entity View import score_card
  WHICH EXPORTS: <none> FROM Entity View export golfer
                <none> FROM Entity View export scoring_record
                <none> FROM Entity View export score_card
  IF EXITSTATE IS EQUAL TO processing_ok
  EXIT STATE IS return

```

Modify the eGolfer Home Page to link to Score Card

Follow these steps:

- Now we need to define the Link between the EGOLFER_HOME procedure step and the EGOLFER_SCORE_CARD procedure step. Return to the Window Navigation Diagram. In the Window Navigation Diagram, ensure that you are in the EGOLF_WEB_SERVER navigation diagram. In the EGOLF_WEB_SERVER navigation diagram you will now see the new client and server procedure steps as follows:

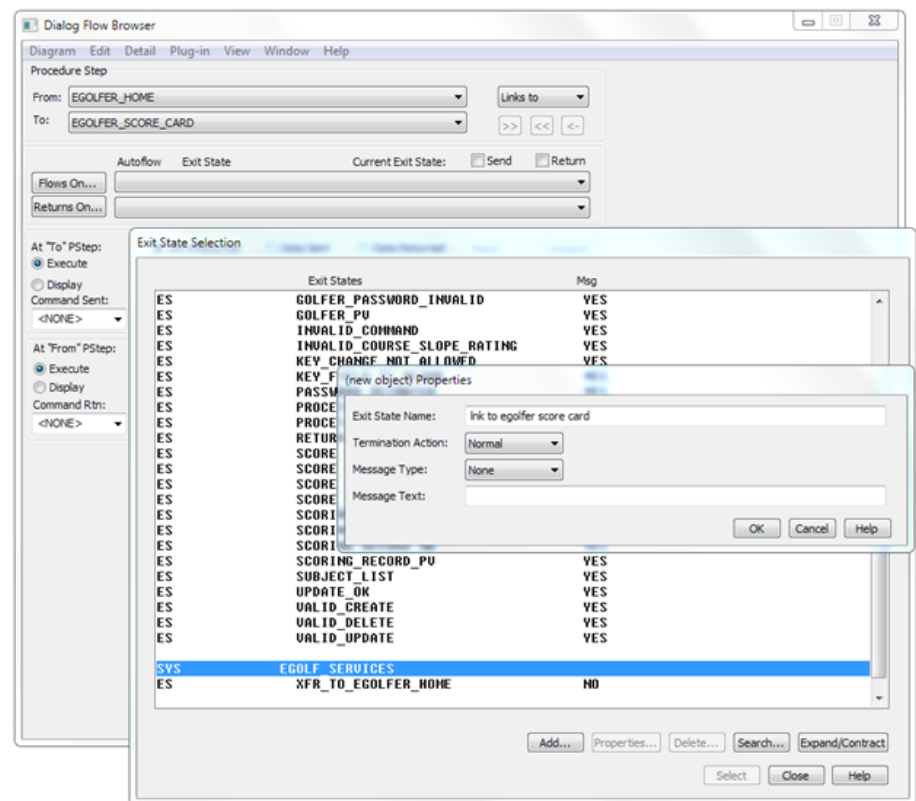


- In the window navigation diagram, select the **eGolfer_Home** window first. Then using the **Ctrl** key, select the **eGolfer_Score_Card** window. Then from the Menu Bar click **Edit, Join**. The Dialog Flow Browser opens.

If you clicked the two procedure steps in the correct order, the From Procedure Step will be EGOLFER_HOME and the To Procedure Step the EGOLFER_SCORE_CARD procedure step. If they are not, Cancel out and try again.

- To the right of the From Procedure Step, you can see that the flow defaulted to Links to, which is what we want. Make no changes here.

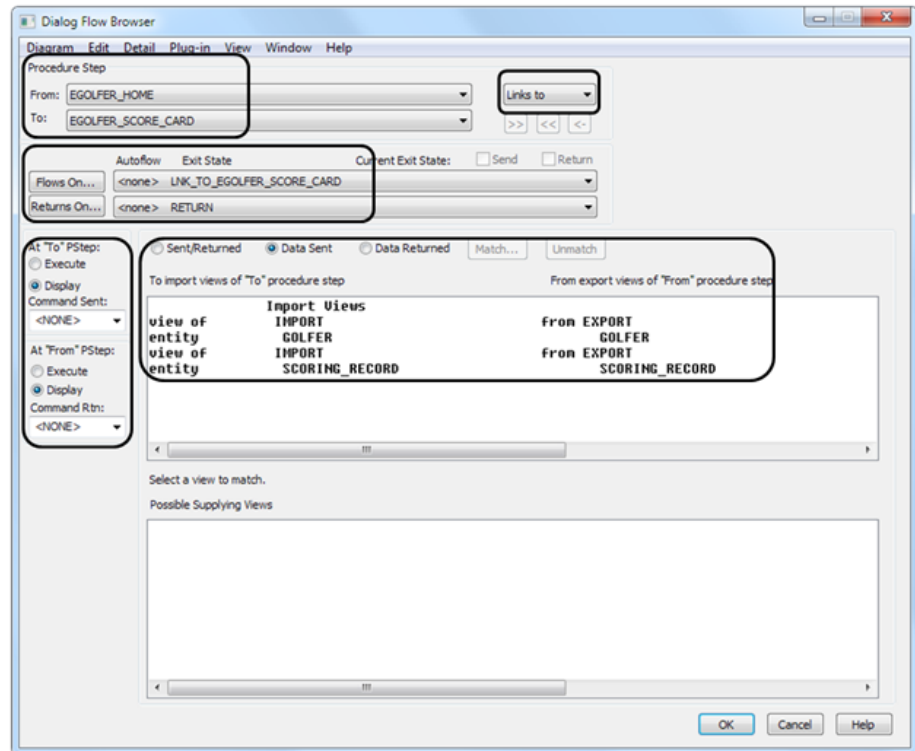
The Flows On and Returns On exit state values have not been set. Click the **Flows On** button. The Exit State Selection dialog opens. Scroll down and select the **EGOLF_SERVICES** business system. Click the **Expand/Contract** button. With **EGOLF_SERVICES** still selected, click the **Add** button. In the (new object) Properties dialog, enter the exit state name "**lnk to egolfer score card**" as follows:



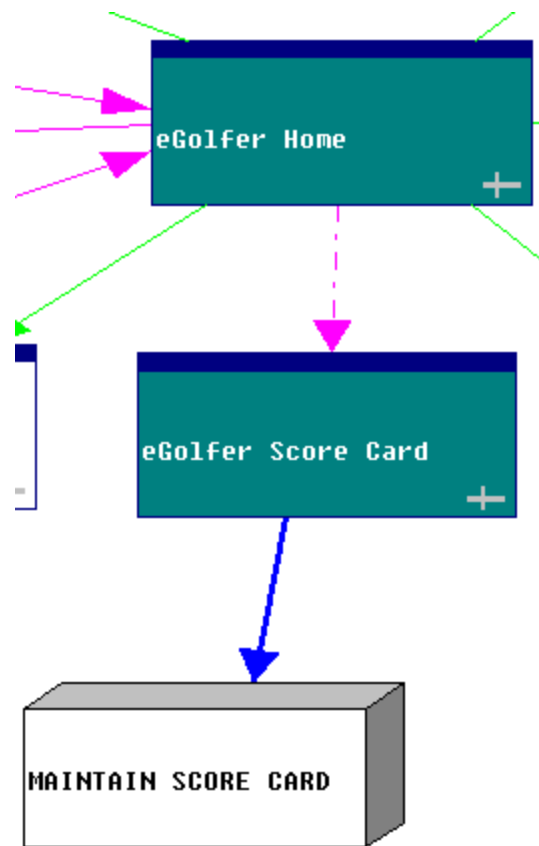
- Click **OK**. The new exit state value will be added to the EGOLF_SERVICES business system.
- Select the new **LNK_TO_EGOLFER_SCORE_CARD** exit state and click the **Select** button. The Exit State Selection dialog will close and the Flows On exit state value in the Dialog Flow Browser will show LNK_TO_EGOLFER_SCORE_CARD.
- Click the **Returns On** button. In the Exit State Selection list expand the **<Global Exit States>** and select the **RETURN** exit state. Click the **Select** button. The Exit State Selection list closes and the Returns On exit state value in the Dialog Flow Browser will show RETURN.
- Below the Flows On and Returns On buttons are the **At To** and **At From PStep** options. Select the **Display** radio button for both.
- Finally, we have to pass data from one client procedure step to the other along the flow. To the right of the At To PStep radio buttons are the data sent and returned radio buttons. Select the **Data Sent** radio button. When you do that, the top panel shows the import views of the EGOLFER_SCORE_CARD procedure step. We want to match those views to export views from the EGOLFER_HOME procedure step. Select the **IMPORT GOLFER** view. When you do that the bottom panel shows the views of Golfer in the export view of EGOLFER_HOME. Select **EXPORT GOLFER** and click the **Match** button above the panels. Select the **IMPORT SCORING_RECORD** in the top panel. Select the **EXPORT SCORING_RECORD** in the bottom panel and click the **Match** button.

We do not have to return any data from the EGOLFER_SCORE_CARD procedure step.

9. The completed Dialog Flow Browser is as follows:



10. Click **OK** to close the Dialog Flow Browser.
11. Back in the window navigation diagram, the new flow is depicted with a dashed line as follows:



The dashed line indicates that we have not set the exit state value in the eGolfer Home procedure step to initiate the flow. Double-click **eGolfer Home** to open its window design. The window opens as follows:

eGolfer Home

Place Your Ad Here

Personal Profile

Welcome XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Your Handicap Index is ZZ9.9

Scoring Record

Date	Time	Score	Rating	Slope	Note
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX

12. Right-click the Scoring Record **remove** push button and drag it down leaving room between it and the update push button for another button.
13. From the Menu Bar click **Add, Push Button**. In the Push Button Properties dialog enter **"score card"** for the text. Click the **Events** push button. In the Events Processing dialog click the **Add** button. It will add a new event to the list of events. Click the **Close** button on the Event Processing dialog and the **OK** button on the Push Button Properties dialog. Position the new push button between the update and remove push buttons.
14. Click the Scoring Record **add** push button. Using the **Ctrl** key select the **update**, scorecard, and **remove** push buttons in that order. All of the Scoring Record push buttons will be highlighted. From the Menu Bar click **Edit, Position**. In the Field Positioning dialog select the **Equal Height** and **Equal Width** checkboxes. Select the **Align Vertically Left** checkbox. Select the **Separate Vertically** checkbox and enter the Distance **5**. Click **OK**.

We need to update the video display properties of the score card push button. Select the **score card** push button. From the Menu Bar click **Detail, Video Properties**. In the Push Button Video Properties dialog clear the selection of **Automatic** for the Foreground Color, Background Color, and the Font selection. For the Font, click the **Select** button. In the Font dialog, select **Bold** in the Font Style list and click **OK**. Then click **OK** on the Push Button Video Properties dialog. The eGolfer Home page will look as follows:

Personal Profile

update

remove

Welcome XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Your Handicap Index is ZZ9.9

Scoring Record

add

update

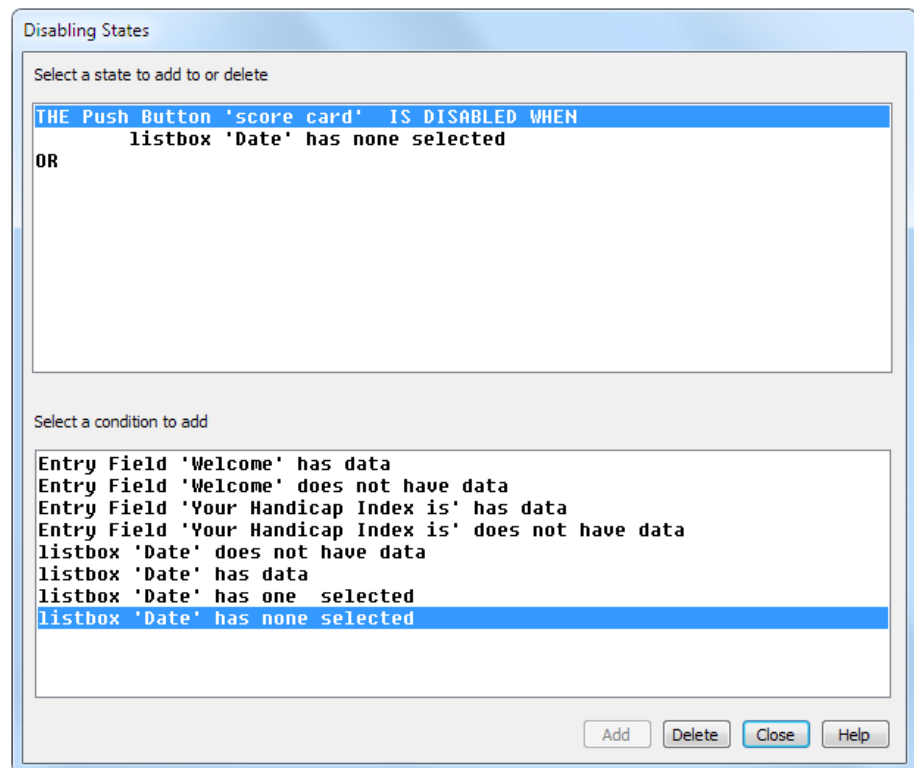
score card

remove

Date	Time	Score	Rating	Slope	Note
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX
MM/DD/YY	HH:MM	ZZZ	ZZ.Z	ZZZ	XXXXXX

logout

15. The score card push button must be disabled if no scoring records are selected. Select the **score card** push button in the window design. From the Menu Bar click **Detail, Disabled By**. In the Disabling States dialog select **THE Push Button 'score card' IS DISABLED WHEN** line in the top panel. Select the **listbox 'Date' has none selected** in the bottom panel. Then click the **Add** push button. The Disabling States dialog will look as follows:



16. Click **Close**.
17. Open the eGolfer Home procedure step action diagram. Scroll to the very bottom where the new event has been added. The only thing that is necessary to do here is get the row highlighted in the import group of scoring records and move it to the export scoring record. Then set the Exit State value to LNK_TO_SCORE_CARD. The statements to get the row highlighted can be copied from other events. Then add the EXIT STATE IS statement. The completed event is as follows:

```

EVENT ACTION egolfer_hom_pb_score_card_click
GET ROW HIGHLIGHTED IN import_group_of_scoring_records STARTING AT 1 GIVING SUBSCRIPT OF
import_group_of_scoring_records
MOVE import_group_scoring_record TO export_scoring_record
EXIT STATE IS lnk_to_egolfer_score_card

```

18. Close the EGOLFER_HOME action diagram. In the window navigation diagram the flow between the two procedure steps will now be a solid line.
19. **Save** the model.

Construction and Test

Database Generation

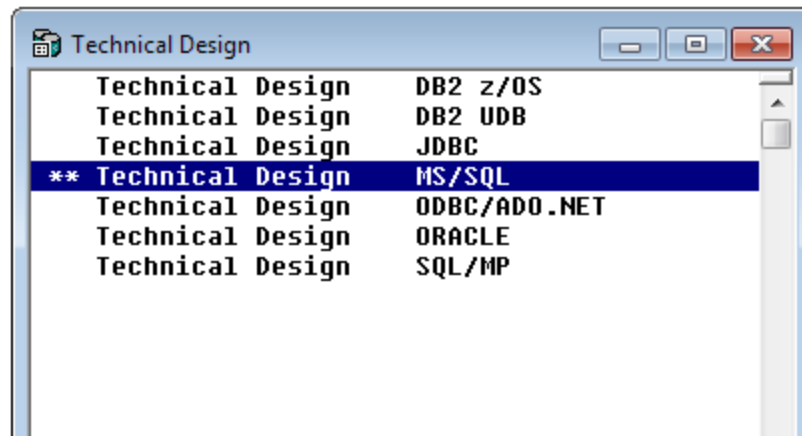
Data Model Transformation

Before we can generate the DDL for the new database table, we have to convert the logical representation of the Score Card entity type to a physical representation of a database table.

Normally this might be accomplished by doing a complete transformation of the logical data model to a physical data model as represented by the Data Structure List and the Data Store List diagrams. Collectively these two diagrams are often referred to as the Technical Design. But if your DBA had made many implementation specific changes to the Technical Design, like reordering columns of a table or changing the order of indexes, a complete transformation would erase those changes as it would delete the existing Technical Design first and then re-apply the basic rules of transformation to the data model objects to create the new Technical Design. So rather than perform a complete transformation, we will perform an incremental transformation instead.

Follow these steps:

1. Under the Technical Design Defaults, ensure that you have the appropriate Technical Design selected. Your diagram will look similar to the following image:

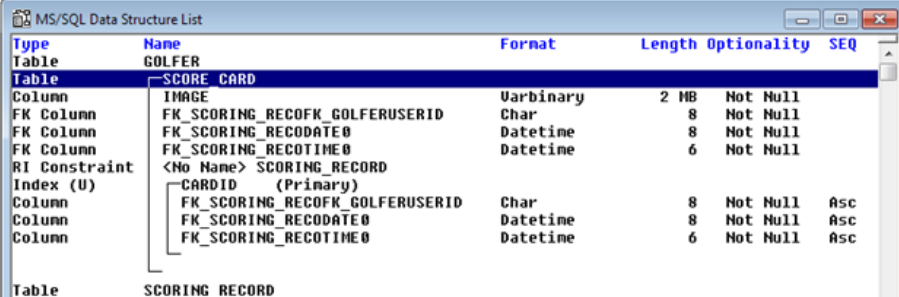


2. Open the **Data Structure List**. From the Menu Bar click **Edit, Implement Entity Type**. In the Unimplemented Entity Types and Subtypes dialog select the entity type **SCORE_CARD**. Upon selecting it a Consistency Check will be run.

On the Consistency Check dialog click the **Continue** push button. The dialog will close.

Note: If the Continue push button is disabled, this implies that the Consistency Check has failed. Click the Display push button to see the report.

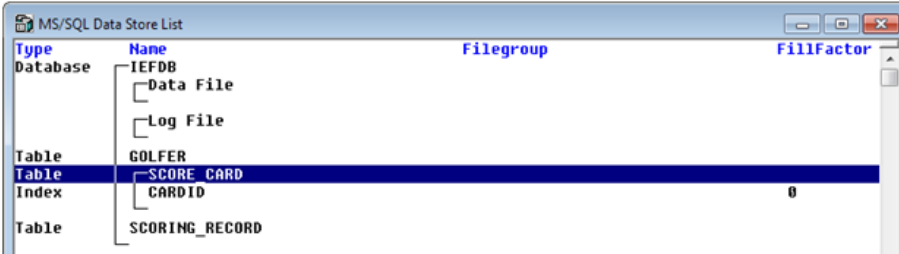
3. Back on the Unimplemented Entity Type dialog click **OK**. A (new object) (MS/SQL) Table Properties dialog will appear.
4. Click **OK** on the Table Properties dialog. The dialog will close and you will now see that the new table has been added to the Data Structure List.
5. Select the **SCORE_CARD** table in the Data Structure List and expand it. The Data Structure List will look as follows:



The screenshot shows the 'MS/SQL Data Structure List' window. The 'SCORE_CARD' table is expanded, showing its columns and constraints. The table 'GOLFER' is also visible above it.

Type	Name	Format	Length	Optionality	SEQ
Table	GOLFER				
Table	SCORE_CARD				
Column	IMAGE	Varbinary	2 MB	Not Null	
FK Column	FK_SCORING_REC0FK_GOLFERUSERID	Char	8	Not Null	
FK Column	FK_SCORING_REC0DATE0	Datetime	8	Not Null	
FK Column	FK_SCORING_REC0TIME0	Datetime	6	Not Null	
RI Constraint	<No Name> SCORING_RECORD				
Index (U)	CARDID (Primary)				
Column	FK_SCORING_REC0FK_GOLFERUSERID	Char	8	Not Null	Asc
Column	FK_SCORING_REC0DATE0	Datetime	8	Not Null	Asc
Column	FK_SCORING_REC0TIME0	Datetime	6	Not Null	Asc
Table	SCORING_RECORD				

6. Open the **Data Store List**. Select the **SCORE_CARD** table and expand it. The Data Store List will look as follows:



The screenshot shows the 'MS/SQL Data Store List' window. The 'SCORE_CARD' table is expanded, showing its index 'CARDID'. The table 'GOLFER' is also visible above it.

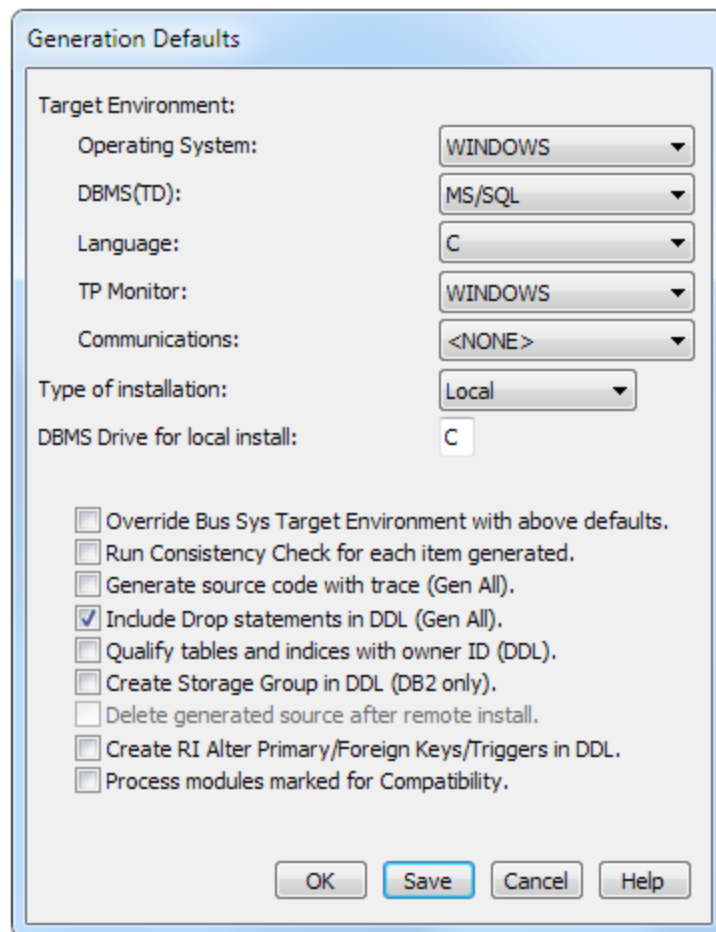
Type	Name	Filegroup	FillFactor
Database	IEFDB		
	Data File		
	Log File		
Table	GOLFER		
Table	SCORE_CARD		
Index	CARDID		0
Table	SCORING_RECORD		

DDL Generation

We are now ready to generate the DDL. Because we are just introducing a new table, and that table contains the foreign keys, we need only to generate the DDL for that table.

Follow these steps:

1. Open the DDL Generation dialog and then from the Menu Bar click **Options, Generation Defaults**. Verify the generation default settings look as follows (assuming you are targeting MS/SQL):



The screenshot shows the 'Generation Defaults' dialog box. It has a title bar 'Generation Defaults' and a light blue border. Inside, there are several sections. The 'Target Environment:' section contains five dropdown menus: 'Operating System' (set to 'WINDOWS'), 'DBMS(TD):' (set to 'MS/SQL'), 'Language:' (set to 'C'), 'TP Monitor:' (set to 'WINDOWS'), and 'Communications:' (set to '<NONE>'). Below this is 'Type of installation:' (set to 'Local') and 'DBMS Drive for local install:' (set to 'C'). A list of checkboxes follows: 'Override Bus Sys Target Environment with above defaults.' (unchecked), 'Run Consistency Check for each item generated.' (unchecked), 'Generate source code with trace (Gen All).' (unchecked), 'Include Drop statements in DDL (Gen All).' (checked), 'Qualify tables and indices with owner ID (DDL).' (unchecked), 'Create Storage Group in DDL (DB2 only).' (unchecked), 'Delete generated source after remote install.' (unchecked), 'Create RI Alter Primary/Foreign Keys/Triggers in DDL.' (unchecked), and 'Process modules marked for Compatibility.' (unchecked). At the bottom are four buttons: 'OK', 'Save' (highlighted with a blue border), 'Cancel', and 'Help'.

2. In the DDL Generation dialog, select the **SCORE_CARD** table and **CARDID** index for generation and installation as follows:

DDL Generation				
Type	DDL	Inst	Drop	Name
database	-	-	-	IEFDB
table	Y	Y	-	SCORE_CARD
index	Y	Y	-	CARDID
table	-	-	-	GOLFER
index	-	-	-	GOLFERID
table	-	-	-	SCORING_RECORD
index	-	-	-	SCOREID

- From the Menu Bar, click **Generate, DDL, Selected**. The DDL Generation dialog will change to report that the generation completed and then will automatically launch the Build Tool.
- The Build Tool will open and you may be prompted to logon to the database. If everything completes successfully, the Build Tool displays the Build-OK status as follows:

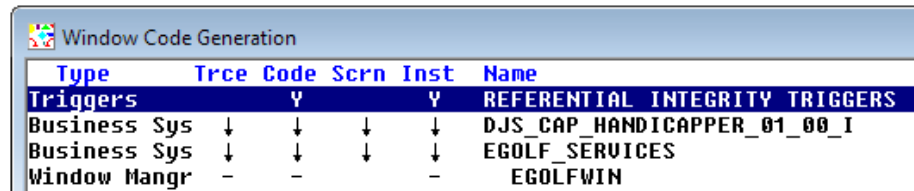
Module	Exten...	Date Generated	Type	Status	OS
EGOLF SERVICES					
IEFDB	icm	2013/02/28 17:11:52	DDL	Build-OK	WINDOWS

RI Trigger Generation

We can now generate the Referential Integrity Trigger library. The RI Triggers need to be re-generated for most database changes.

Follow these steps:

1. Open the Window Code Generation dialog. Select the **Referential Integrity Triggers** for code generation and installation as follows:



Type	Trce	Code	Scrn	Inst	Name
Triggers	Y	Y	Y	Y	REFERENTIAL INTEGRITY TRIGGERS
Business Sys	↓	↓	↓	↓	DJS_CAP_HANDICAPPER_01_00_I
Business Sys	↓	↓	↓	↓	EGOLF_SERVICES
Window Mangr	-	-	-	-	EGOLFWIN

2. From the Menu Bar, click **Generate, Code, Selected**. The Window Code Generation dialog will change to report that generation completed successfully and automatically launch the Build Tool if it is no longer open.
3. If everything completes successfully, the Build Tool displays the Build-OK status as follows:

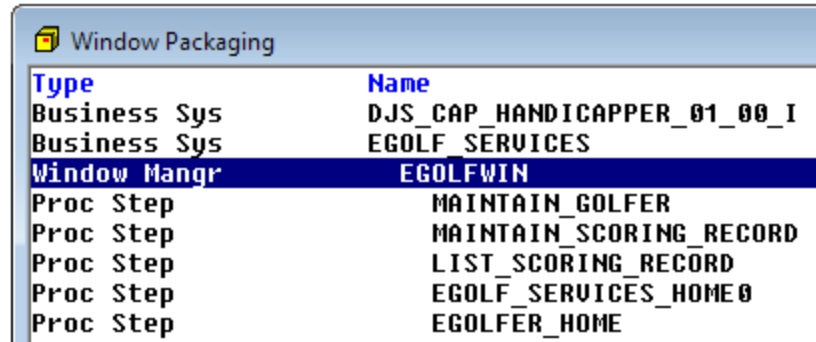
Module	Exten...	Date Generated	Type	Status	OS
EGOLF SERVICES					
IEFDB	icm	2013/02/28 17:11:52	DDL	Build-OK	WINDOWS
CASCADE	icm	2013/02/28 17:38:12	CASCADE	Build-OK	WINDOWS

Load Module Packaging

Before we can generate our source code and executables, we need to define how we want it all packaged.

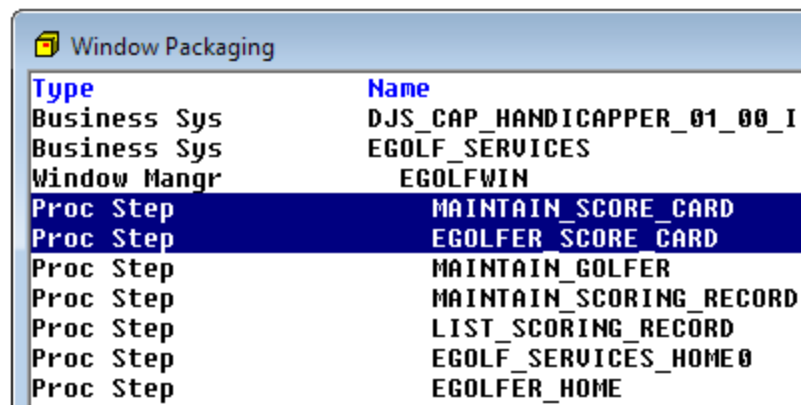
Follow these steps:

1. Open Window Packaging. In the Window Packaging dialog, select (highlight) the window manager **EGOLFWIN** and expand it one level. It will look as follows:



Type	Name
Business Sys	DJS_CAP_HANDICAPPER_01_00_I
Business Sys	EGOLF_SERVICES
Window Mangr	EGOLFWIN
Proc Step	MAINTAIN_GOLFER
Proc Step	MAINTAIN_SCORING_RECORD
Proc Step	LIST_SCORING_RECORD
Proc Step	EGOLF_SERVICES_HOME0
Proc Step	EGOLFER_HOME

2. From the Menu Bar, click **Edit, Add Procedure Step**. The Unpackaged Procedure Steps dialog will open. Select both **procedure steps** and click **OK**. The two procedure steps will be added to the EGOLFWIN load module as follows:

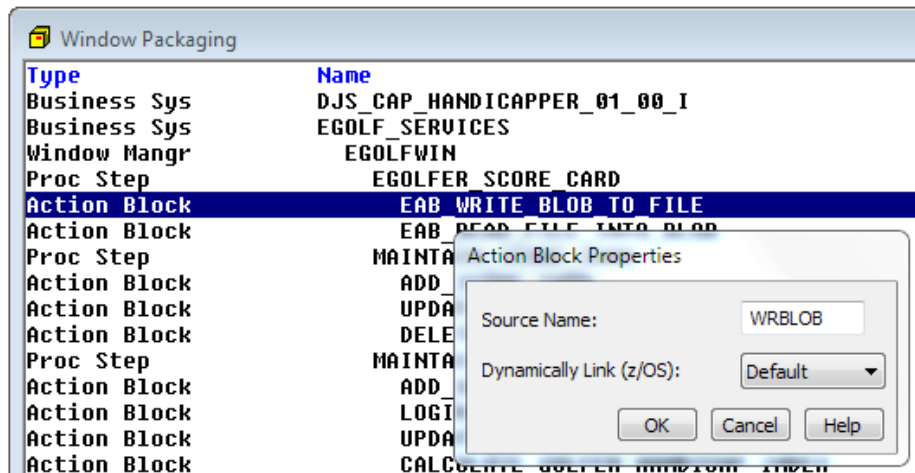


Type	Name
Business Sys	DJS_CAP_HANDICAPPER_01_00_I
Business Sys	EGOLF_SERVICES
Window Mangr	EGOLFWIN
Proc Step	MAINTAIN_SCORE_CARD
Proc Step	EGOLFER_SCORE_CARD
Proc Step	MAINTAIN_GOLFER
Proc Step	MAINTAIN_SCORING_RECORD
Proc Step	LIST_SCORING_RECORD
Proc Step	EGOLF_SERVICES_HOME0
Proc Step	EGOLFER_HOME

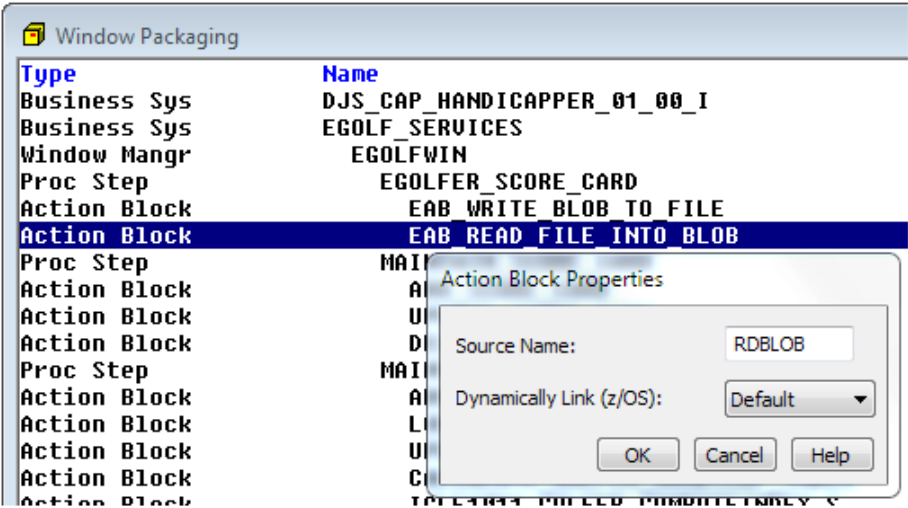
3. In the Window Packaging dialog, select the **EGOLFWIN** load module once again. From the Menu Bar click **Edit, Complete**. The window manager will be expanded automatically as follows:

Window Packaging	
Type	Name
Business Sys	DJS_CAP_HANDICAPPER_01_00_I
Business Sys	EGOLF_SERVICES
Window Mangr	EGOLFWIN
Proc Step	EGOLFER_SCORE_CARD
Action Block	EAB_WRITE_BLOB_TO_FILE
Action Block	EAB_READ_FILE_INTO_BLOB
Proc Step	MAINTAIN_SCORE_CARD
Action Block	ADD_SCORE_CARD
Action Block	UPDATE_SCORE_CARD
Action Block	DELETE_SCORE_CARD
Proc Step	MAINTAIN_GOLFER
Action Block	ADD_GOLFER
Action Block	LOGIN_GOLFER
Action Block	UPDATE_GOLFER
Action Block	CALCULATE_GOLFER_HANDICAP_INDEX
Action Block	IGLF1011_GOLFER_COMPUTEINDEX_S
Action Block	DELETE_GOLFER
Proc Step	MAINTAIN_SCORING_RECORD
Action Block	ADD_SCORING_RECORD
Action Block	UPDATE_SCORING_RECORD
Action Block	DELETE_SCORING_RECORD
Proc Step	LIST_SCORING_RECORD
Proc Step	EGOLF_SERVICES_HOME0
Proc Step	EGOLFER_HOME

4. The last things we need to do are to make sure the two External Action Blocks have the correct source code names. These names will be used by the Build Tool to link in the appropriate external. Double-click the action block **EAB_WRITE_BLOB_TO_FILE** and verify it has **WRBLOB** for the source name. Update it if necessary as follows:



5. Double-click the action block **EAB_READ_FILE_INTO_BLOB** and verify it has **RDBLOB** for the source name. Update it if necessary as follows:

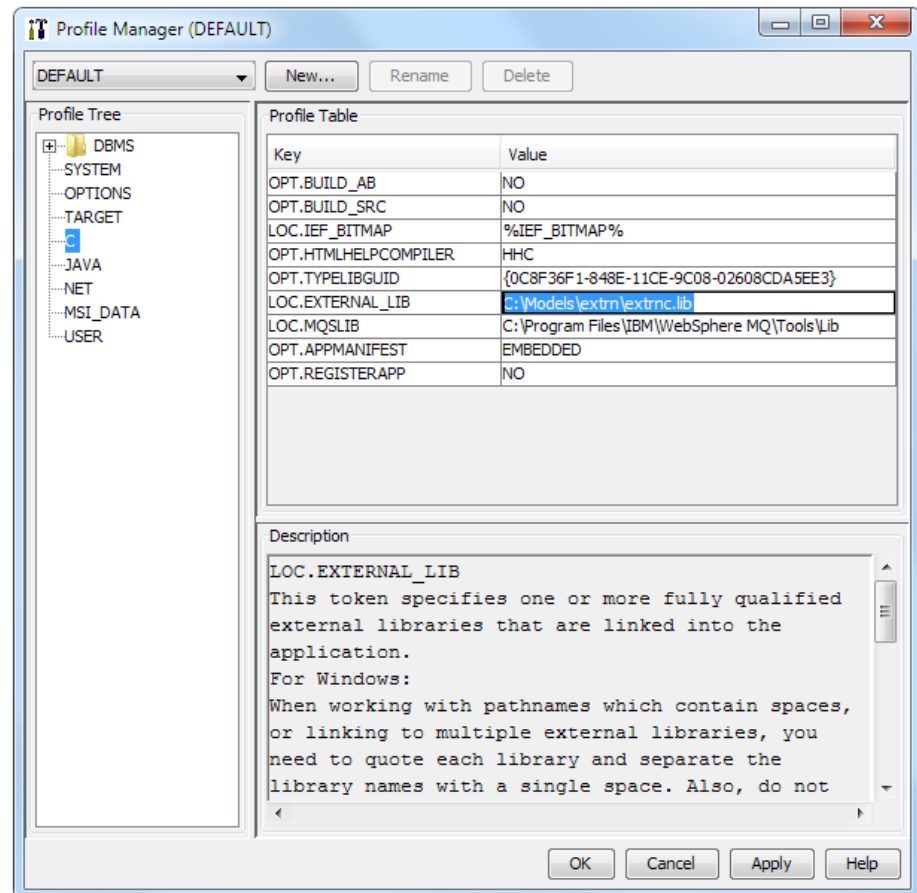


Configure the Build Tool to Support External Action Block Linking

Earlier, in the section titled Software Required to Complete this Appendix, you were instructed to copy the `extrn` directory to your models directory. The `extrn` directory contains the externally coded action blocks. The Build Tool Profile needs to be updated to know where to find the EAB library.

Follow these steps:

1. If it is not already opened, open the Build Tool.
2. From the Build Tool Menu Bar, click **Tools, Profile Manager**. The Profile Manager will be opened.
3. In the Profile Manager Profile Tree, select the **C** node within the tree.
4. In the Profile Table, update the value of the **LOC.EXTERNAL_LIB** key to point to the library containing the externally generated code. In our example that was **C:\Models\extrn\extrnc.lib** as follows:



5. Click **OK** to save the changes and close the dialog.

Note: The Tutorial eGolf model also makes use of a “black box” component. If you have not done so already, you will need to follow the instructions in Chapter Five of the Tutorial for Configuring the Build Tool to Support Component Linking.

Application Generation

Now we are ready to generate the remaining code.

Follow these steps:

1. In the Toolset, open the WindowCode Generation.
2. In the WindowCode Generation dialog, select the Window Manager **EGOLFWIN**. From the Menu Bar, click **View, Expand All**. The window manager will be fully expanded.
3. We created two new procedure steps, the client procedure step **EGOLFER_SCORE_CARD** and the server procedure step **MAINTAIN_SCORE_CARD**. And we modified one existing client procedure step, **EGOLFER_HOME** with a link to the score card client. So at a minimum, those three procedure steps and all of their associated action blocks need to be generated.

Additionally, we defined a flow between the two client procedure steps and changed the Windows user interface. So the Window Manager needs to be regenerated.

Select the **down arrow** under the **Code** column for the window manager. That will select all of the action diagrams for code generation. But we also want them to be compiled and linked, so select the **down arrow** under the **Install (Inst)** column.

If we wanted to be more efficient in our generation, and we had already generated and tested the application earlier, we could clear the selection of the portions of the load module that have not changed as follows, otherwise leave everything selected:

Window Code Generation					
Type	Trce	Code	Scrn	Inst	Name
Triggers		-		-	REFERENTIAL INTEGRITY TRIGGERS
Business Sys	↓	↓	↓	↓	DJS_CAP_HANDICAPPER_01_00_I
Business Sys	↓	↓	↓	↓	EGOLF SERVICES
Window Mangr	-	Y		Y	EGOLFWIN
Proc Step	-	Y			EGOLFER_SCORE_CARD
Action Block	-	Y			EAB_WRITE_BLOB_TO_FILE
Action Block	-	Y			EAB_READ_FILE_INTO_BLOB
Proc Step	-	Y			MAINTAIN_SCORE_CARD
Action Block	-	Y			ADD_SCORE_CARD
Action Block	-	Y			UPDATE_SCORE_CARD
Action Block	-	Y			DELETE_SCORE_CARD
Proc Step	-	-			MAINTAIN_GOLFER
Action Block	-	-			ADD_GOLFER
Action Block	-	-			LOGIN_GOLFER
Action Block	-	-			UPDATE_GOLFER
Action Block	-	-			CALCULATE_GOLFER_HANDICAP_INDEX
Action Block	-	-			IGLF1011_GOLFER_COMPUTEINDEX_S
Action Block	-	-			DELETE_GOLFER
Proc Step	-	-			MAINTAIN_SCORING_RECORD
Action Block	-	-			ADD_SCORING_RECORD
Action Block	-	-			UPDATE_SCORING_RECORD
Action Block	-	-			DELETE_SCORING_RECORD
Proc Step	-	-			LIST_SCORING_RECORD
Proc Step	-	-			EGOLF_SERVICES_HOME0
Proc Step	-	Y			EGOLFER_HOME

4. From the Menu Bar, click **Generate, Code, Selected**. The Window Code Generation dialog will change to report that generation completed successfully and automatically launch the Build Tool if it is no longer open.
5. If everything completes successfully, the Build Tool displays the Build-OK status as follows:

Module	Exten...	Date Generated	Type	Status	OS
EGOLF SERVICES					
IEFDB	icm	2013/02/28 17:11:52	DDL	Build-OK	WINDOWS
CASCADE	icm	2013/02/28 17:38:12	CASCADE	Build-OK	WINDOWS
EGOLFWIN	icm	2013/02/28 19:45:22	LM	Build-OK	WINDOWS

6. Now that we have generated some code, we can look at the workings of External Action Blocks in a little more detail.

Go to your models "C" directory and locate the RDBLOB.c file. Open the file with Notepad.

What you will find is the shell of a C program with parameters defined corresponding to the import and export views defined in our EAB action diagram. If you scroll to the bottom of the diagram you will see a section with the note;

```
/* User-written code should be inserted here */
```

So now someone who wanted to write their own code would take this stub as a starting point, copy it to another directory and add their code to it, being careful not to change the properties of the parameters already defined. Then they would compile it like they would any program they were handwriting and make the external code available to the Gen generated application through an external library.

For an example of this, locate the same C file in the extrn folder. Open it and scroll to the bottom to see where someone added the code to read the BLOB from a file location provided to it in its import parameter and put its contents into the export parameter. It was then compiled and made available through the extrnc.lib.

Application Test

We have provided a few sample golf score card image files in the models bitmap directory. The score card images begin with “card”.

When tracing BLOB attribute views, the actual value of the BLOB is not shown in the Diagram Trace Utility, only the actual length as follows:

▶ ◆ EXPORT View	
▶ ◆ View - EXPORT GOLFER	
◆ USERID	TIGER
▶ ◆ View - EXPORT SCORING_RECORD	
◆ DATE	20120101
◆ TIME	010000
▶ ◆ View - EXPORT SCORE_CARD	
◆ IMAGE	{BLOB;maxlength=2097152;length=37045}