Team Developer™

Introducing Team Developer

Product Version 7.0

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1
Tel: +1-519-888-7111 Toll Free Canada/USA: 1-800-499-6544 International: +800-4996-5440
Fax: +1-519-888-0677
Support: http://support.opentext.com
For more information, visit https://www.opentext.com

Table of Contents

Preface

The purpose of this document is to help you install Team Developer on your workstation and to get you started building applications and managing projects.

Audience

This document is intended for first-time Team Developer users, users looking to try the new features of Gupta Team Developer, or for those evaluating Team Developer for use within their organization. This document helps you become familiar with the Team Developer interface and programming tools.

Requirements

This manual assumes you have experience with:

- Intel-compatible Personal Computers
- Microsoft Windows
- A programming language such as C, Java, or Basic
- SQL (Structured Query Language) and relational databases
- COM (Common Object Model) programming
- DLLs (Dynamic Link Libraries)
- LANs (Local Area Networks)

Contents

This manual contains software installation procedures, explanations of Team Developer concepts, and tutorials explaining the basics of SQLWindows programming. After using the manual, you will be ready to use Team Developer to write your own applications.

- Chapter 1 – Using SQLWindows

  Shows you how to start building applications with Gupta SQLWindows. Includes a tutorial that shows you how to build a simple Windows application using SQLWindows.

- Chapter 2 – Desktop and Components

  Introduces the Gupta user interface and the various tools included with SQLWindows.

- Chapter 3 – Using ActiveX Objects

  Explains how to use an ActiveX object in a SQLWindows application.

- Chapter 4 – Developing N-Tier Applications Using TD and COM

Introduces COM programming in SQLWindows and guides you through the process of creating a simple COM-based application using the sample COM application provided with Gupta Team Developer.

- Chapter 5 – Using Team Developer and COM+

Describes how to modify a SQLWindows developed COM application so that it can be deployed using COM+ (previously MTS). It uses the sample COM+ application provided with Gupta Team Developer to illustrate this process. It also shows you how to develop an ASP-based COM client for a SQLWindows-based COM server.

- Chapter 6 – Managing Teams and Objects

Introduces Team Object Manager, which enables you to track large-scale projects involving teams of developers.

- Chapter 7 – More Information

Provides information on the Gupta Bookcase and Gupta Books Online.

Chapter 1 – Using SQLWindows

This chapter introduces you to developing applications with SQLWindows, a client/ server application development and deployment environment for Microsoft Windows. For information on the SQLWindows desktop, see chapter 2, "Desktop and Components." There you can learn about the outline tab and the tools you use in this chapter such as the Coding Assistant and the Attribute Inspector.

SQLWindows lets you code and layout applications in an integrated graphical environment:

SQLWindows includes the following features:

- Drag-and-drop user interface design

- Multiple source code and user interface views

- Context-sensitive coding assistant

- Online help for all functions

- Interactive debugging

Installing Team Developer

Follow these steps:

1. Start Microsoft Windows.

2. Put the Team Developer CD in the CD drive.

    The TD installer should start automatically. If it does not, select **Start** > **Run** and type **D:\TDSetup** (or the letter of your CD drive) in the Open field.

    Click **OK**. The Installation dialogs that follow provide instructions on all of the installation options available for Gupta Team Developer.

3. Proceed to the next section to start building an application.

Installing and Using Team Developer from Remote Clients

Server setup

First you will need to map the drive for use by all remote clients. To map the drive, select "My Computer" from the Windows menu (Start menu in pre-Vista Windows versions). In the Tools menu, select "Map Network Drive." Then select a drive letter (for example, X:) and create a location for installing Team Developer. The mapped path will be used by "clientruntimedir" and "autostartserverpath" entries in the sql.ini. Now perform a local install of Team Developer in the mapped drive folder on the server machine.

Once the installation has finished, launch Team Developer from the Windows menu. On first usage of Team Developer, you will be prompted to register and activate. After you have registered the product, Gupta will e-mail you a Product Activation Key to activate the product. Registration and activation only need to be performed once per installation.

In the sql.ini file in the Team Developer directory, replace "localhost" with the hostname or IP address of the server machine. This enables remote clients to properly connect to SQLBase, as well as use the tools available through the Team Developer "Tools" menu.

Register and activate the product before performing remote installs.

Client setup

From a remote machine, map a drive letter and location corresponding with installation on the server machine. Find and run **SetupClient.bat** in the Team Developer directory. This installs registry entries, program group entries, and Add/ Remove Programs entries.

- **register.htm** is a reminder for registration of the product. This is created in the temp folder of the remote client.

- **SetupTDClient.reg** and **SetupRBClient.reg** update registry entries for file associations. Double click on these registry files to run them. These are created in the temp folder of the remote client.

The PATH environment variable on the remote machine is not updated as part of the installation. You may need to modify the PATH to point to the program directory on the mapped drive. For example add "X:\;X:\Program Files\Gupta\Team Developer 6.2;" to the PATH environment variable. To access environment variables, right-click **My Computer** and select **Properties**. In the Advanced tab, click the **Environment Variables** button.

If the PATH is not modified, issues, such as not being able to access the online help, could arise.

Please register and activate the product.

You cannot run Team Developer both locally and via network on the same machine unless you appropriately set registry entries like the INIFilePath, as well as address the file associations.

Tutorial for SQLWindows

The tutorial that follows introduces you to SQLWindows, the Windows development tool included with the Gupta Team Developer application development environment.

It is helpful to examine the finished version of the tutorial application, called Account Info, before you begin to build it.

To open, compile, and use the completed Account Info application in SQLWindows, follow these steps:

1. Launch **SQLWindows**. The default Start menu location is Programs, Gupta, Team Developer 6.2, SQLWindows 6.2.

2. Select **File** > **Open**.

3. Navigate to the **..\Program Files\Gupta\Team Developer 6.2\Samples\Connectivity** directory.

   You must have samples installed. The samples installer is on the installation CD as TDSamples51.exe.

4. Double-click **AccountInfo.app**.

   The Account Info application opens. This application requests a valid Logon and then allows you to view account information for companies stored in the Island database.

5. Select **Debug** > **Go**.

   The application compiles, and you view the Database login dialog.
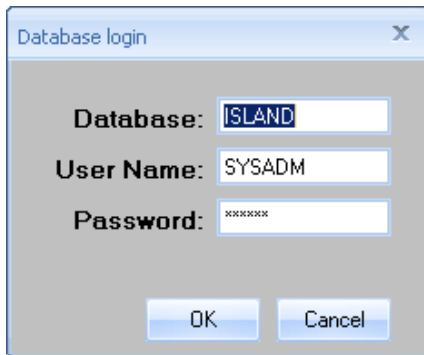
6. Click **OK**. SQLBase, SQL Database, is launched and the Account Information Form is displayed.
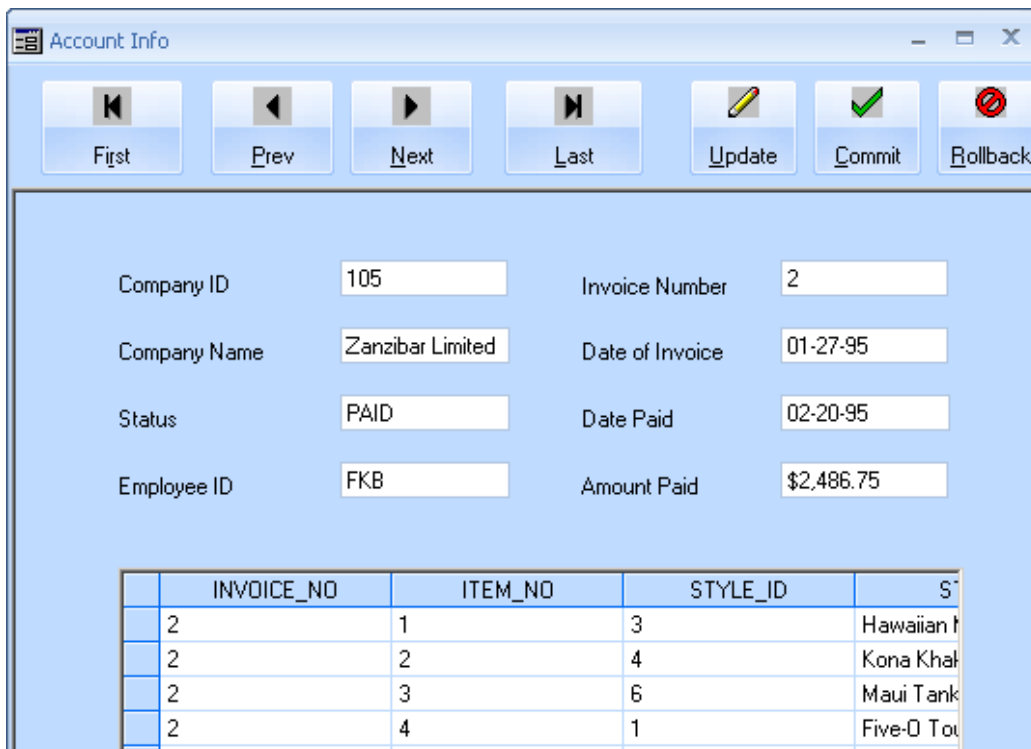
7. Click the **First**, **Next**, **Previous**, and **Last** push buttons to scroll through the available data.

   This application draws information from a sample database called ISLAND that is included with SQLBase. The ISLAND database includes account and product information for a fictitious company called Island T-Shirts. The Account Info application allows you to examine the clients and invoices listed in the ISLAND database. The table shows the active orders for the company name on the screen. The data fields show the status of the account.

8. Select **Close** from the System menu in the title bar of the Account Information window.
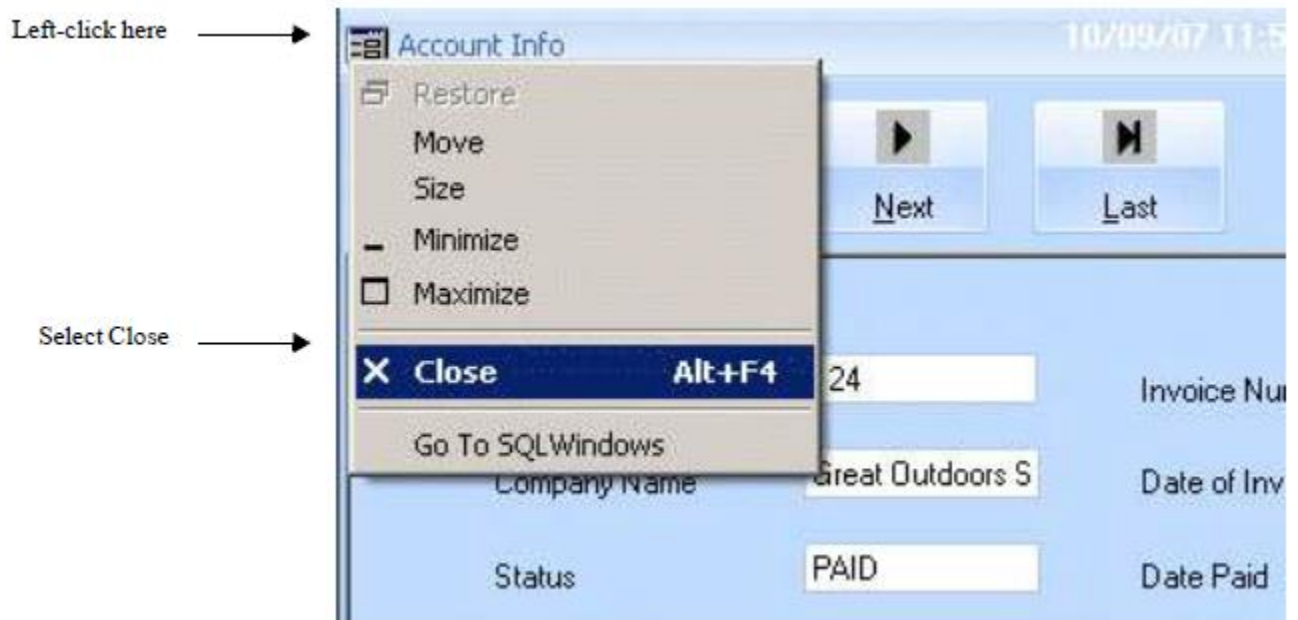
When you select Close, SQLWindows returns you to Designtime mode, where you can continue development on your application. You are now ready to build the AccountInfo.app application yourself.

9. To begin, select **New** from the **File** menu. Select **newapp.app** as the template for your application. Click **OK**. The Templates dialog box is displayed.
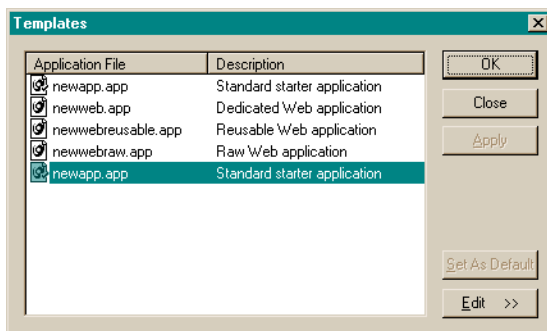
Creating AccountInfo.app: Overview

This sample application has two windows; a Login dialog and a form for displaying database information. To build the Login dialog:

- Create a dialog with three data fields and two push buttons.

- Code the data fields to accept a database, username, and password.

- Code the OK push button to accept the dialog information and open the Account Information window.

- Code the Cancel push button to cancel the dialog.

If you followed the steps in the previous section, you are running SQLWindows. A blank application template is ready for your use. Your workspace appears as follows.
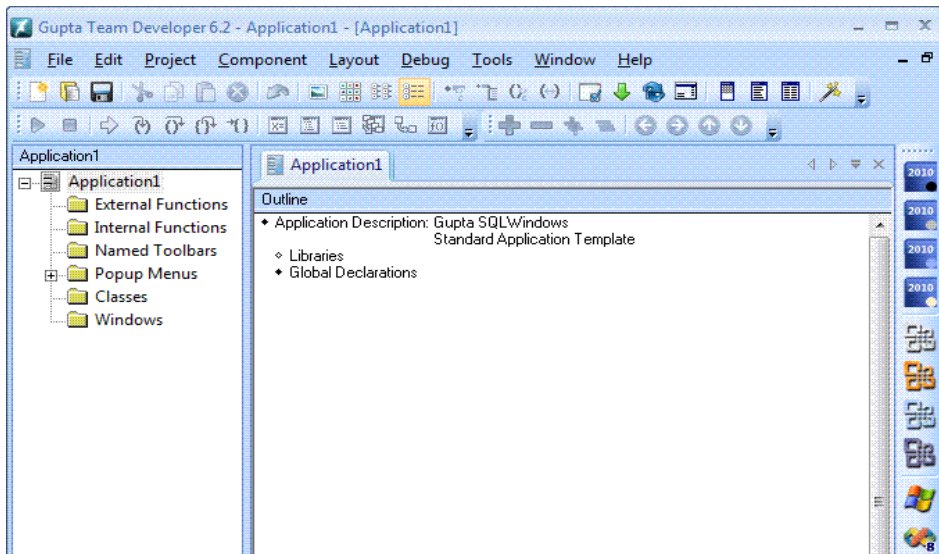
Creating the Login Dialog

To create the login dialog box for the application, follow these steps:

1. Right-click the Windows folder in the left hand pane. Select **New** > **Dialog Box**. A standard dialog window appears.
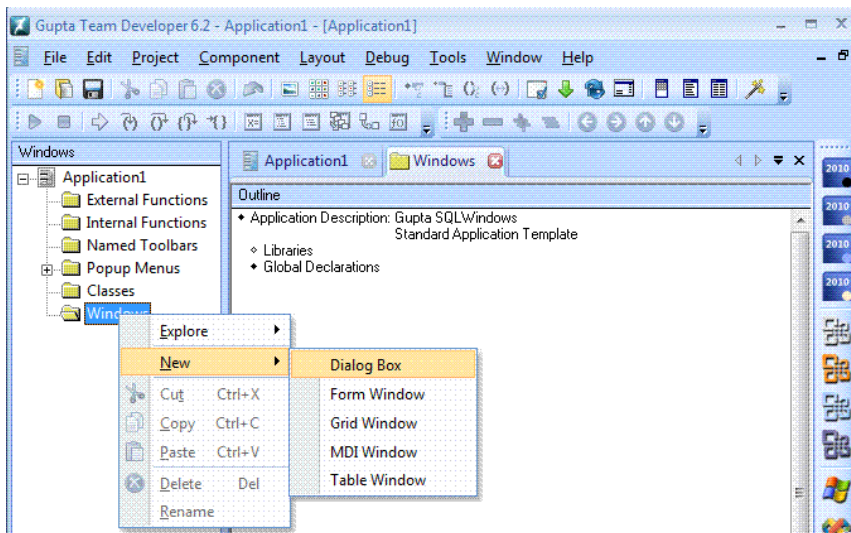
2. Select **Attribute Inspector** from the Tools menu. Type `dlgLogin` in the Dialog Box: field at the top, and `Database Login` in the Object Title field.

Attribute Inspector — Dialog Box: dlgLogin

Object Title	Database Login
Allow Child Docking	No
Allow Dock to Parent	
Docking Orientation	
Accessories Enabled	No
Tool Bar Visible	
Tool Bar Size	
Tool Bar Position	
Docking Tool Bar	
Tool Bar Docking Oriental	
Status Bar Visible	
Display Style	Default
Resizable	No
Type of Dialog	Modal
Absolute Screen Location	Yes
Left	Default
Top	Default
Width	Default
Height	Default
Vertical Scroll	
Horizontal Scroll	
Background Color	Default
Text Color	Default
Font Name	Default
Font Size	Default
Font Enhancement	Default
QuickObject DLL	[none]
XAML Style	[none]
Background Brush	[none]

dlgLogin

Database Login

3. Select **Controls** from the **Tools** menu to bring up the Controls toolbar. Select the standard Data Field control from the Controls palette, move your cursor over the dialog, and click to drop the data field on the dialog box.
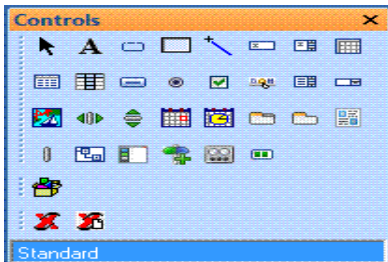
4. With the focus on the data field, go to the Attribute Inspector. Change the Data field name from df1 to dfDatabase.

5. From the Controls toolbar, drop a second standard data field under the first one in the dialog window. In the Attribute Inspector, change the Data Field name from df2 to dfUser. Next, drop a third standard data field under the second. In the Attribute Inspector, change the Data Field name from df3 to dfPassword. Arrange the data fields and resize the dialog as shown below:

14
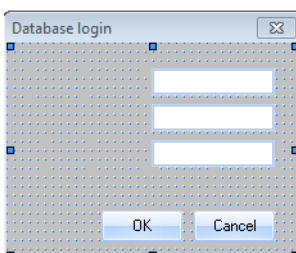
6. Label the data fields. Select the standard Background Text control from the Controls palette.
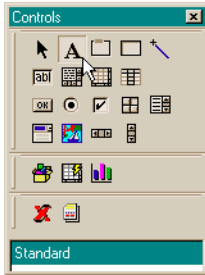
7. Drop a Background Text box next to each of the data fields. Label them as follows:

    Database

    User Name

    Password

    Your dialog should appear as shown below:
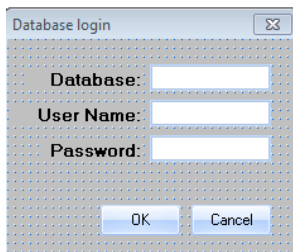
8. Select the Standard Push Button control from the Controls toolbar.
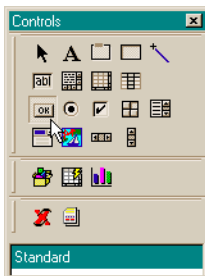
9. Place the push button object at the bottom of the dialog box and type OK. With the push button selected, open the Attribute Inspector. Change the push button name from pb1 to pbOK.

10. Select the Standard Push Button control from the Controls palette again. Drop a second push button on the dialog to the right of the **OK** push button. Type Cancel on the push button. In the Attribute Inspector, change the Pushbutton name from pb2 to pbCancel.

    Your dialog should now appear as shown below:
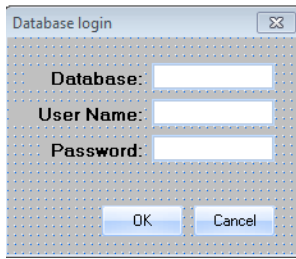
You must code functionality into the objects on your dialog. Some of that functionality connects the dialog to the form window of the application. Therefore, before coding the items in this dialog, you must create the form window.

Click the Outline tab in the right pane of SQLWindows. In the left pane, click Application1. Right-click the Windows section underneath it. Select New, Form Window from the pop-up menu. Type frm1. In the Attribute Inspector, there is an option for Automatically Create; change this to **No**. Type **Account Info** in the Object Title field. Your new form appears as shown below:
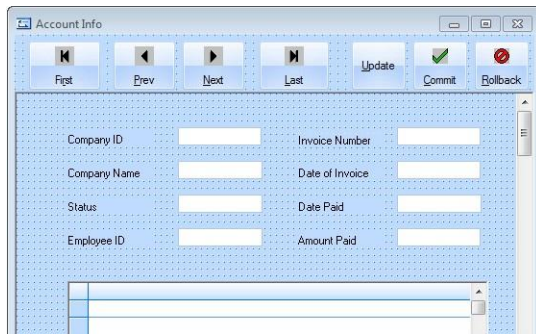
Coding the Login Dialog

This section provides details on how to code the fields and buttons of the Login dialog.

Defining Variables

To define the global variables for the application, follow these steps:

1. Click the **Outline** tab in the right pane.

    In the left pane, select **Application1**. The outline for the entire application is displayed in the right pane.

2. Double-click **Global Declarations** in the right pane, and then click **Variables**.

    Select **Coding Assistant** from the Tools menu.

    In the Coding Assistant, double-click **Sql Handle** to add it to the Variables section of your outline.

    Type `hSql` in the outline.

3. In the Coding Assistant, double-click **Boolean** to add it to your outline.

    Type `bConnect` in the outline.

    Your outline should appear as follows:
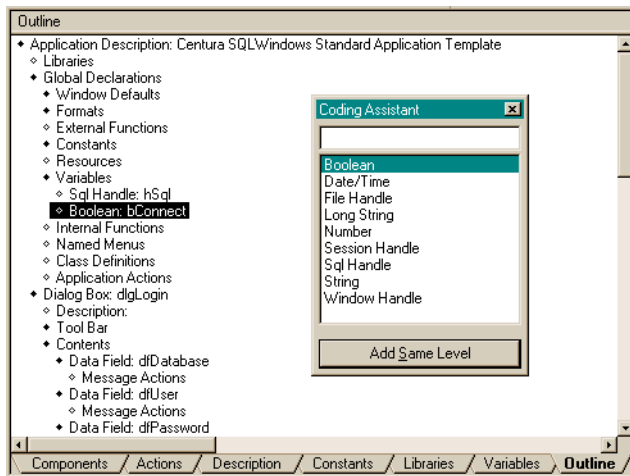
Coding the Application Start-up and Exit Functions

To code the start-up and exit functions, follow these steps:

1. Under **Global Declarations**, click **Application Actions**.

   In the Coding Assistant, double-click **On SAM_AppStartup**.

2. Double-click **If** in the lower part of the Coding Assistant.

   Select Sal Functions from the drop-down menu in the Coding Assistant. The Coding Assistant displays the available functions.

   Start typing SalModalDialog in the text box below the drop-down box in the upper part of the Coding Assistant to locate this function. Notice that the function name appears highlighted in the function list.

   Press **Enter** when SalModalDialog becomes highlighted.
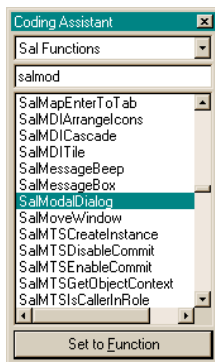
   If SalModalDialog(Template, Window_Handle)appears in your outline. Template, Window_Handle is highlighted.

   Type:

   ```
   dlgLogin, hWndNULL, 'ISLAND', 'SYSADM', 'SYSADM'
   ```

   This text replaces Template, Window_Handle. Your outline looks like this:

   ```
   Application Actions
      On SAM_AppStartup
         If SalModalDialog (dlgLogin, hWndNULL,'ISLAND','SYSADM','SYSADM')
   ```

3. Press **Enter** to insert a line.

Double-click **Call** in the lower part of the Coding Assistant.

Start typing `SalCreateWindow` in the text box below the drop-down box in the upper part of the Coding Assistant.

Press **Enter** when SalCreateWindow becomes highlighted in the function list.

Call SalCreateWindow(Template, Window_Handle) appears in your outline.

Replace the Template, Window_Handle highlighted text with frm1, hWndNULL.

4. Select **On SAM_AppStartup**. The code you select in the outline determines where SQLWindows inserts the next call.

5. Double-click **On SAM_AppExit** in the upper part of the Coding Assistant.

   Double-click **If** in the lower part of the Coding Assistant.

   Type `bConnect`, so that your outline now reads `If bConnect`.

   Press **Enter**.

6. Double-click **Call** in the <u>lower</u> part of the Coding Assistant.

   Start typing `SqlDisconnect` in the text box below the list combo box in the upper part of the Coding Assistant.

   Press **Enter** when SqlDisconnect becomes highlighted in the function list.

Note: If the Coding Assistant does not already display "SAL Functions" or "SAL+User Functions" in the uppermost combo box after you double-click Call, you should drop down the combo box and select one of those two values and begin typing SQLDisconnect in the text box.

`Call SqlDisconnect( Sql_Handle )` appears in your outline.

Replace the highlighted Sql_Handle text with hSql.

The outline should appear as follows:

```
On SAM_AppStartup
   If SalModalDialog( dlgLogin, hWndNULL, 'ISLAND', 'SYSADM', 'SYSADM' )
      Call SalCreateWindow( frm1, hWndNULL )
On SAM_AppExit
   If bConnect
      Call SqlDisconnect( hSql )
```

Defining Parameters

To define the parameters for the data fields, follow these steps:

1. Under the **dlgLogin** section in the outline, select **Window Parameters**.

   In the Coding Assistant, double-click **String**, type `strDefDatabase`, and press **Enter**.

2. Double-click **String** again, and type `strDefUser`. Press **Enter**.

3. Double-click **String** again, and type `strDefPassword`. The outline should appear as follows:

```
Window Parameters  String: strDefDatabase  String: strDefUser  String:
strDefPassword
```

Coding the Data Fields

To code the data fields in the Login dialog box, follow these steps:

1. Under Dialog Box: dlgLogin, Contents, double-click Data Field: dfDatabase in the outline so that Message Actions is showing.

   Select Message Actions under the Data Field: dfDatabase section.

   Double-click **ON SAM_Create** in the Coding Assistant.

   Double-click **Set** in the lower part of the Coding Assistant.

2. In the outline, after **Set**, type `dfDatabase = strDefDatabase`.

   The statement should appear as follows:

   ```
   Set dfDatabase = strDefDatabase
   ```

3. Repeat steps 1 and 2 for dfUser (Set dfUser = strDefUser) and dfPassword (Set dfPassword = strDefPassword).

   The outline should appear as follows:

   ```
   Data Field: dfDatabase
      Message Actions
         On SAM_Create
            Set dfDatabase = strDefDatabase
            Set dfUser = strDefUser
            Set dfPassword = strDefPassword
   ```

Coding the OK Push Button

To code the OK push button in the Login dialog box, follow these steps:

1. Double-click Pushbutton: pbOK in the outline.

   Highlight Message Actions under the Pushbutton: pbOK section.

   Double-click **ON SAM_Click** in the Coding Assistant.

   Double-click **Set** in the lower part of the Coding Assistant.

2. In the outline after Set type SqlUser = dfUser, press **Enter**.

3. Double-click **Set** in the Coding Assistant, type SqlPassword = dfPassword, and press **Enter**.

4. Double-click **Set** in the Coding Assistant, type SqlDatabase = dfDatabase, and press **Enter**.

Now you call a function to change the cursor to an hour glass to show that the application is busy.

5. Double-click **Call** in the Coding Assistant.

   Start typing SalWaitCursor in the text box below the list combo box in the upper part of the Coding Assistant.

   Press **Enter** when SalWaitCursor becomes highlighted.

6. Call SalWaitCursor(Boolean) appears in your outline.

   Boolean is already highlighted. Type `TRUE` over it.

Now set a function to allow connection to the database.

7. Double-click **Set** in the Coding Assistant and type bConnect = SqlConnect(hSql)

Now you call a function to change the cursor back from an hour glass to show that the application is no longer busy.

8. Double-click **Call** in the Coding Assistant.

    Start typing `SalWaitCursor` in the text box below the list combo box in the upper part of the Coding Assistant to find this function.

    Press **Enter** when SalWaitCursor becomes highlighted.

    Call SalWaitCursor(Boolean) appears in your outline. Boolean is already highlighted. Type **FALSE** over it.

9. Double-click **If** in the Coding Assistant and type `bConnect`.

10. Double-click **Call** in the lower part of the Coding Assistant.

    Start typing `SalEndDialog` in the text box below the list combo box in the upper part of the Coding Assistant.

    Press **Enter** when SalEndDialog becomes highlighted.

    Call SalEndDialog(Window_Handle, Number) appears in your outline. Window_Handle, Number are already highlighted.

    Type **dlgLogin, TRUE** over them. Your outline looks like this:

```
Pushbutton: pbOK
   Message Actions
      On SAM_Click
         Set SqlUser = dfUser
         Set SqlPassword = dfPassword
         Set SqlDatabase = dfDatabase
         Call SalWaitCursor( TRUE )
         Set bConnect = SqlConnect ( hSql )
         Call SalWaitCursor( FALSE )
         If bConnect
            Call SalEndDialog( dlgLogin, TRUE )
```

Coding the Cancel Push Button

To code the Cancel push button in the Login dialog box, follow these steps:

1. Double-click **Pushbutton: pbCancel** in the outline.

    Highlight **Message** Actions under the Pushbutton: pbCancel section.

    Double-click **ON SAM_Click** in the Coding Assistant.

1. Double-click **Call** in the lower part of the Coding Assistant.

    Start typing `SalQuit` in the text box below the list combo box in the upper part of the Coding Assistant to find this function.

    Press **Enter** when SalQuit becomes highlighted. Call SalQuit( ) appears in your outline. Your outline looks like this:

```
Pushbutton: pbCancel
   Message Actions
      On SAM_Click
         Call SalQuit( )
```

Coding the Dialog Box to Open on Start-up

To code the Login dialog box to open on application start-up, follow these steps:

1. Click **Message Actions** under Dialog **Box: dlgLogin**.

   Double-click **On SAM_Create** in the Coding Assistant.

2. Double-click **Call** in the Coding Assistant.

   Start typing `SalCenterWindow` in the text box below the list combo box in the upper part of the Coding Assistant to find this function.

   Press **Enter** when SalCenterWindow becomes highlighted.

   Call SalCenterWindow(Window_Handle) appears in your outline. Window_Handle is already highlighted. Type `hWndForm` over it.

3. Double-click **Call** in the Coding Assistant.

   Double-click **SalSetDefButton** in the Coding Assistant.

   Call SalSetDefButton(Window_Handle) appears in your outline.

   Window_Handle is already highlighted. Type `pbOK` over it. Your outline looks like this:

```
Message Actions
   On SAM_Create
      Call SalCenterWindow( hWndForm )
      Call SalSetDefButton( pbOK )
```

You have just finished building and coding your Login dialog and:

- Created a dialog with three data fields and two push buttons.

- Coded the data fields to accept a database, username, and password.

- Coded the OK push button to accept the dialog information and open the Account Information window.

- Coded the Cancel push button to destroy the dialog when clicked.

Running the Application

Before laying out the form window, compile your application and see how it runs.

1. Select Go from the Debug menu.

Note: You can also select **Go** from the **Project** menu, or click the Compile (checkmark) icon on the toolbar.

   SQLWindows asks you if you want to save changes to the outline.
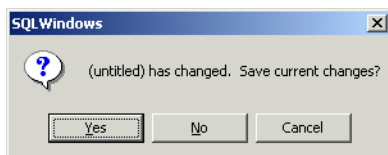
2. Click **Yes**.

A dialog appears where you can specify the directory where you want to store your application and specify the name of the application.

3. Type **Account.app** in the Filename field.
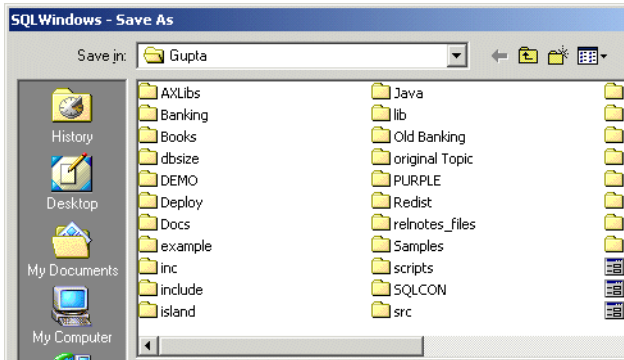
Click **Save**.

The application compiles. Your Database Login dialog appears.
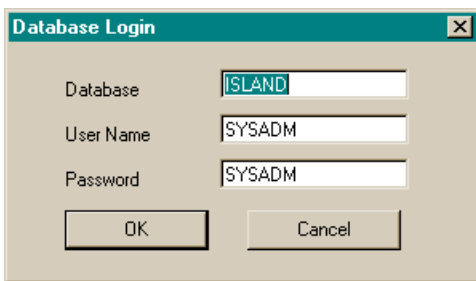
4. Click **OK**. Your form window appears.

5. To return to design mode, select **Close** from the system menu on your form window.
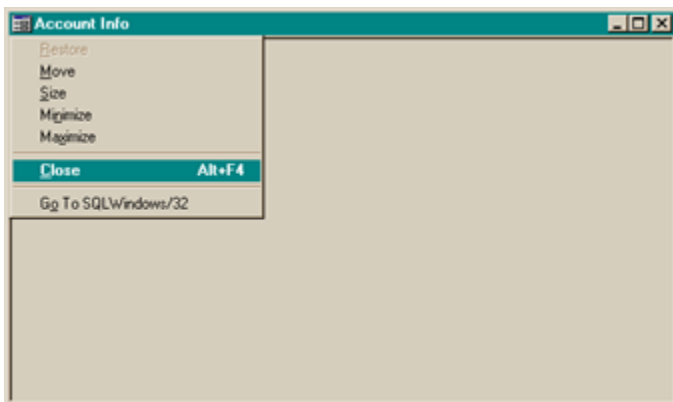
Creating the Form Window

Now you are ready to lay out your Account Information window. You want to drop eight data fields and their labels for viewing account formation. You also want to add a table window so you can view order information for each account.

1. Right-click the **frm1** window in the Outline.

Select **Preview Window** from the menu that appears. The desktop should appear as shown below:
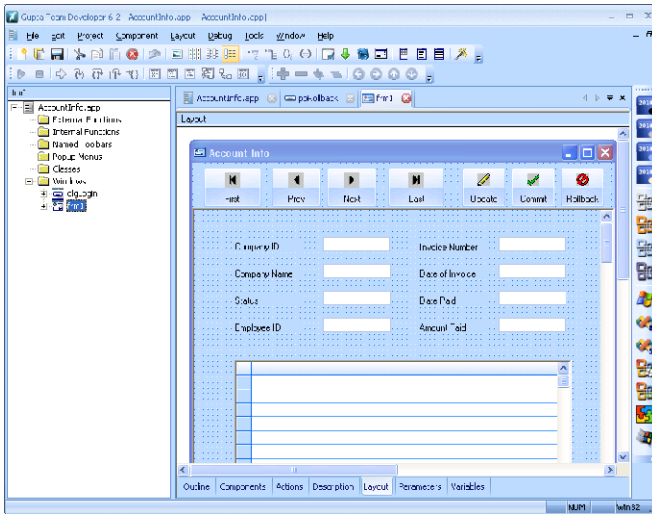
2. Click the standard background text control on the Controls palette and drop this on the left side of the form.

   Enter the text `Company ID`. Repeat this step for each of the following labels:

   ```
   Invoice Number
   Company Name
   Date of Invoice
   Status
   Date Paid
   Employee ID
   Amount Paid
   ```

   Make two columns of labels on the form.

Note: An easy way to align fields and labels so they look nice is to hold down the **Shift** key and click on the various fields until they are all selected. Then, choose **Align** from the **Layout** menu.

   Your form looks like this:
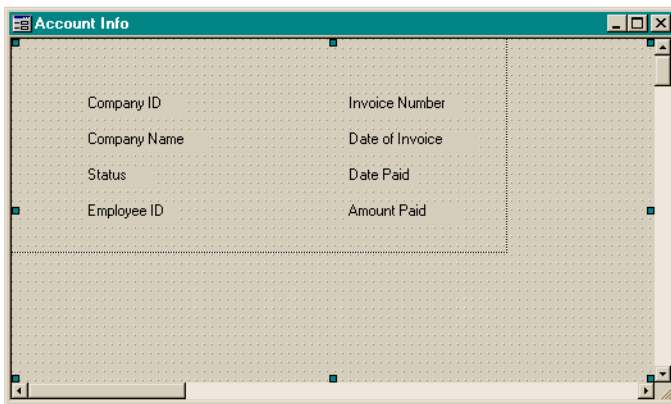
3. Select the standard data field from the Controls palette and drop a data field next to each label, with attributes for each as follows (use the Attribute Inspector):

Field Label	Object Name	Data Type
Invoice Number	dfINVOICE_NO	Number

Company ID	dfCOMPANY_ID	Number
Company Name	dfCOMPANY_NAME	String
Date of Invoice	dfINVOICE_DATE	Date/Time
Date Paid	dfDATE_PAID	Date/Time
Status	dfSTATUS	String
Amount Paid	dfAMOUNT_PAID	Number
Employee ID	dfEMPLOYEE_ID	String

Your form looks like this:
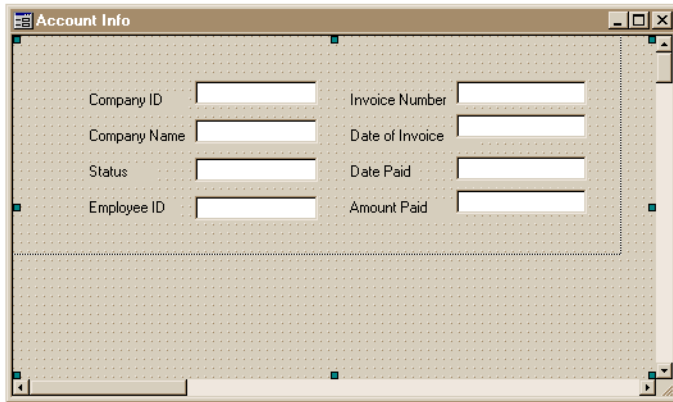
4. Click the data field next to Date of Invoice. Use the Attribute Inspector to change the **Format** to **MM-dd-yy**. Repeat this step for Date Paid.

5. Select the standard child table window from the Controls palette and drop this at the bottom of the form.
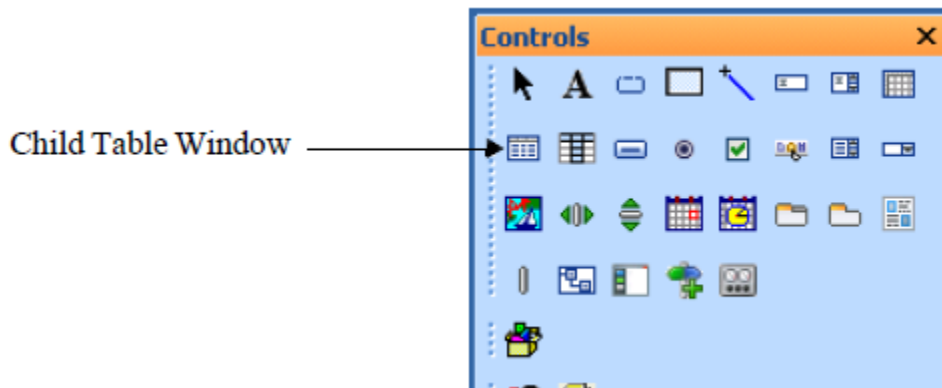
6. In the Attribute Inspector, name the Child Table tblINVOICE_ITEM.

Your form now looks like this:
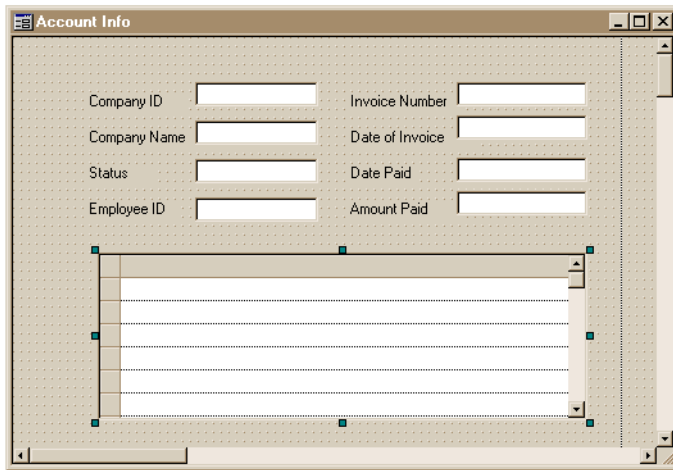
Populating the Form Using SQL and SAL

This section describes had to add code to the various elements of the form window, including the data fields and child table window.

Populating the Data Fields

To code the form's data fields to display the corresponding information from the database, follow these steps:

1. In the outline, click **Form Window: frm1**.

    Click **Functions.**

    Double-click **Function** in the Coding Assistant and type `PopulateFormWindow` in the outline.

2. Click **Window Variables**.

    In the Coding Assistant, double-click **Number**, and type `nFetch`.

    Click **Window Variables** again, double-click **Number**, and type `nLastRecord`.

3. Click Functions.

    Click **Actions**, and the double-click **Call** in the Coding Assistant.

    Double-click SqlPrepare in the Coding Assistant.

    Call SqlPrepare(Sql_Handle, String) is added to the outline.

    Replace Sql_Handle, String with:

```
hSql, 'SELECT
  INVOICE_NO,
  COMPANY_ID,
  COMPANY_NAME,
  INVOICE_DATE,
  DATE_PAID,
  STATUS,
  AMOUNT_PAID,
  EMPLOYEE_ID
FROM INVOICE INTO
  :dfINVOICE_NO,
  :dfCOMPANY_ID,
```

```
    :dfCOMPANY_NAME,
    :dfINVOICE_DATE,
    :dfDATE_PAID,
    :dfSTATUS,
    :dfAMOUNT_PAID,
    :dfEMPLOYEE_ID' )
```

Note: When you enter this code without the Coding Assistant, you begin by pressing the **Insert** key. Then, type the programming statement. To add the next statement, press **Enter**. To continue a programming statement to the next line, press **CTRL + Enter**.

To indent the current line for a programming statement, hold down the **Alt** key and press the right arrow key. If you are indenting for readability only, use the **Tab** key.

4. Double-click **Call** in the Coding Assistant.

    Double-click **SqlExecute** in the Coding Assistant.

    Call SqlExecute(Sql_Handle)is added to the outline.

    Replace Sql_Handlewith hSql.

5. Double-click **Call** in the Coding Assistant.

    Double-click **SqlGetResultSetCount** in the Coding Assistant.

    Replace Sql_Handle, Numberwith hSql, nLastRecord.

6. Double-click **Call** in the Coding Assistant.

    Double-click **SqlFetchNext** in the Coding Assistant.

    Call SqlFetchNext(Sql_Handle, Number)is added to the outline.

    Replace Sql_Handle, Numberwith hSql, nFetch.

    Your code looks like this:

```
Actions
  Call SqlPrepare( hSql, 'SELECT
      INVOICE_NO,
      COMPANY_ID,
      COMPANY_NAME,
      INVOICE_DATE,
      DATE_PAID,
      STATUS,
      AMOUNT_PAID,
      EMPLOYEE_ID
FROM INVOICE
INTO
      :dfINVOICE_NO,
      :dfCOMPANY_ID,
      :dfCOMPANY_NAME,
      :dfINVOICE_DATE,
      :dfDATE_PAID,
      :dfSTATUS,
      :dfAMOUNT_PAID,
      :dfEMPLOYEE_ID' )
Call SqlExecute( hSql )
Call GetResultSetCount (hSql, nLastRecord)
Call SqlFetchNext( hSql, nFetch )
```

Populating the Child Table Window

To add the code necessary to making the child table window functional, follow these steps:

1. In the outline, double-click Child Table: tblINVOICE_ITEM. Click Window Variables.

   Double-click **Sql Handle** in the Coding Assistant. Type `hSqlTable` for the variable name in the outline.

2. Click **Functions**, then double-click **Function** in the Coding Assistant and type `PopulateChildTable`.

3. Click **Actions**.

   Double-click **Call** in the Coding Assistant.

   Double-click **SalTblReset** in the Coding Assistant

   Call SalTblReset(Window_Handle)is added to the outline.

   Replace Window_Handle with tblINVOICE_ITEM.

   Press **Enter**.

4. Double-click **Call** in the Coding Assistant.

   Double-click **SalTblPopulate** in the Coding Assistant.

   Call SalTblPopulate(Window_Handle, Sql_Handle, String, Number)is added to the outline.

   Replace (Window_Handle, Sql_Handle, String, Number) with

   ```
   ( tblINVOICE_ITEM, hSqlTable, 'SELECT
      INVOICE_NO,
      ITEM_NO,
      STYLE_ID,
      STYLE,
      COLOR,
      QUANTITY,
      ITEM_PRICE
   FROM INVOICE_ITEM
      WHERE INVOICE_NO = :dfINVOICE_NO', TBL_FillNormal )
   ```

5. Click **Message Actions** for ChildTable:tblINVOICE_ITEM.

   Double-click **On SAM_Create** in the Coding Assistant.

   Double-click **Call** in the lower part of the Coding Assistant.

   Double-click **SqlConnect** in the Coding Assistant.

   Call SqlConnect(Sql_Handle)is added to the outline.

   Replace Sql_Handle with hSqlTable.

6. Click **On SAM_Create** in the outline. This puts you at the correct level of indentation for your next function call.

7. Double-click **On SAM_Destroy** in the Coding Assistant.

   Double-click **Call** in the lower part of the Coding Assistant.

   Double-click **SqlDisconnect** in the Coding Assistant

   Call SqlDisconnect(Sql_Handle)is added to the outline.

   Replace Sql_Handle with hSqlTable.

   The outline looks like this:

```
+ Child Table: tblINVOICE_ITEM|
   ◇ Contents
   + Functions
      + Function: PopulateChildTable
         ◇ Description:
         ◇ Returns
         ◇ Parameters
         ◇ Static Variables
         ◇ Local variables
         + Actions
            ◇ Call SalTblReset( tblINVOICE_ITEM )
            ◇ Call SalTblPopulate( tblINVOICE_ITEM, hSqlTable, 'SELECT
                              INVOICE_NO,
                              ITEM_NO,
                              STYLE_ID,
                              STYLE,
                              COLOR,
                              QUANTITY,
                              ITEM_PRICE
          FROM INVOICE_ITEM
                  WHERE INVOICE_NO = :dfINVOICE_NO', TBL_FillNormal )
   + Window Variables
      ◇ Sql Handle: hSqlTable
   + Message Actions
      + On SAM_Create
         ◇ Call SqlConnect( hSqlTable )
      + On SAM_Destroy
         ◇ Call SqlDisconnect( hSqlTable )
```

Completing the Application

To call a function to populate the data fields, follow these steps:

1. Click Form Window: frm1. Click Message Actions.

    In the Coding Assistant, double-click **On SAM_CreateComplete**.

    Double-click **Call** in the lower part of the Coding Assistant.

    Double-click `PopulateFormWindow` in the Coding Assistant.

    Call PopulateFormWindow ( ) is added to the outline.

    There are no parameters needed, so this call is complete.

2. Click **Form Window: frm1** again.

    Double-click **Functions**.

    Double-click Function: PopulateFormWindow.

    Double-click Actions.

    Click the statement **Call SqlFetchNext(hSql, nFetch)**.

    Press **Insert**. This inserts a new line.

    Type:

    Call tblINVOICE_ITEM.PopulateChildTable( )

    This makes a call to the Populate function of the Child Table, so that the Child Table is populated.

Add Navigation Controls

1. Right-click on **frm1** in the left pane. Select **Preview Window**.

2. In the Attribute Inspector for the form, change Accessories from **No** to **Yes**.

3. Select a standard push button from the Controls palette and drop the push button on the left corner of the toolbar. Type `First` in the Object title field.

4. In the Attribute Inspector for the push button, name the push button `pbFirst`. Go to File Name: and click the ellipse (…). The Open dialog box appears. Find **toprec.bmp** (you may need to navigate up one directory). Click **Open**.

Repeat steps 3-4 for three more push buttons you add to the toolbar:

- Next (pbNext, Next, nextrec.bmp)

- Previous (pbPrev, Prev, prevrec.bmp)

- Last (pbLast, Last, lastrec.bmp)

The competed form appears as follows:
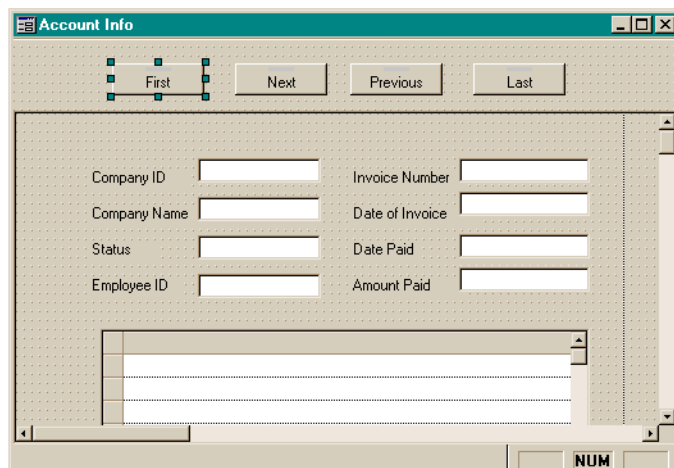
SAL for Navigation Buttons

By now you are used to writing SAL code using the SQLWindows user interface. Use the following handy table of instructions to enter SAL code that activates functionality in the push buttons on your form.

In the…	Do this…	To this
Outline	Click	Form Window
Outline	Click	Window Variables
Coding Assistant	Double-click	Number
Outline	Type	nLastRecord
Outline	Double-click	Toolbar
Outline	Double-click	Contents

Coding the First Push Button

To code the First push button, follow these steps:

1. In the outline, double-click **Pushbutton: pbFirst**.

    Click **Message Actions**.

    In the Coding Assistant, double-click **On SAM_Click**, and double-click **Call** in the lower part.

2. In the Coding Assistant, double-click **PopulateFormWindow**.

    Your code for this push button looks like this:

```
Pushbutton: pbFirst
   Message Actions
      On SAM_Click
         Call PopulateFormWindow( )
```

Coding the Prev Push Button

To code the Prev push button, follow these steps:

1. In the outline, double-click **Pushbutton:pbPrev**. Click **Message Actions**.

2. In the Coding Assistant, double-click **On SAM_Click**, double-click **If,** and type NOT.

3. In the Coding Assistant, double-click SqlFetchPrevious.

    SqlFetchPrevious(Sql_Handle, Number) is added to the outline.

    Replace Sql_Handle, Number with hSql, nFetch.

    The statement reads: If NOT SqlFetchPrevious(hSql, nFetch)

4. Double-click **Call** in the lower part of the Coding Assistant.

    Double-click **SalMessageBox** in the Coding Assistant.

    Call SalMessageBox(String, String, Number) is added to the outline.

    Replace String, String, Number with:

    ```
    'There are no previous records', 'End Of Fetch', MB_Ok.
    ```

5. Click the statement `If NOT SqlFetchPrevious( hSql, nFetch )`.

    Double-click **Else** in the upper part of the Coding Assistant.

    Press the **Insert** key; this creates a new blank line.

    Press the **ALT** + right arrow; this properly indents the outline. Type:

    ```
    Call tblINVOICE_ITEM.PopulateChildTable( )
    ```

Your code for this push button looks like this:

```
Pushbutton: pbPrev
   Message Actions
      On SAM_Click
         If NOT SqlFetchPrevious( hSql, nFetch )
            Call SalMessageBox( 'There are no previous records', 'End of
Fetch', MB_Ok )
         Else
            Call tblINVOICE_ITEM.PopulateChildTable( )
```

Coding the Next Push Button

To code the Next push button, follow these steps:

1. Double-click **Pushbutton: pbNext**.

   Click **Message Actions**.

   In the Coding Assistant, double-click **On SAM_Click**.

   Double-click **If** in the lower part of the Coding Assistant, and type **NOT**.

   In the Coding Assistant, double-click **SqlFetchNext**.

   SqlFetchNext(Sql_Handle, Number) is added to the outline.

   Replace `Sql_Handle, Number` with `hSql, nFetch` and press **Enter**.

   The statement now reads:

   `If NOT SqlFetchNext( hSql, nFetch )`

2. Double-click **Call** in the lower part of the Coding Assistant.

   Double-click **SalMessageBox** in the Coding Assistant.

   Replace `String, String, Number` with:

   `'There are no more records', 'End Of Fetch', MB_Ok`

3. Click the statement `If NOT SqlFetchNext( hSql, nFetch )`.

   Double-click **Else** in the upper part of the Coding Assistant.

   Press **Enter**.

   Press the **Insert** key.

   Press the **ALT** + right arrow (to indent the line).

   Type Call tblINVOICE_ITEM.PopulateChildTable( )

Your code for this pushbutton looks like this:

```
Pushbutton: pbNext  Message Actions
   On SAM_Click
      If NOT SqlFetchNext( hSql, nFetch )
         Call SalMessageBox( 'There are no more records', 'End of Fetch',
MB_Ok )
      Else
         Call tblINVOICE_ITEM.PopulateChildTable( )
```

Coding the Last Push Button

To code the Last push button, follow these steps:

1. Double-click **Pushbutton: pbLast**.

   Click **Message Actions**.

   In the Coding Assistant, double-click **On SAM_Click**.

   Double-click **If** in the lower part of the Coding Assistant.

   In the Coding Assistant, double-click **SqlFetchRow**.

If SqlFetchRow(Sql_Handle, Number, Number) is added to the outline. Replace Sql_Handle, Number, Number with:

hSql, nLastRecord - 1, nFetch.

2. Press **Enter**.

    Press **Insert** (to create a new blank line).

    Press **ALT** +the right arrow key (to indent the outline).

    Type:

```
Call tblINVOICE_ITEM.PopulateChildTable( )
```

Your code for this push button looks like this:

```
Pushbutton: pbLast  Message Actions
   On SAM_Click
      If SqlFetchRow( hSql, nLastRecord - 1, nFetch )
         Call tblINVOICE_ITEM.PopulateChildTable( )
```

You have now added navigational push buttons. They let the user browse to and fro in the database records. You are finished building the Account Information application.

Running the Application

To compile and run the application, follow these steps:

1. Select **Go** from the **Debug** menu.

Note: You can also select **Go** from the **Project** menu, or click the Compile (checkmark) icon on the toolbar.

    SQLWindows asks you if you want to save changes to the outline.

2. Click **Yes**.

    The application compiles. Your Database Login dialog appears.
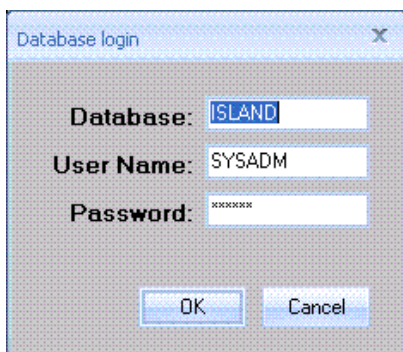
3. Click **OK**. Your form window appears.

4. Use the push buttons at the top to browse through various records.

M

Frst   **Erev**   !!ext   !ast   !pdate   bOmmit   Rollback

Company ID	105	Invoice Numbe1	2
Company Name	Zanzibar Limited	Date of Invoice	01-27-35
Status	'AID	Date Paid	02-20-35
Employee ID	KB	Amount Paid	$2,486.75

INVOICE_NO	ITEM_NO	STYLE_ID	S·
2		3	Hawaiian I
2	2	4	Kona Khal
2	3	6	MauiTank
2	4		Five-0 Toi
2	5	5	Lanai Ling

NUM

Congratulations! You've just completed building and running an application. In the chapters ahead, you will discover the convenience of Gupta ActiveX support for applications and reports, and the power of object-oriented programming.

Chapter 2 – Desktop and Components

This chapter describes the following Team Developer tools:

- SQLWindows

- Database Explorer

- The Visual ToolChest class library

- Dynalibs - dynamically linked libraries

- Team Object Manager

- QuickObjects

- SQLBase database engine

- Team Developer native connectivity

SQLWindows

The most recent versions of Windows are 32-bit, pre-emptive multitasking operating systems that run on the latest generation of microprocessors. Native 32-bit applications provide scalability, performance and robustness for demanding mission- critical business applications. Team Developer applications run as native 32-bit applications on Windows XP, 7, 8, Server 2003, Server 2008, Server 2012 or VISTA. You use SQLWindows to build these 32-bit applications.

SQLWindows has a user interface that closely matches the look and feel of Windows. The primary UI control is similar to the Windows Explorer, where navigation is by components in the left pane (tree view) of a window, and the right pane (tab view) of the window shows the contents of the component selected.
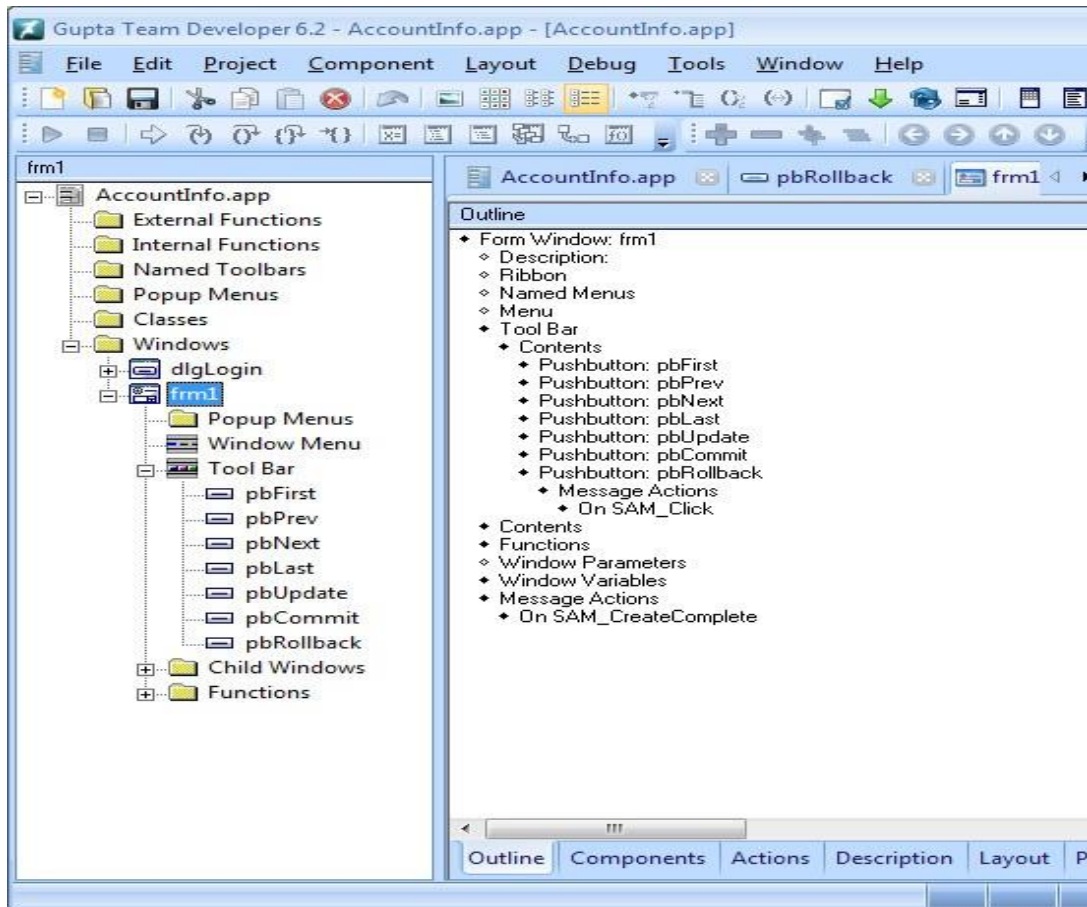
SQLWindows was designed as a fully 32-bit environment to maximize scalability. It was built using the Win32 interface. SQLWindows or 32-bit applications built with SQLWindows do not run on Windows 3.1. Programmers who need to deploy applications on Windows 3.1 must build their applications using the 16-bit version of SQLWindows (i.e. version 1.1.1). You can covert 16-bit windows applications to 32- bit SQLWindows applications, but the reverse is not possible. The most effective way to build scalable and fast applications is to use Win32 on Windows XP, 7, 8, 2003 Server, 2008 Server, 2012 Server and Vista which Gupta fully supports.

SQLWindows Desktop

You have already been introduced to SQLWindows in Chapter 1 where you built SQLWindows applications. Here, you get a more detailed explanation of some of the desktop features.

The desktop is a highly graphical and easy-to-use environment. Some of the tolls include a toolbar, an application window, tabbed application views, status bar, and a Controls toolbar make up the SQLWindows environment.
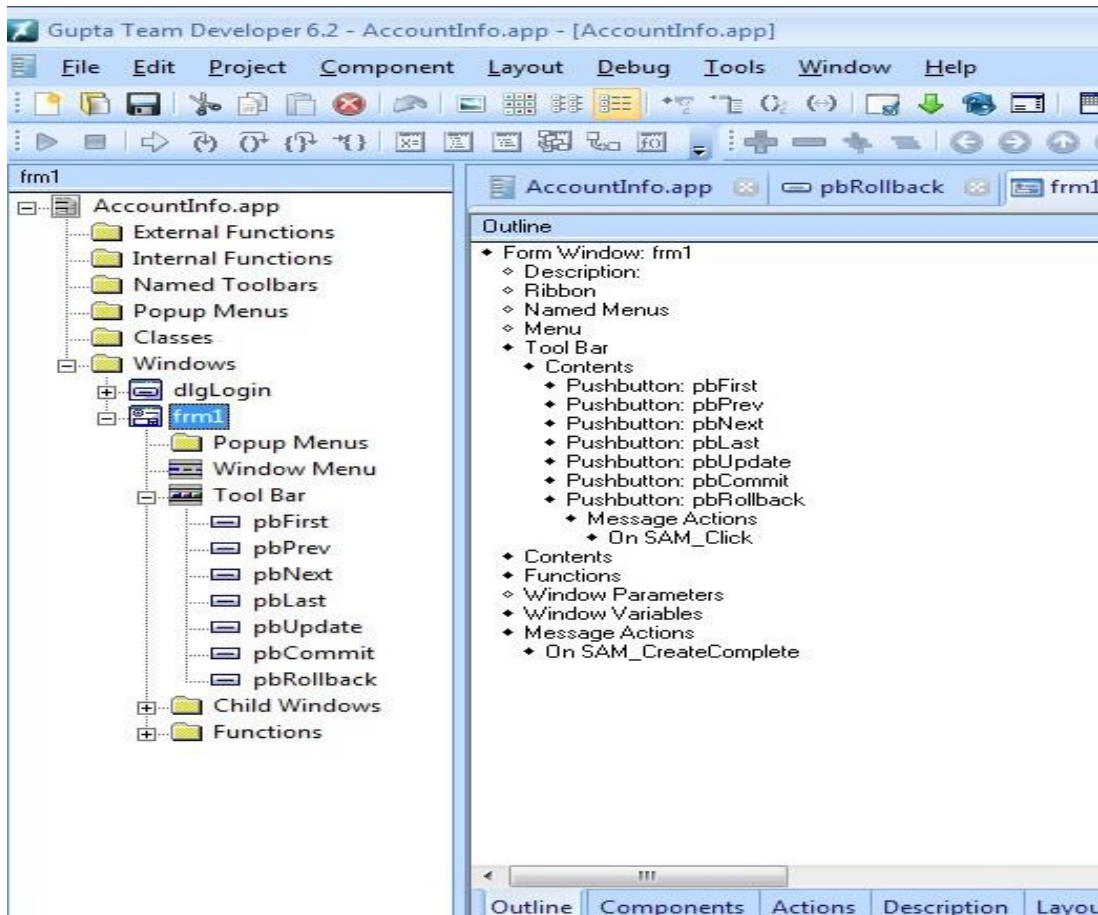
Toolbar

These tools assist you in creating the various parts of an application, navigating through it, and enhancing it as your project grows.

- Use the **toolbar** icons to build and edit your application quickly and easily.

- Use the application **tree view** and **tab views** to navigate through the sections of your code and as a visual summary of your application. Either view can be scrolled vertically with a wheel mouse.

- Use the tab views to see various presentations of different sections of your application code. As you build an application, SQLWindows automatically adds the appropriate items to your outline.

- Use the **status bar** to see the setting of the Num Lock, Scroll Lock, and Caps Lock keys, and get other helpful information.

- Use the **Controls toolbar** to add graphical objects like push buttons, data fields, and ActiveX objects to your layout.

The toolbars provide icons that help you build and edit your application quickly and easily. They also provide quick access to various components of Team Developer.
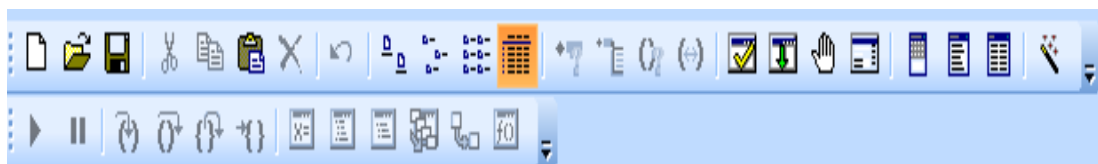
You can customize your toolbar in SQLWindows. The following picture shows you a typical set of SQLWindows toolbars.

Each icon supports Tooltips. When you place your cursor over the icon, a label displays the capability of the icon.

Customizing the toolbar

1. Select Toolbars from the Tools menu to open the Toolbars dialog.

2. Check the bars you want on your desktop.
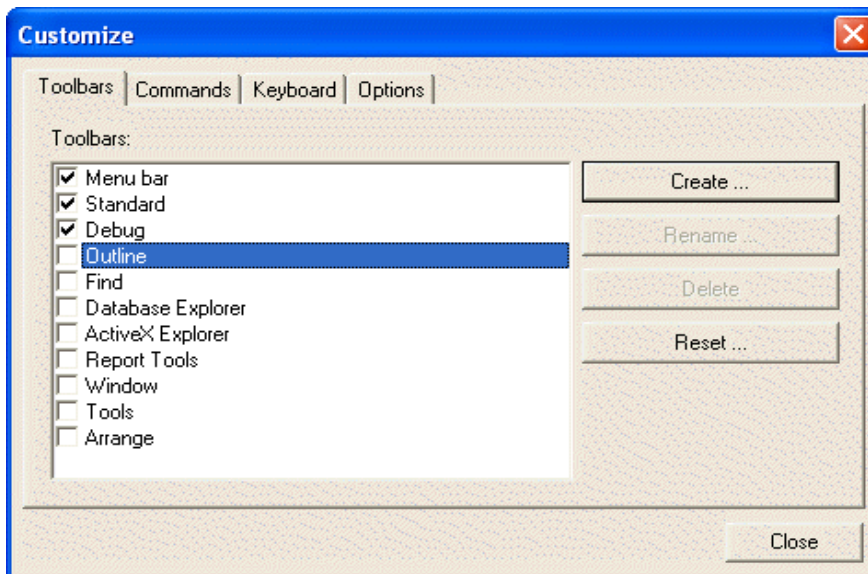
Each tool bar listed in this dialog has several tools associated with it. Click on various options to see how you effect your desktop.

3. Click **Tools…** to open the Tools dialog if you want to add more executable tools. In this dialog, you can select individual tools to add to the Tools toolbar and specify commands to make available to run from the tool bar.
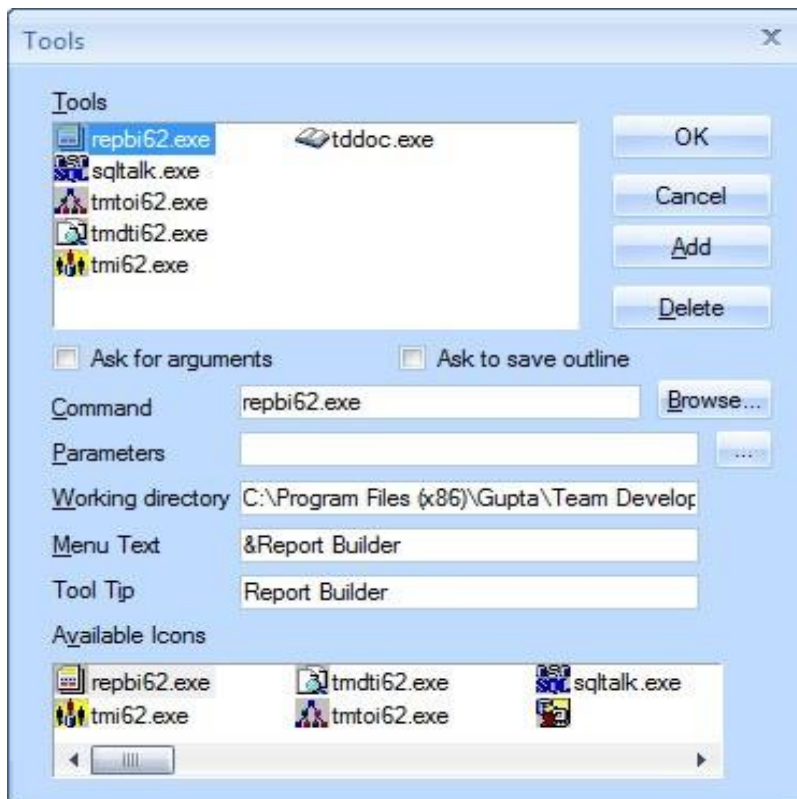
Status bar

SQLWindows has a status bar at the bottom that shows the settings of the NUM Lock, Scroll Lock and Caps Lock keys. The status bar also displays a message that shows the currently selected item on the menu or tool bar and the currently selected object on a form or outline view.

Turn on the status bar by right-clicking on a blank section of the toolbar. Select Status Bar from the menu. Turn the status bar off by either right-clicking on the status bar and selecting Status Bar from the menu or by right-clicking on a blank section of the toolbar and selecting Status Bar.
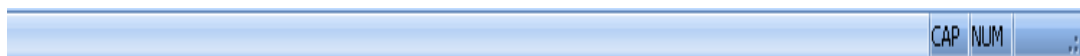
Controls toolbar

This toolbar is dockable. The dockable Controls toolbar contains icons representing a rich choice of graphical objects you can easily add to an application as part of your interface design.
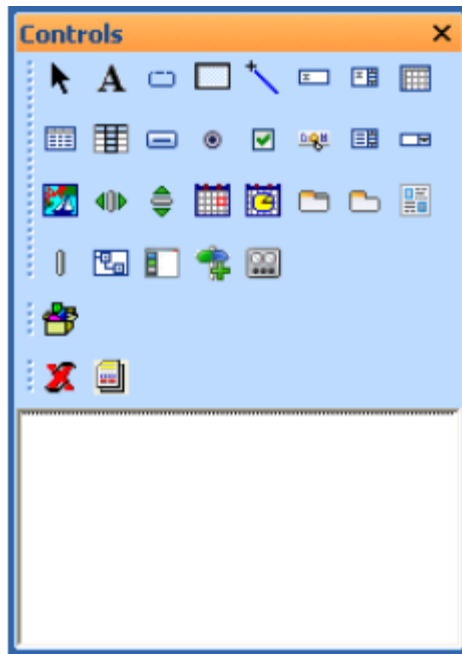
Icon	Name
↗	Grabber
A	Background Text
▭	Group Box
☐	Frame
↖	Line
abl	Data Field
▦	Multi-line Textfield
▦	Grid
▦	Child Table Window
▤	Table Column
OK	Push Button
⊙	Radio Button
☑	Check Box
⊞	Option Button
▤	List Box
▭	Combo Box
🖼	Picture
▥	Horizontal Scroll Bar
▮	Vertical Scroll Bar
📅	Date Picker
📆	Date Time Picker
▭	Tab Bar
🐛	Custom Control
✗	
📄	Active X
	Report

Displaying and using the Controls toolbar

To display the Controls toolbar at any time, press Alt+4 or select Controls from the Tools menu.

1. Click the icons for a data field, push button, picture, or other object.

2. Click the class type, Standard or Quick, from the class list box in the bottom part of the toolbar.

3. Click on the form window and drop the object onto the form.

Tip: When you click a control on the toolbar, you do not have to hold down the mouse button while moving to the form to drop the object.

Attribute Inspector

When you first create applications in SQLWindows, it is similar to laying out shapes on a canvas. The shapes are the objects that you use. You use the Attribute Inspector to modify an object name, title, and background color, and so on.

Each object has an **Edit** menu.

Attribute Inspector — Dialog Box: dlgLogin

Object Title	Database login
Allow Dock to Parent	
Docking Orientation	
Accessories Enabled	No
Tool Bar Visible	
Tool Bar Size	
Tool Bar Position	
Docking Tool Bar	
Tool Bar Docking Orientati	
Status Bar Visible	
Display Style	Default
Resizable	No
Type of Dialog	Modal
Absolute Screen Location	Yes
Left	3.675"
Top	1.5"
Width	3.9"

Undo All

Both the Edit menu and the Attribute Inspector are tailored for the ways in which you use an object. The Attribute Inspector shown above is for a dialog box.

To display the Edit menu, drop an object on a form. Right-click on the object. The Edit menu appears.

To display the Attribute Inspector, select Attribute Inspector from the Tools menu.

You can set attributes for your Form window and any graphical objects you drop on your form window. The Attribute Inspector gives you complete control over the look and feel of graphical objects.

Coding Assistant

SQLWindows Coding Assistant makes coding applications quick, easy, and far less error prone. Click on any section of the outline and appropriate code choices appear in the Coding Assistant.
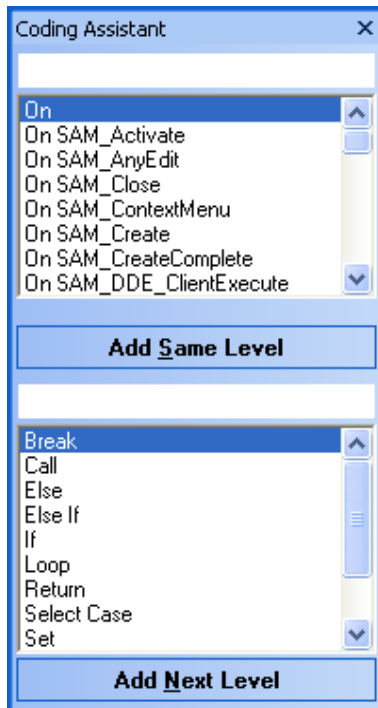
Bring up the Coding Assistant by selecting Coding Assistant from the Tools menu.

Double-click on an item in the Coding Assistant to add it to the outline. If you choose a function, the full syntax appears in the outline and you can quickly replace the parameters.

The Coding Assistant suggests whether to add new code to the same level or indented another level.

Online help

You can get the help at any time by pressing F1.

Component wizards

Programmers can get started quickly by using the new Component Wizards. SQLWindows comes with a number of wizards for easy application building.
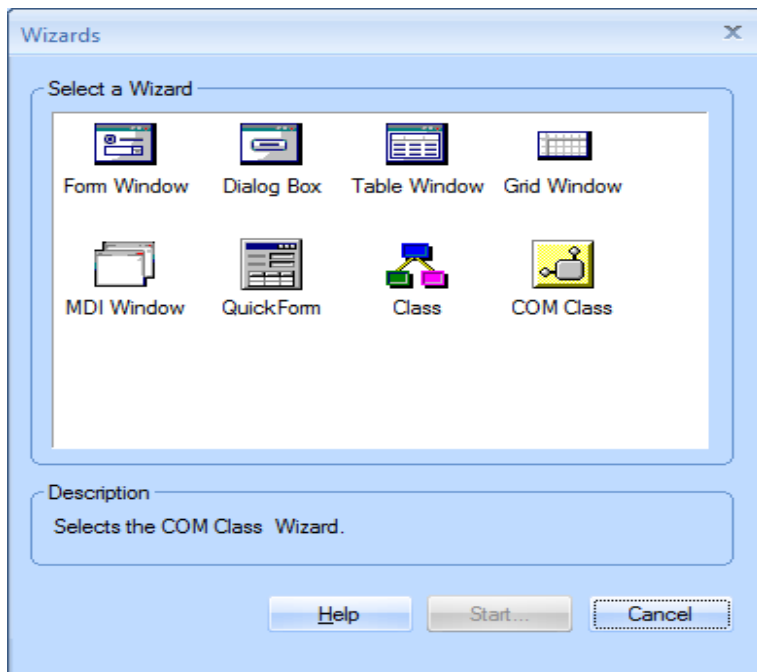
Report Builder

You can use Report Builder (often referred to as Reports) to design reports in an easy to use graphical environment. This chapter describes how you can use Reports to:

- Design and print various types of reports to meet your reporting needs.
- Format each report element with easily accessible formatting options.
- Display summary calculations to neatly arrange your report.
- Import and display data and graphics from different sources.
- Design a report template you can use repeatedly with different data sources.

To bring up the Report Builder, select Report Builder from the Tools menu in SQLWindows.

Database Explorer

To help developers work with all database schemas, whether simple or complex, SQLWindows includes easy to use schema management functionality.

Database Explorer allows you to browse, create or modify database object like tables, views and indexes. You can also design and format reports.

Because it is based on an intuitive "explorer" metaphor, it makes navigation of database structure very easy. It also includes full data preview capabilities for quickly reviewing database content. The Database Explorer works with all popular data sources in the same easy and consistent manner. You can write stored procedures, queries, edit data and execute SQL scripts, all from one utility.

To bring up the Database Explorer, select Tools, Database Explorer in SQLWindows.
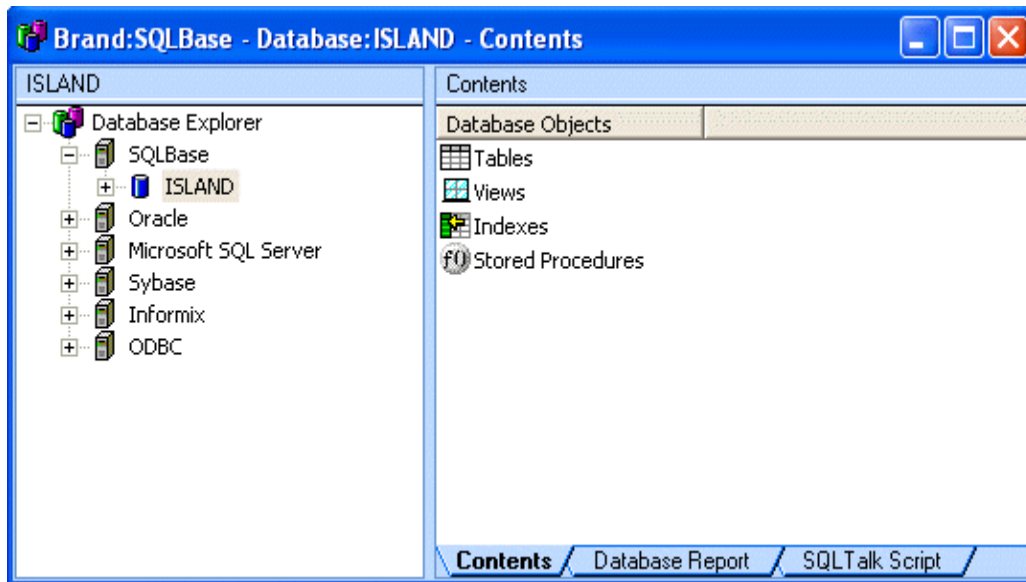
To manage stored procedures in SQLBase and Oracle databases, Team Developer's multi-database stored procedure manager offers some unique functionality. It allows you to browse, edit, create, compile and debug stored procedures in popular databases using a consistent and attractive user interface. After the stored procedure has been fully developed, Team Developer offers a highly productive way to integrate it into applications. It includes a Stored Procedure Class Wizard for stored procedure access which allow you to invoke stored procedures and display results without writing any code.

For example, a push button can be created to automatically fire-off a stored procedure, and the data set retrieved can be displayed in a table window graphical object. This unique capability works in the same easy and consistent way across popular databases, and is a huge productivity enhancing factor. Now even novice developers can integrate stored procedures into their Team Developer applications with just a few mouse clicks.
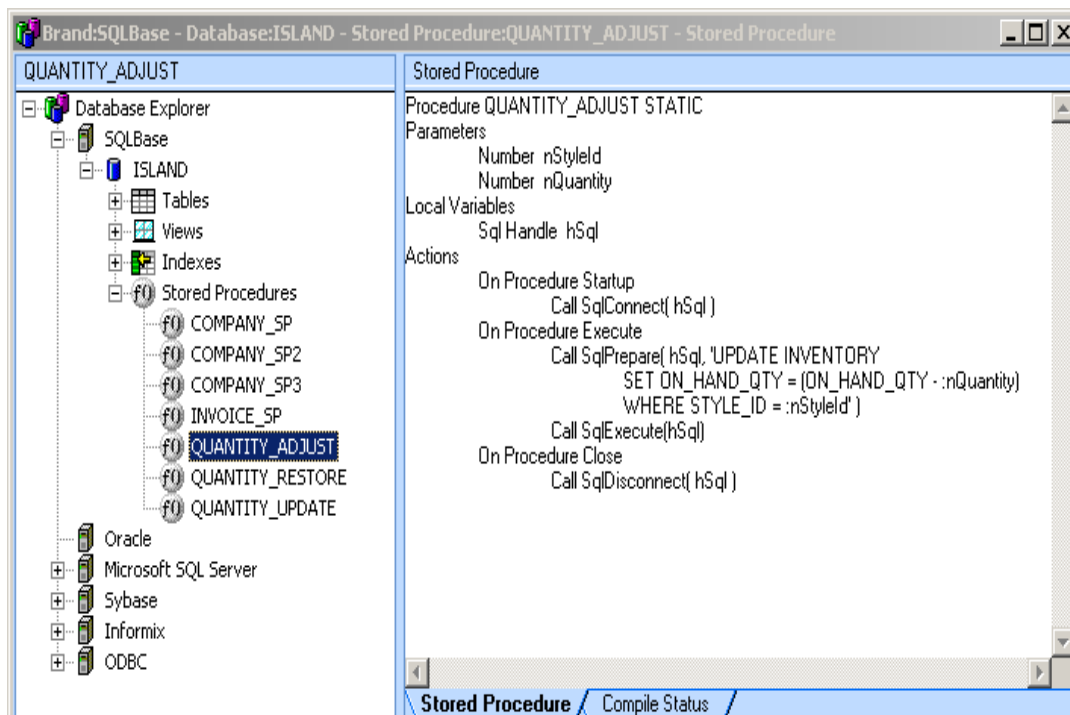
Visual Toolchest Class Library

The Visual Toolchest is a class library (APL file) that links into your SQLWindows applications to provide more than 250 functions and 17 classes for:

- Windows controls

- Powerful data manipulation

- Table-tree database display

- Font and color control

- Creating customized Windows controls

- Extended keyboard accelerator support

- Expanded file management

- and much more...

The Visual Toolchest class library includes Windows Explorer, calendar, color palette, and split window controls. Programmers can use these rich visual controls for building intuitive applications, some specially designed to emulate the Windows look and feel.

Windows Explorer Controls

The Explorer Tree control, cExplorerTree, is a list box control which emulates the Explorer tree in Windows. The Split Window control, cSplitterWindow, is a form window with a resizeable control which provides a narrow horizontal or vertical bar to allow two child controls to share the form window's client region. When used together, they provide convenient emulation of the Windows Explorer application.

Calendar Controls

The cCalendar and cCalendarDropDown date controls display one month at a time. User can scroll a day, week, month or year at a time. Keyboard shortcuts are provided for all mouse actions. cCalendarDropDown combines an edit field with a popup calendar to provide date selection capabilities. The programmer has full control over keyboard, mouse and display configuration.

Color Palette Controls

These are similar to the Windows color palette in look and feel. The cColorPalette and cColorPaletteDropDown color controls provide a fixed palette of colors, a menu region and a most recently used (MRU) color region. cColorPaletteDropDown combines a color sample with a popup palette to provide PowerPoint-like color selection capabilities. Programmers have full control over main color palette values, menu entries and the MRU region.

Dynalibs: Dynamic Linked Objects

A large application may eventually consist of several hundred windows. An increasingly common way to design such an application is to have the entire application "live" within a single executable, say, within a

single MDI frame. This simplifies the problems associated with frequently switching from one program to another.

You can build objects using SQLWindows, then compile them into object libraries, called Dynalibs, that may be dynamically loaded by your application at execution time. Dynalibs allow you to manage such an application by placing parts of a large system into independent dynamic run-time libraries. You can assign parts of a large project to separate developers and bring them together at run-time.

Applications can share pre-compiled components, resulting in faster development because of greatly reduced compilation. Application distribution is easy because certain components of a SQLWindows application could be replaced without requiring the entire application to be recompiled and rebuilt. Providing updates to your applications can be as simple as distributing a Dynalib.

Dynalibs provide the ability to segment the application logic into pre-compiled libraries that are linked into the application at run-time. Dynalibs themselves can use all features of SQLWindows, including the Object Compiler.

When building a Dynalib, the programmer can specify which objects are "exported" from the uncompiled library (.APL file) to the Dynalib (.APD file). This is called export control, and it allows you to define the public/private members of a Dynalib. The objects that can be exported to a Dynalib are:

- Global functions

- Global variables

- Window functions

- Top-level window objects - forms, tables, dialogs, MDI windows, and MDI children.

Team Object Manager

Team Developer includes the industry's most comprehensive and powerful team and object management environment called Team Object Manager, which is based on the shared Team Object Repository. It allows you to manage large projects with dozens of developers, hundreds of end-users, and thousands of objects.

Note: Start using Team Object Manager by working through the tutorial in chapter 5.

Unlike other development environments that claim team programming support but only support external version control through third-party tools like Intersolv's PVCS, Team Developer not only has built-in version control support, but goes far beyond it to include features like project branching, diff and merge, coding standards management, deployment management, impact analysis, audit trails, management reporting and much more. For openness, it also supports PVCS. Thus, it is a truly comprehensive project and object management environment, and is a project manager's dream come true.

Project and configuration management is easy with Team Object Manager, even when you are working with multiple versions. The project branching feature allows you to work on multiple versions concurrently. For example, after developing version 1 of an application, you might want to simultaneously work on version 1.1 and version 2.
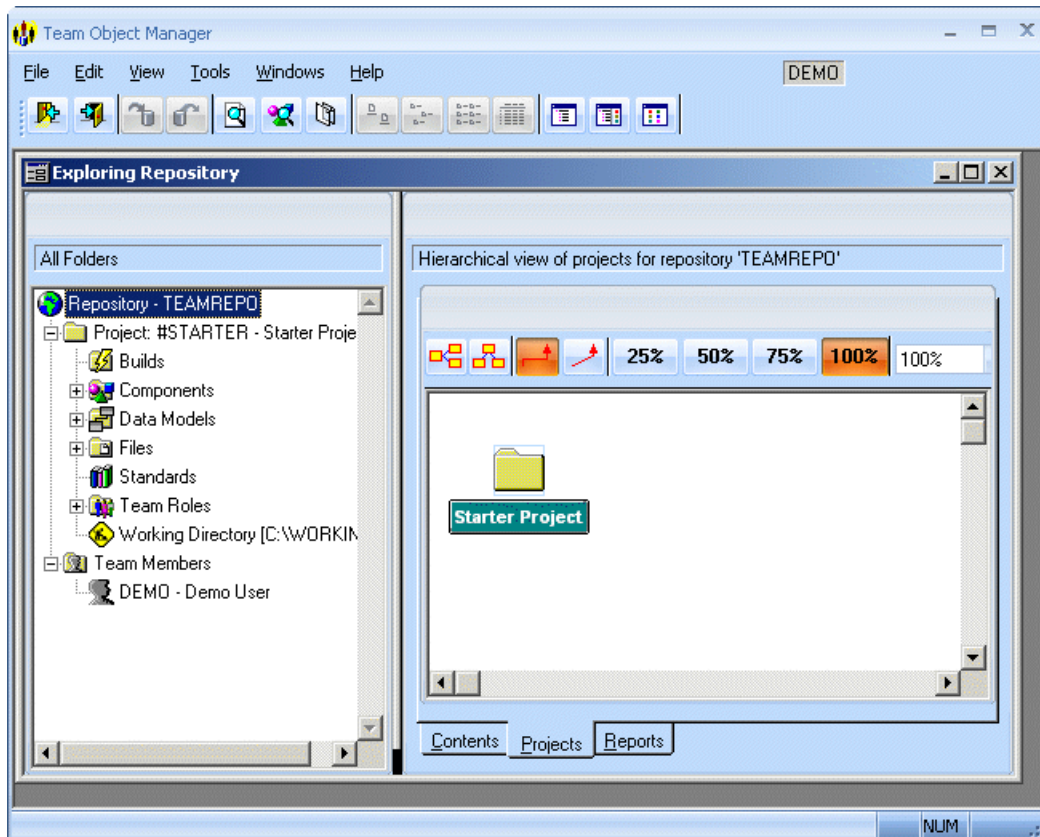
With Team Object Manager you can visually branch your applications. Sophisticated version control at the file and object levels ensures that all your team members can securely coordinate their work, no matter how large the project. Since the typical application development team includes members with different roles, Team Developer allows you to assign and manage roles and privileges for various team members.

Team Object Repository

Team Developer includes a scalable architecture which is repository-driven and component-based. Teams of developers can collaborate to develop large-scale applications that can be deployed to hundreds of end-users in a 2-tier or 3-tier architecture.

All this is possible because Team Object Manager has a shared, object-oriented repository called the Team Object Repository. The Repository can be resident on many leading relational DBMSs, such as SQLBase, Microsoft SQL Server, Oracle, and Sybase System 11.

Visual Diff and Merge

Teams of programmers have a powerful, reliable tool for managing changes made to applications by multiple programmers. Team Object Manager has an advanced version management feature commonly known as Diff and Merge. Commonly, in other version control systems like PVCS, a difference report shows the differences between two revisions of the same file. For example, you can use a difference report to:

- Identify specific differences between revisions of the same file.

- Determine what changes have been made to a file by other programmers before checking it in.

- Determine which changes to keep or remove when checking in a file that has been changed by other programmers since you checked it out.

The Diff and Merge utility goes far beyond difference reports in these ways:

- Graphical comparison of two files to determine where they differ. The Diff tool shows a list of the differences in a very visual manner. A sample of this work is shown below, demonstrating the way that the differences between the two files are displayed side-by-side.

- Creation of reverse delta files to store the changes between file revisions. This means that, for example, an original file and a revision of that file can be stored as one base file, and a list of the changes that would be necessary to convert the stored file to the previous version. Additional revisions would mean only additional change lists need be stored, rather than files containing the complete files. This is very space efficient, and thus allows Team Object Manager to store an unlimited number of revisions for future recovery.

- Two- and three-way merging of files. Team Object Manager can perform two way merges, combining a base file and a revised file to produce a unified project version. Three-way merges work in the same way, combining a base file and two modified files to produce a file containing both sets of changes.

  If conflicts are detected during this process the conflicting code (objects) from all files is added so that the conflicts can be resolved manually.
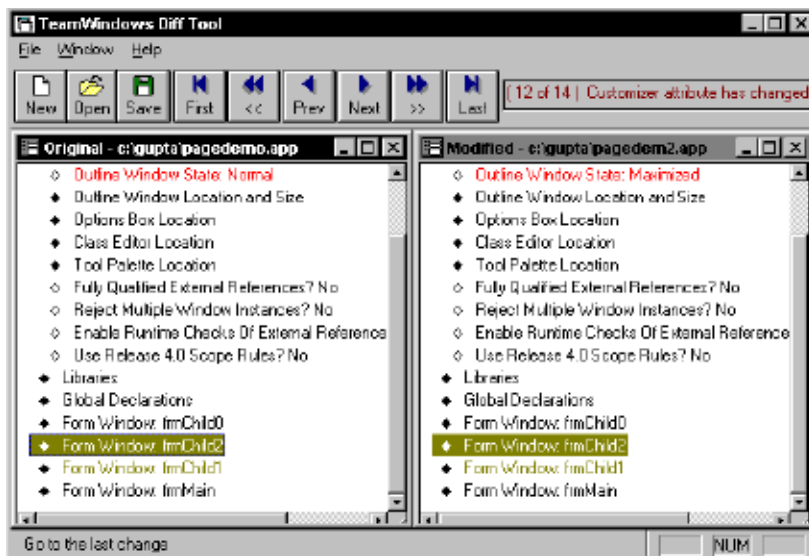
Object Browser

Teams of programmers have a powerful, reliable tool for documenting and understanding object-oriented code. The Object Browser in Team Object Manager allows complex OOP classes to viewed and printed graphically. A number of standard reports provide details of class behavior.
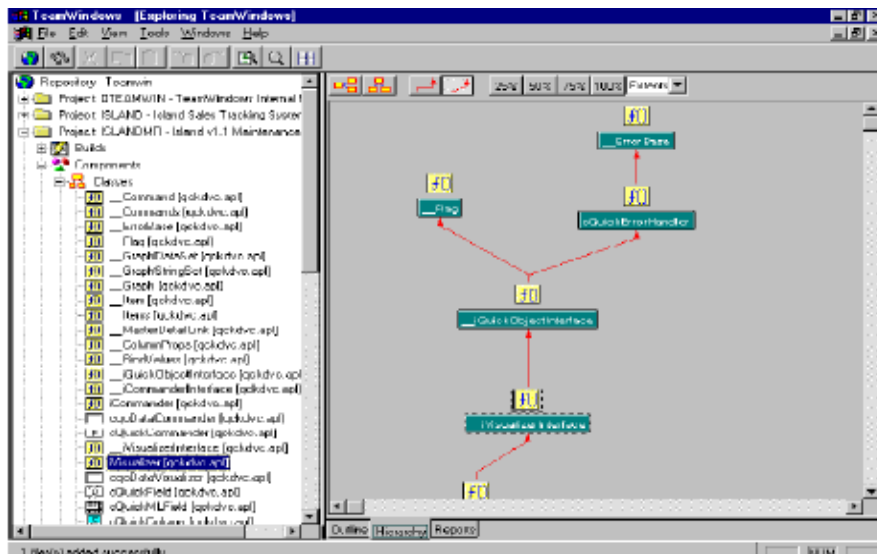
Data Model Viewer

The Team Object Repository stores all the information about the structure of your project databases: the names of the tables (entities) and columns (attributes), the primary and foreign key relationships, and information about data types, display formats, default values and validation criteria.

Data models are formalized ways of displaying database structure information. The Data Model Viewer is built around the concept of data models as objects. Data Models are treated just like any other file in the repository in that they can be branched, shared, or checked in and out, and any number of revisions can be kept. Multiple data models can be maintained for each project.

In addition to modeling the normal database information (tables, columns, relationships etc.) the Data Model Tool is designed to include properties that are used by QuickObjects. Extensive facilities are provided to attach these properties to your database information so that when you use a QuickObject in your application its property information can be supplied from the model.

The Data Model Viewer offers a powerful graphical interface to the repository, called the Entity Relationship diagram. The Entity Relationship diagram allows you to graphically view the database model in order to improve clarity. It does not, however, allow you to change the database schema.

QuickObjects

QuickObjects are software classes with a design time interface. QuickObjects provide you with an easy point-and-click alternative to writing code when you create client/ server applications. The purpose of QuickObjects is to bring ease-of-use, re-usability, and flexibility to the development process.

The QuickObject architecture provides a robust foundation on which more experienced developers can make their existing classes easier to use without sacrificing power and functionality. You can use an instance of a DVC QuickObject (data source, visualizer, and commander). In addition to these DVC QuickObjects, Gupta provides many other QuickObjects, such as Quick OLE, QuickEmail, and QuickReport.

Read the manual, Using and Extending QuickObjects for a detailed description of QuickObjects, including a lengthy discussion of the DVC framework

Benefits to using QuickObjects include the following:

- You can use QuickObjects right away to build your applications, as explained in Chapter 1.

- QuickObjects are based on object-oriented concepts and are fully extensible. For example, you can take a QuickObjects class such as a Fetch All Records push button and derive your own specialized Fetch Records for customers who pay by credit card.

Create Your Own QuickObjects

Creating your own QuickObjects is done via standard SAL programming. You can create a new QuickObject class and have it inherit behavior from one or more existing objects or QuickObject classes, or create a class that is entirely new.

In this way, you can expand your choice of QuickObjects and create objects that have re-usable behavior tailored to your specific needs or vision, to be used either in your own organization or marketed to larger audiences of developers.

One of the easiest ways to take advantage of QuickObjects is through the QuickForm wizard.
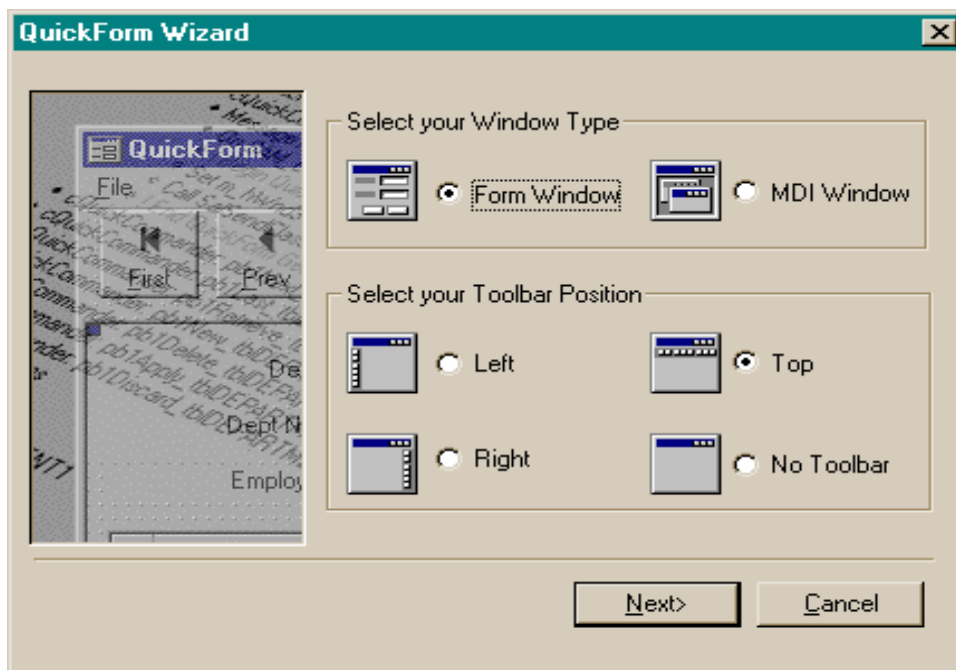
Programmers want to either extend applications by creating their own QuickObjects or modifying the existing QuickObjects. Read the ***Using and Extending QuickObjects*** book for information on QuickObject concepts, the QuickObject Framework, QuickObject libraries for OLE2, and eMail.

Web Access QuickObjects

SQLWindows includes Web Access QuickObjects designed specifically for use in applications that work over any network supporting TCP/IP, HTTP, and FTP. These QuickObjects let you develop applications for the Internet without previous knowledge of the various low-level Winsock APIs and protocols. Team Developer's Web Access QuickObjects make it easy to create applications that integrate Internet data with traditional relational data.

There are three Web Access QuickObjects: cQuickHTML, cQuickHTTP, and cQuickFTP.

- cQuickHTML

HTML is used to create World-Wide Web documents that are subsequently transmitted using HTTP. The HTML control parses and displays HTML

QuickEmail

pages. The HTML QuickObject lets you integrate Internet Web browser capabilities directly into your applications. You can drop the control onto a form or dialog.

- cQuickHTTP

  HTTP is the language of the World Wide Web. An HTTP request goes to a server, which then replies with an HTML page. The HTTP QuickObject lets you retrieve HTML page contents without displaying the pages. Retrieval time is significantly faster because you are not retrieving and displaying graphics and non-textual data. You can then parse links from a page and extract the textual data without the HTML tags.

- cQuickFTP

  The FTP QuickObject lets you perform file operations on a remote FTP server. FTP Servers are similar to LAN file servers in that they supply files to client machines that request the files. Using this control, you can write applications that can list files and directories on servers, get and put files on servers, and read and write files on servers.

SQLWindows includes eMail connectivity to popular eMail systems. Previously, the process of mail-enabling an application was long and tedious, often requiring programmers to write DLLs and learn the intricacies of MAPI. With QuickEmail, you can mail-enable an application in the same point-and-click technique used with other QuickObjects.

QuickReport

A QuickReport makes it easy for programmers to build and use reports within the SQLWindows environment. WYSIWYG reporting within SQLWindows and reuse of templates improves productivity for report designers. Report designers can create generic report templates that can be made available to other programmers who need to tie these reports to various types of data. For example, a report designer may design across tabular report template that conforms to corporate standards. This report template is, in turn, used by programmers in several departments:

- In the accounting department, the report would be tied to Product and Revenue tables to create a report on product revenue by financial quarter and by product.

- In the manufacturing department, the report would be tied to Part and Inventory tables to create a report on part inventory by month and by warehouse.

QuickReport allow you to create new report templates by integrating with the Report Builder. QuickReports make it easier to reuse templates, populate report templates with data, and integrate reports seamlessly with other objects on an application form. The process of using QuickReports to integrate a report into an application is:

- Report designer builds a report using the Report Builder.

- Report designer builds a QuickReport template from the report, and makes these available to other programmers. Typical report templates may be horizontal in nature, such as Mailing Label reports; or they may be vertical in nature, such as Department Salary reports.

- Report assemblers use QuickReports within SQLWindows to build reports- in-forms from these templates. This involves associating data, in the form of QuickObject data sources, with QuickReport templates.

This control lets you build tabbed forms and dialogs. QuickTabs enable the creation of property sheets similar to the style in Windows. The default style is a top tab orientation. We will also support a bottom tab orientation. The bottom tab style is a small tab similar to how Excel displays multiple sheets in a Workbook. With this control, you can place a form window on a tab, so that the Window is created and destroyed when the tab is shown/hidden.

QuickTabs

This control lets you build tabbed forms and dialogs. QuickTabs enable the creation of property sheets similar to the style in Windows. The default style is a top tab orientation. We will also support a bottom tab orientation. The bottom tab style is a small tab similar to how Excel displays multiple sheets in a Workbook. With this control, you can place a form window on a tab, so that the Window is created and destroyed when the tab is shown/hidden.

QuickGraph

QuickGraph integrates business graphics into SQLWindows applications through graph controls that are directly linked to data. This elegant feature brings drag-and- drop simplicity to the creation of charts and widens the choice of display styles for your data. Stunning graphics can be combined within your application without the need to resort to coding.

SQLBase Database Engine

Team Developer includes a SQLBase database engine for Window.. This 32-bit single-user database engine combines the power and programmability of a native relational database server with the ease and unobtrusiveness expected of a graphical PC application.

The SQLBase single-user database engine is built for easy deployment, including easy installation, a tool for easy set-up of databases, and interfaces for SQLWindows, C, C++ and ODBC. Its programmability options, such as advanced cursor handling and result set processing, allow more robust applications to be deployed off the LAN and onto the desktop. More robust applications can also run against larger data sets, taking advantage of SQLBase's native relational functionality, such as query and sort optimization.

The SQLBase database engine included in Team Developer is intended for development only. When you deploy your application, you may purchase a deployment suite called SQLBase Desktop, or SQLBase Server, a workgroup database server. SQLBase Server, the fastest and easiest server for Linux and Windows, is a robust database server designed for running workgroup applications. It provides the functionality you expect in a high-performance database server, including triggers and stored procedures, multiple concurrency options, and query and sort optimization. SQLBase Server delivers this power and performance with the unobtrusiveness expected of PC applications, including a very small footprint. Its tools and maintenance options facilitate the easy set up your databases for speed, remote management, and the automation of routine tasks.

Database Utilities

SQLBase Desktop comes with a comprehensive set of graphical tools to manage and administer your desktop database environment.

- SQLConsole provides point-and-click administration of databases, including the automation of routine tasks, such as database back-up.

- SQLTalk is an interactive SQL utility that helps programmers test and debug connectivity to all relational databases.

Native Connectivity to SQL Databases

The product line gives you high-performance native connectivity to popular enterprise databases, so you can make the best use of your data. Team Developer includes high-performance native SQLRouters for Informix, Ingres, Oracle, SQLBase, and Sybase that fully support all features of each database. These native routers yield higher performance than standards-based connectivity solutions. It also includes ODBC connectivity in those cases where the database vendor recommends ODBC as the native API. For example, Microsoft has implemented SQL Server's native API using ODBC.

Therefore, ODBC is the preferred way to connect to SQL Server. But in most other cases, the database vendor recommends the use of a native API like OCI for Oracle, and CT-LIB for Sybase, rather than ODBC. In such cases, the native API is directly and fully supported. Unlike other multi-database solutions that take a 'lowest common denominator' approach, Team Developer connectivity supports the 'highest common denominator' because each database link is engineered as a separate 'router'. The specific targeting of each database with custom links ensures maximum integration. Access to mainframe databases like DB2, and legacy data sources like VSAM, is directly supported through a companion product, SQLHost.

ODBC Connectivity to SQL Databases

Connectivity to other ODBC compliant databases such as Microsoft Access, IBM DB2/400, and others, is supported via third-party ODBC drivers from Microsoft, Starware, and others, respectively. Since most ODBC drivers follow different rules for configuration, Team Developer provides one consistent way to describe any ODBC data source to an application. You can use the Database Connectivity Configuration (DCC) feature to configure any ODBC data source for use with SQLWindows.

SQLTalk

SQLTalk is an interactive user interface that allows you to use SQL database commands. SQLTalk can be used as an interface for other databases besides SQLBase, such as DB2.

SQLConsole

Use SQLConsole to simplify database administration. Use it to manage database objects, monitor performance, and automate database maintenance.

Chapter 3 – Using ActiveX Objects

With Team Developer ActiveX support, you can embed objects from other applications in your applications. This chapter shows you how to build an application that displays database budget information in both table and graph form, linked to Microsoft Calendar Control 8.0 to display the month.

This chapter describes how to use Team Developer QuickObjects to do the following:

- Use QuickObjects to save yourself coding time by using a pre-defined database login dialog.

- Link and embed Microsoft Calendar Control 8.0 in your application.

- Create a QuickObject graph linked to the calendar and table to display a pie chart.

To add the Microsoft Calendar Control 8.0, it must be installed on your machine. It was installed if you installed Microsoft Access during a Microsoft Office 97 install, selected Options, and checked Calendar Control.

Look at the Finished Application

Before you begin, take a look at the application you are going to build.

1. In SQLWindows, select **Open** from the File menu.

2. In the \Samples directory, select **Budgets.app**.

3. When it is open, select **Go** from the **Debug** menu.

   A database login dialog appears.
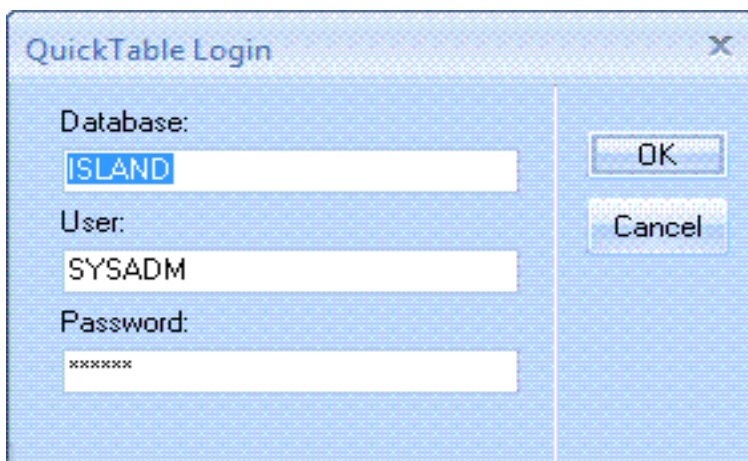
4. Enter **ISLAND**, **SYSADM**, and **sysadm**, respectively, in the fields and click **OK**.

   The Department Budgets window opens.

Department	January	February	March	Apri	May	June	July	August	S
CSS	10000	10000	10000	10000	10000	10000	10000	10000	1
FIN	10770	19570	21570	21570	21570	19570	24570	22570	7
MKTG	32000	36000	37000	34000	32000	34000	37000	36000	3
MFG	50000	39000	41500	40500	40000	31500	29000	45000	3
PCI	15000	12000	15750	14000	12000	11500	11500	11500	1
HR	28000	19000	28000	28000	29000	13000	19000	19000	1
MIS	30000	25000	75000	25000	25000	17500	17500	35000	4

Use the push buttons in the toolbar to navigate through the budget records in the database.

Prepare the Login

This section describes how to add QuickObject functionality. The application in Chapter 1 – Using QLWindows explained how to program a Login dialog box from scratch. The following exercise explains how to use a pre-made Login dialog box, a QuickObject. This saves time in coding, but is not editable, so you get to use it as is.

1. Select **New** from the **File** menu to begin with a new, blank application template.

2. Click the **Outline** tab in the right pane.

   In the outline, click **Libraries**.

   In the lower part of the Coding Assistant, double-click **Database QuickObjects**.

   This automatically adds Dialog Box: dlgQOLogin to the outline. This text is *not* editable.

3. In the Coding Assistant, double-click **File Include…**

This brings up the Open file dialog box.

Find **qckttip.apl** in the Samples directory and click **Open**.

Press **Enter**.

Add qckttip.apl to enable tooltips in your application. This lets you define tooltip help for the push button in your application. In addition, another library, called ttmngr.apl, is automatically added to the Libraries outline section.

4. In the Coding Assistant, double-click **File Include…**

   Find **table.apl** in the Samples directory and click **Open**.

   This file provides the class support for the table window and links the Calendar Control, Table, and Graph together. For a step-by-step description of how to create the table.apl library, see Chapter 3 – Object-Oriented Programming.

   Your outline should look like the following:

```
Libraries
   File Include: qckdvc.apl
   File Include: qckttip.apl
   File Include: table.apl
   File Include: Ttmngr.apl
Global Declaration
Dialog Box: dlgQOLogin
```

Create the Form Window

To create a form for the application, follow these steps:

1. Right-click the Windows section in the left pane (called the Tree view).

   Select **New, Form Window**.

   Type **frm1**

2. Select **Attribute Inspector** from the Tools menu.

   Type **Department Budgets** for the Object Name.

   Change Enable Accessories to **Yes**.

   This enables the toolbar and the status bar on your form.

   In layout view, your form should look like the following:
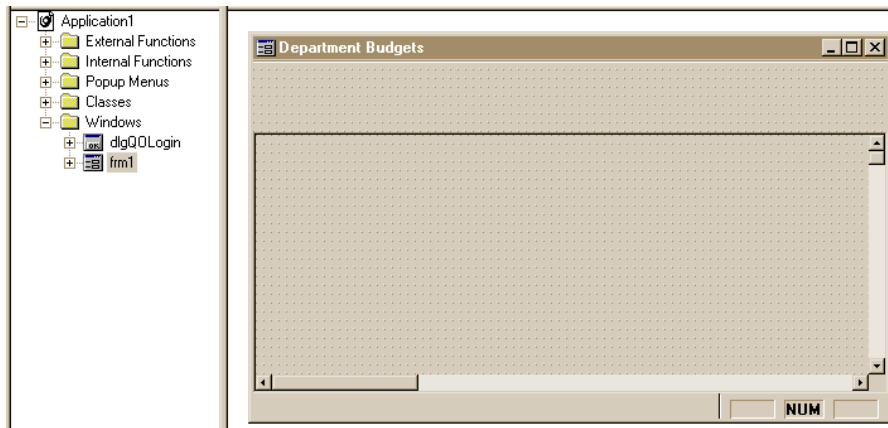
Add a Calendar Control, Table, and Graph

To add the Microsoft Calendar Control 8.0, you must have it installed on your machine. You have it if you installed Microsoft Access during a Microsoft Office 97 install, and selected Options, and checked Calendar Control.

1. Select **Controls** from the Tools menu.

2. Click the ActiveX object (icon has a red X) in the Controls toolbar.

    In the lower part of the Controls toolbar, a list of currently registered ActiveX controls appears.

3. Select **Calendar Control 8.0** and drop this object in the upper left corner of the form.

    Resize it so that you can see the calendar.

    When you drop the Calendar Control object on the form, the following libraries are automatically inserted into the outline:

    ▸ File Include: C:\GUPTA\AXLibs\ Microsoft Calendar Control 8.0.apl
    ▸ File Include: Automation.apl
    ▸ File Include: OLE Automation.apl
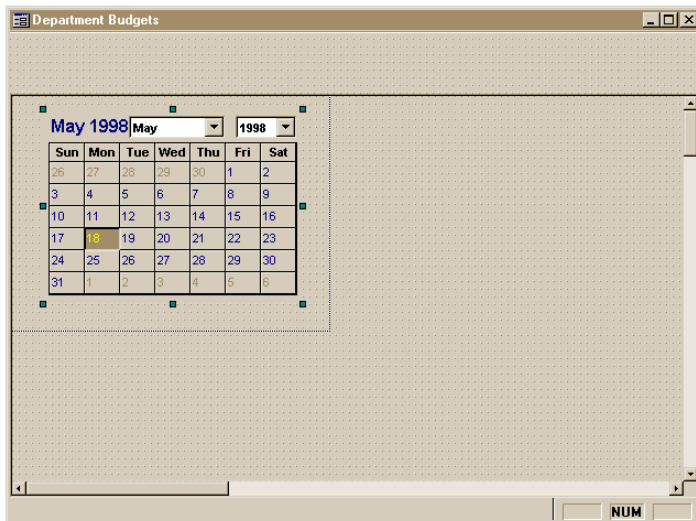
4. Right-click the Calendar Control and select **Control Properties**.

   In the General tab, on the left side of the Properties dialog, select **Short** for Day Length and Month Length.

   Set Grid Cell Effect to **Flat**.

   On the right side of the Properties dialog, there is a group called **Show**. Remove all of the checks from the check boxes except for the **Month/Year Title**.

   The Properties dialog should look like the following:
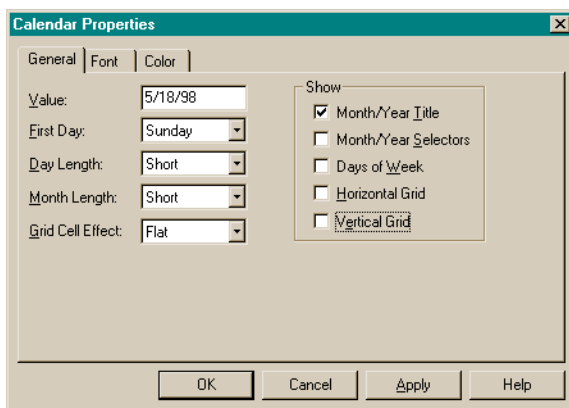
   Click **OK**.

5. Select the Child Table Window object in the Controls toolbar.

   In the lower part of the Controls toolbar, select **BudgetTable**. Drop this on the form under the Calendar Control.

   Your form should look like the following:

57
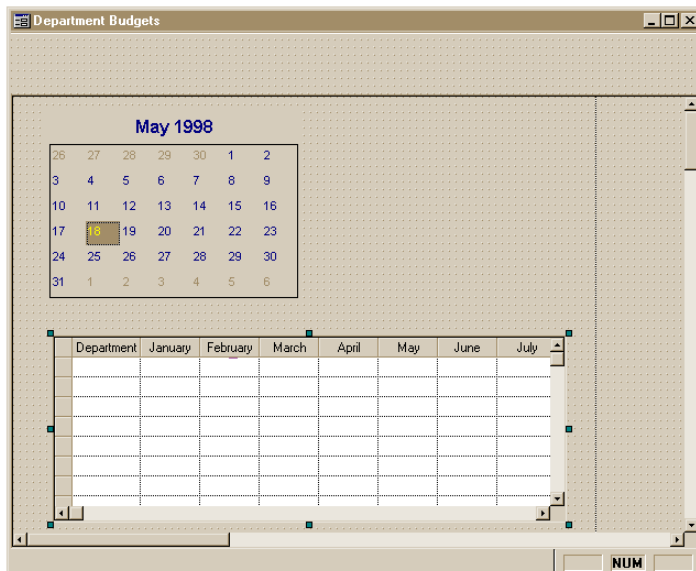
6. Select the cQuickGraph object in the Controls toolbar. Drop this to the right of the Calendar Control, and resize the cQuickGraph so that you can see the entire graph.

7. Right-click the cQuickGraph object.

   Select cQuickGraph Properties.

   On the **2D Gallery** tab, select the **Pie** graph.

8. Select the **Titles** tab.

   Enter Monthly Budget by Department for the Graph Title.

9. Select the **Design** tab.

   In the Toolbar group, uncheck the Run Time and Design Time checkboxes.

10. Select the **QuickGraph** tab.

    Remove the check mark from the None checkbox in the **Values to Graph** (Y- axis).

11. Click **OK**.

    Your form should look like the following:
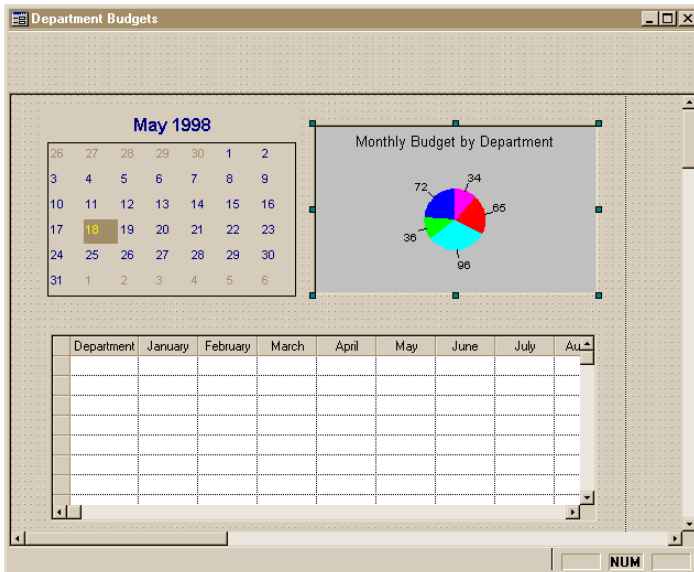
Drop the Visual Controls

1. Select the Push Button object in the Controls toolbar.

   In the lower part of the Controls toolbar, select the **QuickToolTipPushButton**. Drop this on the left side of toolbar.

   In the Attribute Inspector, change the name from **pb1** to **pbFirst**.

   Locate **File Name**, and click the ellipses (**...**).

2. In the Open dialog box, select **toprec.bmp** (usually in the C:\GUPTA directory).

   Click **Open**. An arrow icon appears on the push button.

3. Right-click the push button.

   Select **Tooltip...** This brings up the Tooltip QuickObject Properties box.

   Enter **First** in the ToolTip Text datafield.
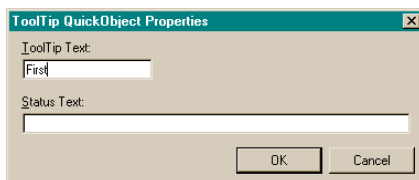
   Click **OK**.

4. Repeat the previous three steps three more times to add the following controls:
   - ▶ Previous (pbPrev, prevrec.bmp, Previous)
   - ▶ Next (pbNext, nextrec.bmp, Next)

‣ Last (pbLast, lastrec.bmp, Last)

Note: Resize the push buttons so you can see the arrow bitmaps.

Your form should look like the following:
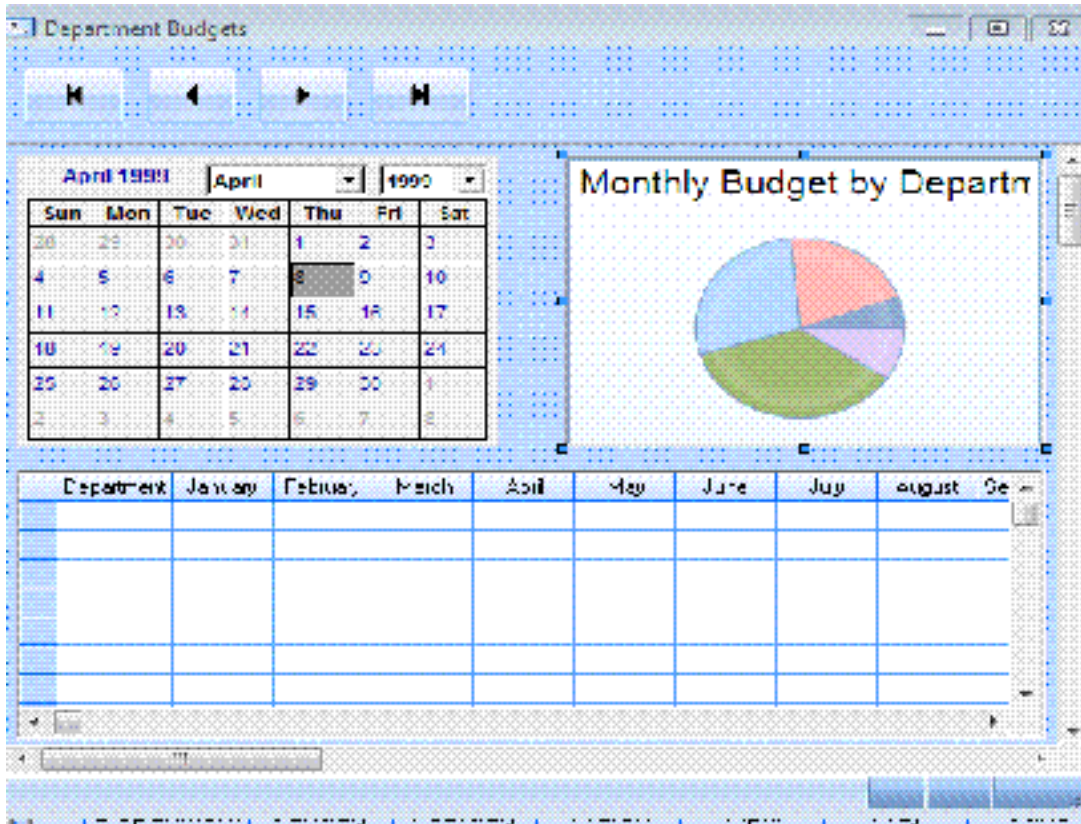
Code the Application

Call the Login Dialog

When your applications starts, you want to call the database dialog and pass three parameters to it. Follow these steps:

1. Double-click **Global Declarations**.

   Click **Variables**.

   Double-click **String** in the Coding Assistant and, in the outline, after String, type **sDatabase**

   Double-click **String** in the Coding Assistant and type **sUser**

   Double-click **String** in the Coding Assistant and type **sPassword**

2. Click **Application Actions** in the Global Declarations section.

   Double-click **On SAM_AppStartup** in the coding assistant.

   Double-click **Call** in the lower part of the Coding Assistant.

   Double-click SalModalDialog in the Coding Assistant to add it to your outline.

Change **Template, Window_Handle** to **dlgQOLogin, hWndNULL, sDatabase, sUser, sPassword**

This calls the Login QuickObject.

Define the Form Variables

3. In the outline, click **Form Window: frm1**.

    Click **Window Variables**.

    Double-click **Number** in the Coding Assistant and type **nMonth**

    Click **Window Variables**.

    Double-click **MSACAL_ICalendar** in the Coding Assistant and type **Calendar**

    Your code should look like the following:

```
Window Variables
   Number: nMonth
   MSACAL_ICalendar: Calendar
```

Add Functions to Create the Form

4. Click **Message Actions**.

    In the Coding Assistant, double-click **On SAM_Create**.

    Double-click **Call** in the Coding Assistant.

    In the list box in the Coding Assistant, select **SAL + User Functions**.

    Find and double-click **SalActiveXGetObject** to add it to the outline.

    Replace **Window_Handle, ???** with **ax1, Calendar**

5. Click **On SAM_Create**. This tells SQLWindows where in the outline to insert the next call.

    Double-click **On SAM_CreateComplete** in the upper part of the Coding Assistant.

    Double-click **Call** in the lower part of the Coding Assistant.

    Select the **UDV Functions** from the list box in the Coding Assistant.

    Double-click **Calendar.PropGetMonth** to add it to the outline.

    Change **Number** to **nMonth**

6. Double-click **Call** in the Coding Assistant.

    Select the **Window Functions** from the list box in the Coding Assistant.

    Double-click **tbl1.SelectColumn** to add it to the outline.

    Change **Number** to **nMonth+1**

    Your code should look like the following:

```
Message Actions
   On SAM_Create
      Call SalActiveXGetObject( ax1, Calendar )
   On SAM_CreateComplete
      Call Calendar.PropGetMonth( nMonth )
      Call tbl1.SelectColumn( nMonth + 1)
```

Define Contents for the Graph

To link the graph to the table window, follow these steps:

1. Double-click **Contents**.

    Double-click **cQuickGraph: cc1**.

    Click **Message Actions**.

    Double-click **On DRAWNOTIFY** in the Coding Assistant.

    Double-click **Set** in the lower part of the Coding Assistant and type the following:

    ```
    cc1.Graph.DataSource = 'tbl1'
    ```

    Set the graph to display information from the highlighted column in the table, and specify the number of data fields to graph.

2. Click **On DRAWNOTIFY** in the outline.

    Double-click **Set** in the Coding Assistant.

    In the outline, after Set, type the following:

    ```
    cc1.Graph.DataField[0] = 'col' || SalNumberToStrX( nMonth, 0 )
    ```

    This selects one numeric field from a data source.

3. Double-click **Set** in the Coding Assistant and type the following:

    ```
    cc1.Graph.NumDataFields = 1
    ```

4. Double-click Call in the Coding Assistant and type **Draw ( )**

    Your code should look like the following:

    ```
    Contents
       MSACAL_Calendar: ax1
       BudgetTable: tbl1
    cQuickGraph: cc1
       Message Actions
          On DRAWNOTIFY
             Set cc1.Graph.DataSource = 'tbl1'
             Set cc1.Graph.DataField[0] = 'col' || SalNumberToStrX( nMonth, 0 )
             Set cc1.Graph.NumDataFields = 1
             Call Draw( )
    ```

5. Goto **Form Window: frm1**.

    Double-click **Message Actions**.

    Click **On SAM_CreateComplete**.

    Double-click **Call** in the Coding Assistant.

    Select the **Sal + User Functions** from the list box in the Coding Assistant.

    Double-click **SalPostMsg** to add it to the outline.

    Replace **Window_Handle, Number, Number, Number** with **cc1, DRAWNOTIFY, 0, 0**

    This posts the information from the Calendar control to the graph so that the graph can be drawn.

    Your code should look like the following:

    ```
    Message Actions
       On SAM_Create
    ```

```
On SAM_CreateComplete
   Call Calendar.PropGetMonth( nMonth )
   Call tbl1.SelectColumn( nMonth + 1)
   Call SalPostMsg(cc1, DRAWNOTIFY, 0, 0 )
```

Add an ActiveX Control

To add an ActiveX control, follow these steps:

1. Click **Form Window: frm1**.

    Double-click **Contents**.

    Double-click **MSACAL_Calendar: ax1**.

    Click **Message Actions**.

    Double-click **On NewMonth** in the Coding Assistant.

Note: If you do not see NewMonth in the Coding Assistant, switch to Layout view, select the Calendar object on the form, and switch back to Outline view.

2. Click **Actions** under NewMonth.

    Double-click **Call** in the Coding Assistant.

    Select **UDV Functions** in the Coding Assistant list box.

    Double-click **Calendar.PropGetMonth** to add it to the outline.

    Change **Number** to **nMonth**

    Your code should look like the following:

```
MSACAL_Calendar: ax1
   Message Actions
      On NewMonth
         Paramters
         Actions
            Call Calendar.PropGetMonth (nMonth)
```

Code it so you can cycle through the table.

3. Double-click **Call** in the Coding Assistant.

    Select **Window Functions** in the Coding Assistant list box.

    Double-click **tbl1.SelectColumn** to add it to the outline.

    Change **Number** to **nMonth + 1**

    Post the information from the Calendar control to the graph, so that the graph can be redrawn to the appropriate month.

4. Double-click **Call** in the Coding Assistant.

    Select **Sal+User Functions** in the Coding Assistant list box.

    Double-click **SalPostMsg** to add it to the outline.

    Change **Window_Handle, Number, Number, Number** to **cc1, DRAWNOTIFY, 0, 0**

    Your code should look like the following:

```
MSACAL_Calendar: ax1
   Message Actions
      On NewMonth
         Paramters
         Actions
            Call Calendar.PropGetMonth (nMonth)
            Call tbl1.SelectColumn( nMonth + 1 )
            Call SalPostMsg( cc1, DRAWNOTIFY, 0, 0 )
```

Set the Actions for the Push Buttons

To set the actions for the First push button, follow these steps:

1. Go to **Form Window: frm1**.

   Double-click **Toolbar**.

   Double-click **Contents**.

   Double-click **QuickToolTipPushbutton: pbFirst**.

2. Click **Message Actions**.

   Select **On SAM_Click** in the Coding Assistant.

   Double-click **Call** in the lower part of the Coding Assistant.

   Select the **UDV Functions** in the Coding Assistant list box.

   Double-click **Calendar.SetMonth** to add it to the outline.

   Change **Number** to **1**

To set the actions for the Previous push button, follow these steps:

1. Double-click **QuickToolTipPushbutton: pbPrev**.

2. Click **Message Actions**.

   Select **On SAM_Click** in the Coding Assistant.

   Double-click **Call** in the lower part of the Coding Assistant.

   Double-click **Calendar.PreviousMonth** to add it to the outline.

   There are no parameters.

To set the actions for the Next push button, follow these steps:

1. Double-click **QuickToolTipPushbutton: pbNext**.

2. Click **Message Actions**.

   Select **On SAM_Click** in the Coding Assistant.

   Double-click **Call** in the lower part of the Coding Assistant.

   Double-click **Calendar.NextMonth** to add it to the outline.

   There are no parameters.

To set the actions for the Last push button, follow these steps:

1. Double-click **QuickToolTipPushbutton: pbLast**.

2. Click **Message Actions**.

   Select **On SAM_Click** in the Coding Assistant.

   Double-click **Call** in the lower part of the Coding Assistant.

   Double-click **Calendar.PropSetMonth** to add it to the outline.

   Change **Number** to **12**

Run the Application

To compile the application, follow these steps:

1. Select **Go** from the **Debug** menu.

   When SQLWindows asks if you want to save changes, click **Yes**.

2. Save this application as **Budgets2.app**.
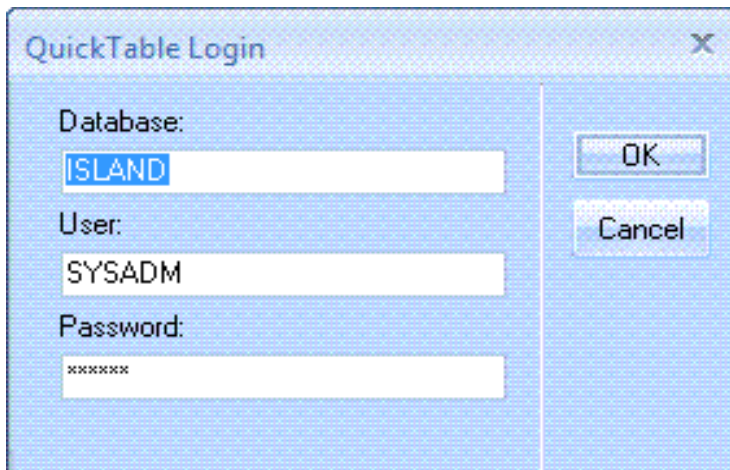
3. When the Login dialog appears, enter **ISLAND**, **SYSADM**, and **sysadm**, respectively, in the data fields.

4. Click **OK** to bring up the form window, Department Budgets.

   The form should look like the following:
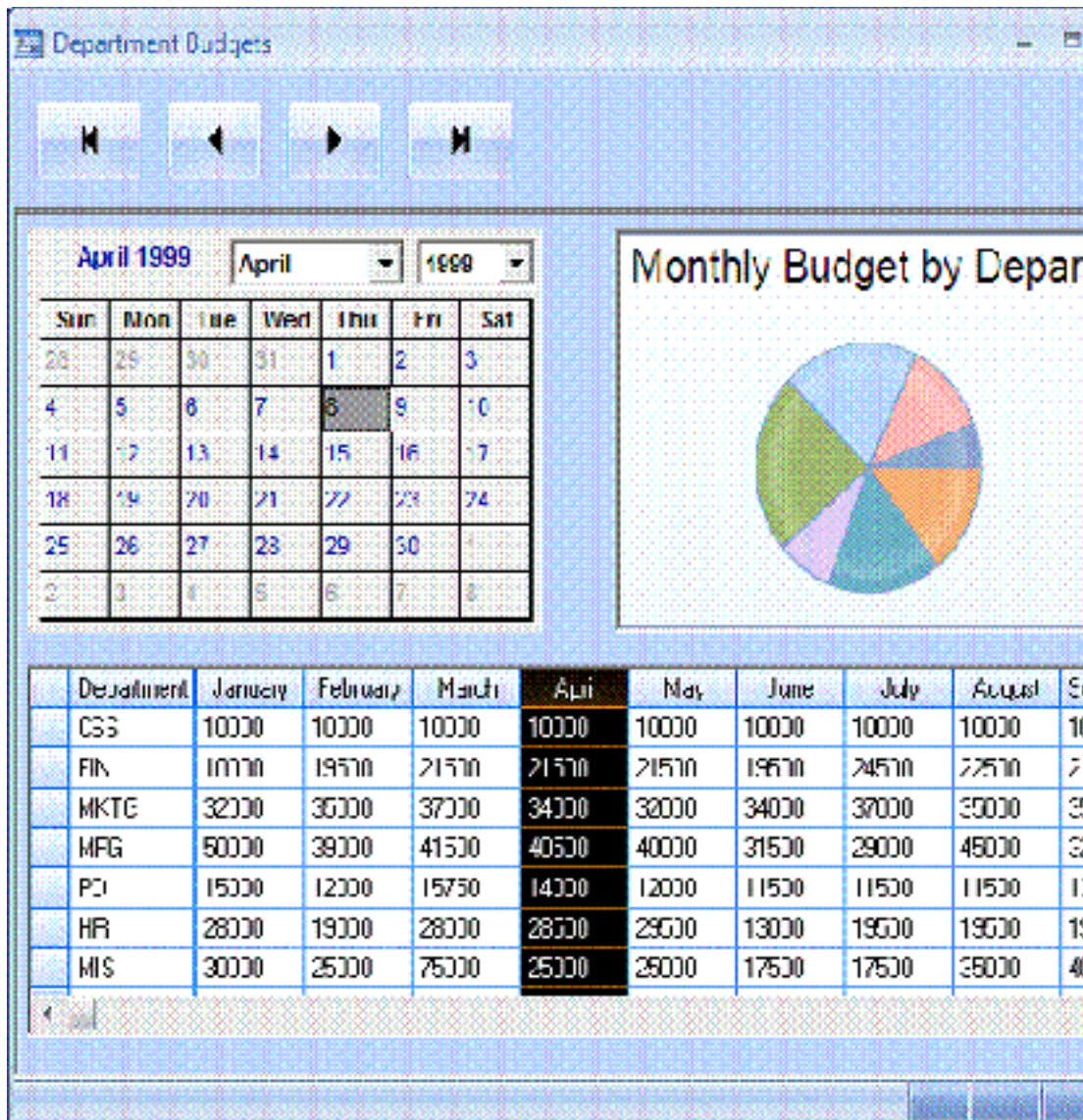
Use the navigation buttons to browse the database records. Notice how the calendar, the graph, and the table are linked.

You've just written a Gupta application with an embedded ActiveX object! In the next chapter, we'll take a close look at the concepts of object-oriented programming by building the table.apl library you used in this tutorial.

Chapter 4 – Developing N-Tier Applications Using TD and COM

The Component Object Model (COM) is a specification developed by Microsoft for writing reusable software, software that runs in a component-based environment. It provides an infrastructure that allows clients and objects to communicate across process and host computer boundaries. COM clients and objects may be developed using different programming languages, and can interoperate even when located on separate machines.

A complete discussion of COM is beyond the scope of this tutorial. If you are not familiar with COM concepts and principles, Gupta recommends that you consult a COM reference manual before proceeding. Most of the information you need to know about COM is available in the MSDN (Microsoft Developer Network) Library. See the Developing with Gupta SQLWindows manual for more information on COM programming in TD.

This tutorial is divided into the following sections:

- Advantages of COM applications

- Overview of the tutorial COM application

- Tutorial COM server

- Tutorial COM client

- Building a COM sever and COM client

- Troubleshooting

- Exercises for the reader

- Running the COM client as a Web application

- Developing Web applications

Advantages of COM Applications

Some of the advantages of developing an application using COM are the following:

- N-tier application environment

  A 2-tier application is typically composed of a client application that interacts with a server, usually a database. In an N-tier application, there is one or more middle tiers between the client and the database. The middle tiers centralize some of the complexity of an application, allowing client applications to be smaller and simpler. The application is effectively divided into a presentation layer (COM client), a business object layer (COM server), and a database layer.

- Programming language independence

  COM clients and servers can be developed in a variety of different programming environments, including SQLWindows, C++, Visual Basic, and ASP. In addition, a COM client developed in one programming environment, ASP for example, could interact with a COM server developed in a different programming environment, SQLWindows for example.

- Binary file reuse and file versioning

  COM allows you to reuse binary files within an application, across applications, and by other COM servers. It also includes a built in versioning system.

- Reduces the maintenance requirements for client applications

  Changes and enhancements can be made to COM servers without having to update the client applications that invoke the COM server functionality.

Overview of the Tutorial COM Application

The tutorial COM application files are included with Gupta Team Developer. The following files are discussed in this chapter:

- IslandSALESInfoSVR.app, a sample COM server

- IslandSALESOrderEntry.app, a sample COM client

Note: This application is relatively complicated. Walking through all the required steps is beyond the scope of this document. The tutorial provides some background on COM and on developing COM applications in Team Developer. The COM samples illustrate a variety of COM-related issues and provide developers with a working COM application to examine for future projects of their own.

The example COM application is a sales order entry system. Once compiled and running, it appears to function in much the same way as a typical client/server application. Data is drawn from the Island sample database. The user interacts with the COM client, selecting companies, creating sales invoices, submitting them to the database, and listing existing orders. However, much of the business logic of the application is run from the COM server.

When the client needs to use the functionality provided by the COM server, it first creates an instance of a COM object on the server. It can then call functions from the Interface associated with that object. The Interface functions make calls to the ISLAND database and pass data back to the client. Events fired by the Interface are handled by a CoClass on the server, which can pass the events back to the COM client. Using a COM Proxy Class, the client can then handle the event.

Running the Tutorial COM Application

To run the tutorial COM application, navigate to \Program Files\GUPTA\Tutorial (the default directory) in the Windows Explorer and double-click IslandSALESOrderEntry.exe.

Note: To use all the features of the tutorial COM application, the following must be installed: GUPTA Team Developer 2005, the GUPTA Web Extensions (CWE), SQLBase 9.0, and the Island sample database.

The tutorial application is mostly self-explanatory. You select a company (select a Company ID number and click GO), create an order for the company (click Place Order), fill out the order form, submit the order (click Submit Order), and review the completed invoice. From the first dialog box you can also list all of the invoices for a particular company (select a company ID, click GO, and click List Orders).

The following diagram illustrates the basic structure of the tutorial COM application, showing the relationships between push buttons, windows, combo boxes, fields, COM objects, and the database (not all relationships are shown):
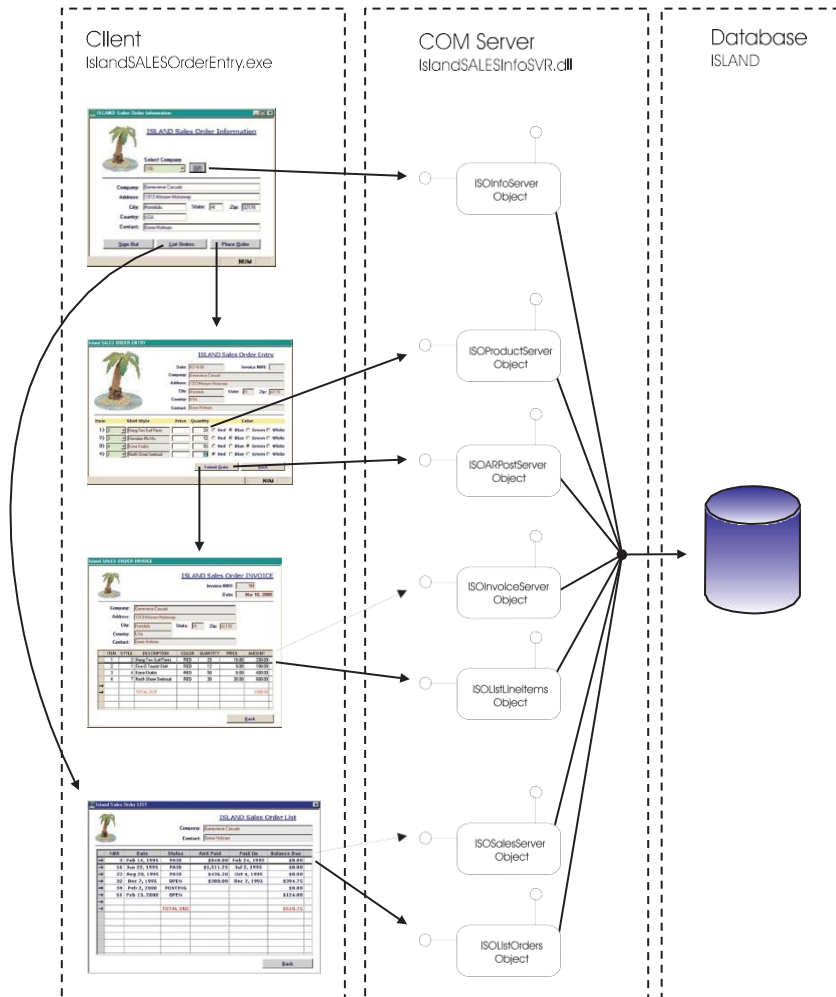
Tutorial COM Server

The IslandSALESInfoSVR.app file located in the \GUPTA\Tutorial directory contains the source code for the tutorial COM server. This section describes some of the features of COM servers and shows you how to create the basic elements of IslandSALESInfoSVR.app, including an Interface and CoClass. It also provides some advice on developing COM servers.

Creating Interfaces and CoClasses

This section explains how to create an Interface class and a corresponding CoClass using the COM Class Wizard in SQLWindows.

What is an Interface?

An Interface class contains the functionality of a COM server. Any business logic, database calls, or other functions that you decide to add to a COM server belong in an Interface. The details of how you have implemented functions in an interface are largely hidden from clients. A client only knows enough about an interface function to be able to use it. For a client to be able to use an interface function, it first must create

an instance of the COM object associated with the interface. The client can then call the interface function as if it were a function defined in the client application.

What does the IISOInfoServer Interface do?

Examine the code for the IISOInfoServer Interface in the IslandSALESInfoSVR.app file. IISOInfoServer models simple business processes for the fictional Island Outfitters company. It retrieves Company IDs and company contact information from the ISLAND database.

The following is part of the SAL code for the IISOInfoServer Interface:

```
Interface: IISOInfoServer
   Description: Island Sales Order Information Interface Class Attributes
      GUID: {45F27A32-C52C-11D3-B966-00485463324F}
   Derived From
   Class Variables
      Number: _nStopCompanyID
   Instance Variables
      ! By convention, names of class member variables are prefixed by "m".
      Number: m_nCount
      Number: m_nCompanyIDs[*]
      Sql Handle: m_hSql
      Number: m_nRetVal
      Functions
         Function: GetCustomerInfo
         Function: GetCompanyIDs
         Function: Item
         Function: PropGetCount
         Function: Add
         Function: Remove
         Function: StopSearch
```

Purpose of the other Interfaces

The other Interfaces in IslandSALESInfoSVR perform other needed functions for IslandSALESOrderEntry.exe. They have the following purpose:

- IISOARPostServer – Enables you can create a new invoice in the database.

- IISOProductServer – Enables you retrieve product information from the database. You can also increment and decrement the product inventory.

- IISOSalesServer – Used in conjunction with IISOListLineItems, enables you to retrieve invoice information from the database.

- IISOListLineItems – Enables you to retrieve item information for a particular invoice from the database.

- IISOInvoiceServer – Used in conjunction with IISOListOrders, enables you to populate a table with a list of the invoices for a particular company.

- IISOListOrders – Enables you to retrieve sales information for a particular invoice from the database.

Collections

A collection is an easy-to-use, managed list of items. It is extensible, allowing you to add and remove items. When you designate an Interface as a collection in the COM Class Wizard, a set of standardized functions is added to the Interface. These functions turn an ordinary Interface into a collection. The functions (Add, Item, PropGetCount, and Remove) allow you to manage the collection.

A collection of CoClasses can store or pass row data as a property of a CoClass. IISOSalesServer, in conjunction with IISOListLineItems and IISOInvoiceServer, in conjunction with IISOListOrders, act as collections of CoClasses. These interfaces are used for passing data from the COM server to the client. Each attribute defines an array data type, and each element of the collection corresponds to a specific row of data.

Both IISOListOrders and IISOListLineItems contain no functions, only Get Properties. When this functionality is implemented in the client, each Property is called. The data is placed in a table and displayed.

Note: By making the Interface a collection, you make it possible for the COM server to be used by client applications written in Visual Basic or C++. For more information on when and why you should use collections, consult a reference on COM and/or C++ programming.

Creating the IISOInfoServer Interface using the COM Class Wizard

The COM Class Wizard simplifies the process of creating a new CoClass, COM+ CoClass, or Interface for your application. It allows you to specify the derived class, functions, and properties of the ActiveX COM component. Although the wizard can generate a component, you must add the functionality of the component under Actions in the SQLWindows Outline.

To create the IISOInfoServer Interface using the COM Class Wizard, follow these steps:

1. Launch SQLWindows.

2. From a new application template, select **Component**, **Wizards**.

3. Select the **COM Class icon** and click **Start**. The COM Class Wizard dialog is displayed.
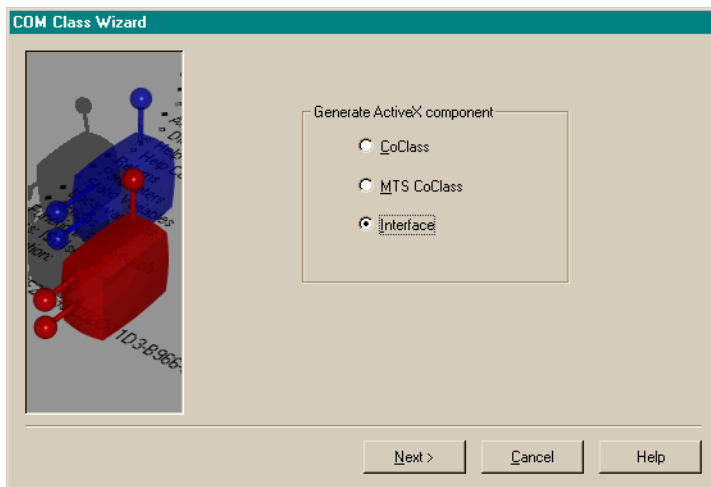
4. Select **Interface** and click **Next**.

5. Enter **IISOInfoServer** in the Name field and check the **Collection check box**.

6. Click **Next**.

    By making the Interface a collection, you make it possible for the COM server to be used by client applications written in Visual Basic or C++.

7. In the next dialog, you specify what the Interface is a collection of by selecting the **Data Type** radio button.

8. From the drop-down list, select **Boolean** and click **Next**.

Adding functions and properties. The next dialog of the COM Class Wizard allows you to add functions and properties to the Interface. The Wizard does not allow you to completely configure the Interface, but it does allow you to specify, at a high level, the functions and properties of the Interface. You must edit the Interface further in the Outline to make it complete.
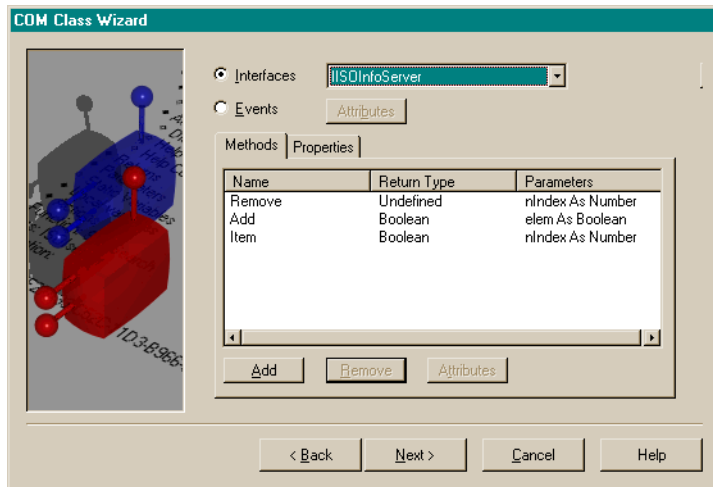
The Remove, Add, and Item functions along with the Count property are added automatically when you designate that the Interface is a collection.

The following sections describe how to add the following functions to the IISOInfoServer Interface:

- Get CustomerInfo
- Get CompanyIDs
- StopSearch

Creating the GetCustomerInfo function

1. On the Functions tab, click **Add**.

    The Add Function dialog box is displayed.

2. In the Name field, type **GetCustomerInfo**

3. Select **Boolean** from the Return Data Type drop-down menu and click **+**.

    The Add Argument dialog box is displayed.

4. In the Name field type **nCompanyID** and, in the Data Type drop-down menu, select **Number(VT_I2)**.

    VT stands for Variant Type. I stands for Integer. VT_I2 represents a 2-byte signed integer.

    Click **OK**.

5. Add the following arguments to the function (all of the following arguments should have a data type of String and should have the Receive box selected.):

    ▸ sCompanyName
    ▸ sAddress
    ▸ sCity
    ▸ sState
    ▸ sZip
    ▸ sCountry

▸ sContactName

The Add Function dialog box should appear as follows:
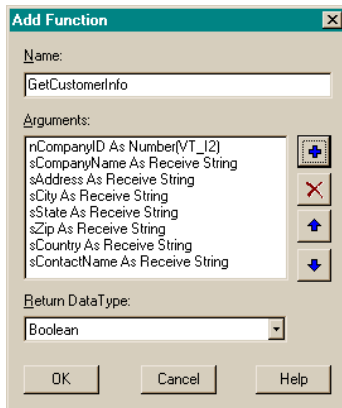
The order of these arguments is important. Make sure that the arguments are ordered as shown. Use the **-** button and the **¯** button to change the order of the arguments.

6. Click **OK** on the Add Function dialog box when you are finished.

Creating the GetCompanyIDs function

7. On the Functions tab, click **Add**.

   The Add Function dialog box is displayed.

8. In the Name field type **GetCompanyIDs**

9. Select **Boolean** from the Return Data Type drop-down menu and click **OK**.

Creating the StopSearch function

10. On the Functions tab, click **Add**.

    The Add Function dialog box is displayed.

11. In the Name field type **StopSearch**

12. Select **Boolean** from the Return Data Type drop-down menu and click **+**.

    The Add Argument dialog box is displayed.

13. Enter **nCompanyID** in the Name field.

14. Set the Data Type to **Number(VT_I2)**.

    VT stands for Variant Type. I stands for Integer. VT_I2 represents a 2-byte signed integer.

15. Click **OK**.

16. In the Add Function dialog box, click **OK**.

Finish creating the interface

17. In the COM Class Wizard dialog box, click **Next**.

    A summary of the Interface is displayed.

18. Click **Finish** to build the Interface.

The IISOInfoServer Interface is added to the application.

19. Save this application (the filename and location do not matter). You must use it in the next section.

Note: At this stage the IISOInfoServer Interface is not complete. Examine the Outline of the completed Interface in IslandSALESInfoSVR.app and compare it to the what you created. Note what the COM Class Wizard adds and does not add for you.

The following sections describe what a CoClass is and how to create one using the COM Class Wizard.

What is a CoClass?

When invoked by a COM client, a CoClass represents a COM object on the server. Using the CoClass, you can link to any associated Interfaces and use any of the Interface functions. In the COM server, it is derived from one or more Interfaces and defines any events related to the associated interfaces.

Note: You can derive CoClasses from multiple Interfaces. You can also derive separate CoClasses from the same Interface. As a novice COM programmer, it is best to derive each CoClass from a single Interface, for the sake of simplicity.

Creating the ISOInfoServer CoClass using the COM Class Wizard

20. Starting from the application you created in the Creating the IISOInfoServer Interface using the COM Class Wizard section, launch the **COM Class Wizard**.

21. Select **CoClass** in the Generate ActiveX component box and click **Next**.

22. Type **ISOInfoServer** in the Name field.

23. In the Derives From box, click **Existing interface** and click **Select**.

24. Select **IISOInfoServer** in the Existing Interfaces list.

    Click the **right arrow**. IISOInfoServer moves to the Derives from list.

    Click **OK**.

25. In the COM Class Wizard dialog box, click **Next**.

26. Select the **Events** radio button and click **Add**. The Add Event dialog box is displayed.

27. In the name field, type **evRecordNotFound** and click **OK**.

28. Add another event and name it **evResultsNotYetReady**. This event includes a single argument.

29. Click **+** to add the argument.

30. In the Add Argument dialog box, name the argument **nCompanyID**, give it the **Number(VT_I4)** Data Type, and click **OK**.

31. Click **OK** in the Add Event dialog box.

32. Add one last event and name it **evSearchError**. This event has no arguments.

33. Click **Next** on the COM Class Wizard dialog box.
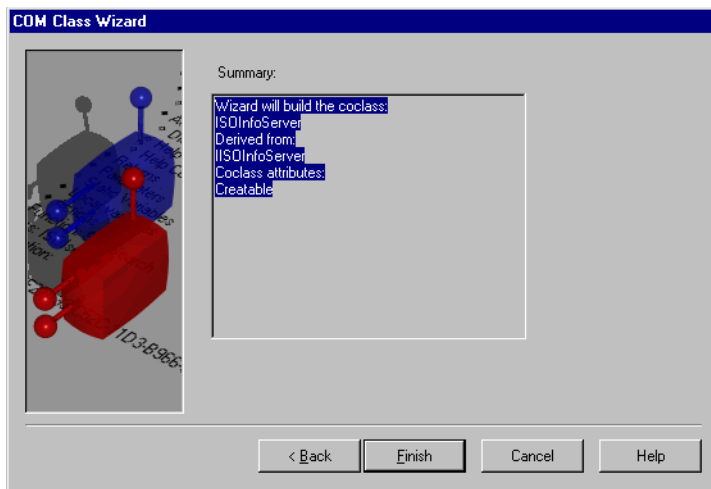
34. Click **Finish**.

The ISOInfoServer CoClass is added to the application. Unlike the IISOInfoServer Interface, this CoClass is complete. Nothing more needs to be added to it in the Outline, except optionally some information in the Description section.

COM Server Issues

The sections that follow explain a number of issues related to creating COM servers.

What belongs in a COM server

When you begin to develop a new COM-based application, one of the first things you should decide is which functions to place in a COM server versus which functions should remain in the client.

The tutorial COM server, IslandSALESInfoSVR, includes all of the functions that make calls to the database. Database calls are one general category of function calls you might place in a COM server. COM servers don't have to be limited to database related functions though. You could add to a COM server any function that might be reusable within a COM client application or in other COM client applications.

COM servers are non-visual, meaning that they cannot include any functions related to the user interface of an application. Any functions related to the user interface should be located in the client. You should also keep any functions that are specific to a single client application within that application.

EXE (out-of-process) versus DLL (in-process) COM servers

Out-of-process (EXE) COM servers can take advantage of the multi-threading in the SQLWindows runtime. When multiple clients access the server at the same time, the whole server does not have to be loaded into memory multiple times. This can increase both system performance and memory utilization. However, though system resources are conserved, the client and server have to marshall calls across process boundaries, a relatively time consuming operation.

In-process (DLL) COM servers do not incur this sort of performance penalty because when it is called the COM Server DLL is loaded into the client's own process space. When two client EXEs call the same DLL server, a copy of that server is created for each client.

GUIDs

Various COM components require an associated globally unique identifier (GUID). CoClasses, Interfaces, functions, events, and Type Libraries all require an associated GUID. GUIDs ensure that COM clients do not access COM components that have been altered in a manner that could cause conflicts with the client code. SQLWindows automatically adds a GUID to any new COM component.

A GUID can have a number of other names depending on what it identifies:

- GUID for a COM Interface is called an Interface Identifier (IID)

- GUID for a CoClass is called a Class Identifier (CLSID)

- In IDL GUIDs can be called Universally Unique Identifiers (UUID)

When to change the GUID or version information of COM components. When there are significant changes to the functionality of a Type Library, CoClass, event, or Interface, you must be sure to update the associated GUIDs before using the completed COM server in a production application. By updating the GUIDs, you prevent COM clients from accessing servers with changed properties. Note that when you change the properties of a CoClass or Interface, you should change the version information in addition to the GUID.

Updating a GUID. You can update GUIDs in the following ways:

- Right-click the GUID in the application outline and select **Regenerate GUID** from the pop-up list.

- Select Project, Regenerate GUIDs.
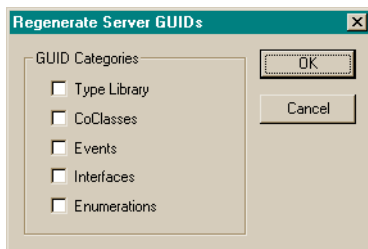
The Regenerate Server GUIDs dialog box allows you to regenerate all of the GUIDs in the COM server

- On the COM Server tab of the Project, Build Settings dialog box, you can change the type library GUID (this GUID identifies the whole component).

Tutorial COM Client

The tutorial COM client application, IslandSALESOrderEntry.app, is a windows driven database client for the ISLAND database. A user enters and requests information in a series of connected dialog boxes. When the process is complete, a sales invoice is created and stored in the database, and the product inventory is decremented to reflect the order.

The business logic of the tutorial COM application is in the COM server. The COM client is mostly devoted to the user interface calling functions in the COM server when business logic is needed.

Note: IslandSALESOrderEntry is configured as both a Windows and Web application. The web class libraries are included and, when needed, web class function calls have been made where normally you would use standard SAL function calls. This allows this application to work either as a Windows .exe or as a Web application that can be called from a browser (see "Developing Web Applications").

Using COM Server Functionality from the Client

The following sections outline how to use the functionality provided by the COM server from the COM client using IslandSALESOrderEntry.app as an example.

Here is a summary of how to implement the functionality of a COM server from a COM client (all of these steps must be completed in the SQLWindows workspace for the COM client):

1. Start ActiveX Explorer and select a COM server type library.

2. Examine the Interfaces and CoClasses in ActiveX Explorer to determine how you will use the COM server.

3. Use ActiveX Explorer to generate an APL with a hierarchy of COM proxy classes that invokes the methods of the server. ActiveX Explorer includes this APL in your current application.

4. If you want to handle events from the COM server, edit your application to add new COM proxy classes that are derived from the CoClasses in the APL.

5. Declare global variables for each of the COM server CoClasses.

6. Using the global variable associated with the CoClass, instantiate the COM object in the client code.

7. You can now invoke Interface functions or retrieve property values from the COM object.

ActiveX Explorer

When you begin to develop a new COM client application, the first thing you must do is to generate the Interfaces, CoClasses, and events in ActiveX Explorer. ActiveX Explorer takes the information from the COM server's type library and creates a COM Proxy Library (.APL file) that contains all the information needed for a COM client application to be able to use the COM server. The COM Proxy Library (APL) makes the SAL application aware of the COM server. It also gives the SAL application the ability to invoke the COM server.

Examining the Interfaces and CoClasses. Once a type library is generated, you can examine its Interfaces, CoClasses, and events using ActiveX Explorer.

Follow these steps:

1. In SQLWindows, open the **IslandSALESOrderEntry.app** application.

2. Select **Tools**, **ActiveX Explorer**.

Note: The first time ActiveX Explorer runs, it stores information about all the COM objects on your machine for future use. This process takes some time.

3. In the ActiveX Explorer Library Selection dialog box, select the **IslandSALESInfoSVR 1.0** Type Library and click **OK**.

   As shown below, the ActiveX Explorer allows you to examine all of the public features of a Type Library, all of the features you must know in order to invoke the COM server's objects in your application.

   ▸ The Classes window pane lists all of the Interfaces, CoClasses, and events available in the Type Library. Right-clicking in the Classes window opens a pop-up menu that allows you to show or hide the various COM components in the server and it also allows you to generate an APL source file that represents the information in the Type Library.
   ▸ If you select an item in Classes, the item's member elements are displayed in the Members window pane. For example, if you select the IISOProductServer Interface, its functions and properties are displayed.

- If you select an item from the Members window pane, the GetProductInfo function for example, then the function's parameters, returns, attributes, and derivations are displayed in the lower window pane.
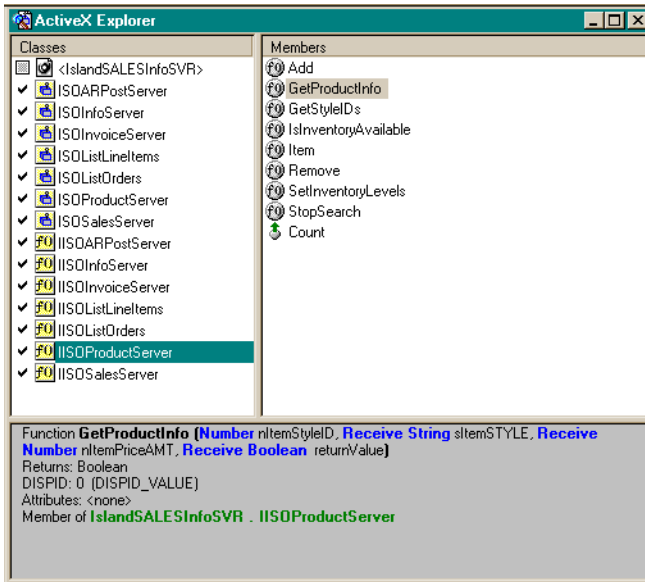
- Right-clicking in the lower window pane opens a pop-up menu that allows you to toggle between SAL syntax and generic COM syntax.

ActiveX Explorer uses the following icons:

Icon	Description
	Interface
	CoClass
	Event
	Function
	Get Property
	Set Property

As a client application developer, you might not be able to examine the internals of a particular COM server. However, using the ActiveX Explorer you can discover all you need to know to be able invoke the COM server's objects and use the server's functionality.

Generating the COM Proxy Library (APL). When you execute the Generate Full command in ActiveX Explorer, ActiveX Explorer does the following:

- Creates a Functional Class for each Interface in the type library.

- Creates a COM Proxy Class for each CoClass in the type library.

- Adds these classes to a new SAL library (APL).

- Includes this new library along with Automation.apl in your current application's outline.

Note: Automation.apl is required for all SAL COM clients.

If you open Global Declarations, Class Definitions in the outline of IslandSALESOrderEntry.app, you can see the Functional Classes and COM Proxy Classes associated with IslandSALESInfoSVR 1.0 Type Library.apl (listed in blue because they are included in the application):

```
Functional Class: IslandSALESInfoSVR_IISOARPostServer
Functional Class: IslandSALESInfoSVR_IISOInfoServer
Functional Class: IslandSALESInfoSVR_IISOProductServer
Functional Class: IslandSALESInfoSVR_IISOSalesServer
Functional Class: IslandSALESInfoSVR_IISOInvoiceServer
Functional Class: IslandSALESInfoSVR_IISOListLineItems
Functional Class: IslandSALESInfoSVR_IISOListOrders

COM Proxy Class: IslandSALESInfoSVR_ISOARPostServer
COM Proxy Class: IslandSALESInfoSVR_ISOInfoServer
COM Proxy Class: IslandSALESInfoSVR_ISOProductServer
COM Proxy Class: IslandSALESInfoSVR_ISOSalesServer
COM Proxy Class: IslandSALESInfoSVR_ISOInvoiceServer
COM Proxy Class: IslandSALESInfoSVR_ISOListLineItems
COM Proxy Class: IslandSALESInfoSVR_ISOListOrders
```

Functional Classes from the COM Proxy Library (APL). A Functional Class models each COM server Interface in the SAL programming environment. It makes available in the COM client application the functions for the associated Interface.

For example, the IslandSALESInfoSVR_IISOInfoServer Functional Class models the IISOInfoServer Interface in the IslandSALESInfoSVR COM server. If you examine the IslandSALESInfoSVR_IISOInfoServer Functional Class in the Outline for IslandSALESOrderEntry.app, you can see that it includes all of the functions of the associated interface:

- Function: GetCustomerInfo

- Function: GetCompanyIDs

- Function: Item

- Function: PropGetCount

- Function: Add

- Function: Remove

- Function: StopSearch

COM Proxy Classes from the COM Proxy Library (APL). The CoClass in the COM server represents the COM object. The COM Proxy Class makes it possible to create an instance of a specific COM object from the COM client. Each COM Proxy Class is derived from a Functional Class.

For example, the IslandSALESInfoSVR_ISOInfoServer COM Proxy Class is associated with the ISOInfoServer CoClass that is a component of the IslandSALESInfoSVR COM server. By declaring this COM Proxy Class in the COM client (a process that happens automatically when you generate the CoClass in ActiveX Explorer), you can create an instance of the ISOInfoServer object from the client. The COM Proxy Class also allows you to link to any interfaces associated with the CoClass. In the case of IslandSALESInfoSVR_ISOInfoServer, you can access the IISOInfoServer Interface.

COM proxy classes

In the previous section, you examined the Functional Classes and COM Proxy Classes that are included in the outline when you generate the Type Library in ActiveX Explorer. You must take an additional step if you want the client application to be able to handle events from the COM server. You must create subclasses for the COM Proxy Classes.

To handle COM server events, you must declare each CoClass in the server as a COM Proxy class under Global Declarations, Classes in the Outline:

```
COM Proxy Class: ISOInfoServer
   Derived From
      Class: IslandSALESInfoSVR_ISOInfoServer

COM Proxy Class: ISOProductServer
   Derived From
      Class: IslandSALESInfoSVR_ISOProductServer

COM Proxy Class: ISOARPostSever
   Derived From
      Class: IslandSALESInfoSVR_ISOARPostSever

COM Proxy Class: ISOSalesServer
   Derived From
      Class: IslandSALESInfoSVR_ISOSalesServer

COM Proxy Class: ISOInvoiceServer
   Derived From
      Class: IslandSALESInfoSVR_ISOInvoiceServer

COM Proxy Class: ISOListOrders
   Derived From
      Class: IslandSALESInfoSVR_ISOListOrders

COM Proxy Class: ISOListLineItems
   Derived From
      Class: IslandSALESInfoSVR_ISOListLineItems
```

When you derive a class, your new class inherits all the properties of the base class. Your new class lets you add additional functionality. In this case, it gives you the ability to handle COM server events in your client application.

Note: Class inheritance is an object-oriented programming feature of SAL. See the Developing with SQLWindows manual for more information.

Handling events. You handle COM server events in from the COM Proxy Classes.

For example, look under Class Definitions, COM Proxy Class: ISOInfoServer, Message Actions, Actions in the Outline:

```
Message Actions
   On evRecordNotFound
      Parameters
      Actions
         Call WebMessageBox( "Company/Contact Record could not be  located",
         "Search Error", MB_Ok )
```

This code displays an error message box in the client application when the ISOInfoServer CoClass fires an evRecondNotFound event.

Declare global variables for each of the IslandSALESInfoSVR CoClasses

Having created COM Proxy Classes to handle events, the next step is to declare the following variables for each COM Proxy Class under Global Declarations, Variables:

```
ISOInfoServer: comIslandINFO
!
ISOProductServer: comIslandPRODUCT
!
ISOARPostServer: comIslandARPOST
!
ISOSalesServer: comIslandSALES
!
ISOInvoiceServer: comIslandINVOICE
!
ISOListOrders: IslandORDERS
!
ISOListLineItems: IslandINVOICEITEMS
```

Once declared, you can use comIslandINFO, comIslandPRODUCT, comIslandARPOST, comIslandSALES, comIslandINVOICE, IslandORDERS, and IslandINVOICEITEMS to call functions in an interface or retrieve property values. The next section illustrates how this is done in IslandSALESOrderEntry.app.

Invoking a COM object and using an Interface function

In this section, segments of code from the COM client and the COM server are used to illustrate how to invoke a COM object and then use a function.

Example A. The code segment that follows illustrates how you create an instance of an object and call a function through that object from the client application. The segment is located in IslandSALESOrderEntry.app.

Using the Outline, navigate to cIslandSalesForm: frmSALESORDER, Message Actions, On Web_CreateComplete.

```
On WEB_CreateComplete
...
À  If comIslandINFO.Create()
Á  If comIslandINFO.GetCompanyIDs( bOK ) Call svrPopulateCMB( cmbCompanyIDs )
Call SalEnableWindow( pbGetINFO ) Call SalSetFocus( cmbCompanyIDs )
Call SalListSetSelect( cmbCompanyIDs, 0 ) Call frmSALESORDER.CustSelected( )
Â  If NOT WebIsWebMode( )
Call SalCreateWindow( frmLISTORDERS, hWndForm ) Call SalCreateWindow(
frmPLACEORDER, hWndForm )
! notice in windows mode can set the parent window as usual
Call SalCreateWindow( frmINVOICE, frmPLACEORDER )
```

À   This line of code creates an instance of the ISOInfoServer object. Before you can use the functionality provided by an object, you must create an instance of it.

Á   This line of code calls the function GetCompanyIDs in the IISOInfoServer Interface. The lines of code that follow obtain the CompanyIDs from the ISLAND database and uses them to populate a Combo box in frmSALESORDER.

Â   These final lines of code are related to making the application work as a web application or a windows application.

Example B. The code segments that follow show how to call the GetCustomerInfo function and show how the code on the client corresponds to the code on the COM server.

The first segment is located in IslandSALESOrderEntry.app under cIslandSalesForm: frmSALESORDER, Contents, cWebButton: pbGetINFO.

```
cWebButton: pbGetINFO
   Message Actions
      On SAM_Click
         Call frmSALESORDER.CustSelected( )
         Call frmSALESORDER.GetCustInfo( )
```

The pbGetINFO button is the first button a user clicks in this application. After selecting a Company ID from the scroll list, the user clicks this button to obtain the information on the corresponding company from ISLAND. In the preceding code sample, the call of frmSALESORDER.GetCustInfo( ) starts the process of retrieving company information.

Here is the code for frmSALESORDER.GetCustInfo(), located under cIslandSalesForm: frmSALESORDER, Functions, Function: GetCustInfo:

```
Function: GetCustInfo
   ...
   Actions
      ...
      If nCompanyIDSelected > 100
         Set dfsCompanyName = ""
         Set dfsAddress = ""
         Set dfsCity = ""
         Set dfsState = ""
         Set dfsZip = ""
         Set dfsCountry = ""
         Set dfsContactName = ""
   À  If NOT comIslandINFO.GetCustomerInfo( nCompanyIDSelected,
      dfsCompanyName, dfsAddress, dfsCity, dfsState, dfsZip,
      dfsCountry, dfsContactName, bOK )
```

À   This line of code calls the function GetCustomerInfo in the IISOInfoServer Interface and obtains the company contact information from the database.

The first argument in this function call, nCompanyIDSelected, is defined as a SetProperty (the Company ID is passed to the server). The remaining arguments are defined as GetProperties (the Company information associated by the Company ID is passed back to the client).

The next code sample shows part of the function, GetCustomerInfo, as it is defined in the IISOInfoServer Interface in IslandSALESInfoSVR.app. In the Outline view, open Global Declarations, Class Definitions, Interface: IISOInfoServer, Functions, Function: GetCustomerInfo.

```
Function: GetCustomerInfo
   Description: Returns Customer Information for Company ID
passed in
   Attributes
   Returns
   Parameters
      Number: nCompanyID
      Receive String: sCompanyName
      Receive String: sAddress
      Receive String: sCity
      Receive String: sState
      Receive String: sZip
      Receive String: sCountry
      Receive String: sContactName
   Static Variables
   Local variables
```

```
Actions
   Set SqlDatabase='ISLAND'
   Set SqlUser='SYSADM'
   Set SqlPassword='SYSADM' If
   SqlConnect( m_hSql )
      ...
   If SqlPrepareAndExecute( m_hSql,
   "SELECTcompany_name,
      address,
      city,
      state,
      zip,
      country,
      cont_first_name || ' ' || cont_last_name
   FROM company a, contact b
   INTO :sCompanyName,
      :sAddress,
      :sCity,
      :sState,
      :sZip,
      :sCountry,
      :sContactName
   WHERE a.company_id = :nCompanyID
   AND a.company_id = b.company_id"
```

In the Actions section of the preceding code sample, the actual connection is made to the ISLAND database and the relevant data is retrieved. The way the tutorial application is designed, the location of the database and the way in which the data is retrieved are irrelevant to the COM client.

Building a COM Server and COM Client

The sections that follow describe how to build a COM server and a COM client from completed source files.

Building a COM Server

This section describes how to build and register a COM server .dll file. In the process of building a COM server .dll file, SQLWindows also generates a corresponding Type Library.

To build a COM server, follow these ssteps:

1. Launch **SQLWindows**.

2. Open the completed .app file for the COM server.

3. Open the **Build Settings** dialog box by selecting **Project** > **Build Settings**.

4. Specify the name of the .dll file in the Target name box.

5. Build the .dll file by selecting **Project** > **Build: ..\nameoffile.dll**.

   Click **OK** in the Build Information dialog box.

   A dialog opens stating that the build was successful.

   When SQLWindows generates the .dll file, it also generates a corresponding Type Library (nameoffile.tlb). The type library contains the type definitions of all of the COM Components in the COM server. The

developer of a client application can understand and invoke the functionality of this server based on the information in the type library.

6. Register the server.

   Select **Project** > **Register Server**. SQLWindows registers the COM server in the Windows registry. All of the COM component in the server are registered: Interfaces, CoClasses, functions, and events.

   A dialog is displayed indicating that the registration was successful.

Note: If you previously registered the server, un-register the server before attempting to re-register it. Make sure to first close any applications that might be using the COM server, including the database. To un-register the server, select **Project**, **Un-Register Server**.

Building and Running the COM Client

This section describes the steps you must complete to incorporate a Type Library in a client application, and then build and run the application.

To build and run a COM client application, follow these steps:

1. Open the completed .app file.

2. Open ActiveX Explorer by selecting **Tools** > **ActiveX Explorer**.

   The ActiveX Explorer Library Selection dialog box is displayed.

Note: The first time you open the ActiveX Explorer, it may take a while to appear. It needs to read the registry to find all of your registered COM objects. This information is cached for future use.

3. Select the appropriate type library from the list and click **OK**.

   The ActiveX Explorer is displayed.

4. Right-click in the **Classes** window pane of the ActiveX Explorer and select all the components you want to generate.

5. Right-click in the **Classes** window pane of the ActiveX Explorer and select **Check All Shown** from the pop-up menu.

6. Right-click in the **Classes** window pane of the ActiveX Explorer and select **Generate Full** from the pop-up menu. The COM Proxy Library (APL) is generated. It exposes the functionality of the COM server to the client application.

Note: If a message box is displayed asking whether you would like to regenerate all of the selected classes, click **Yes**.

7. Close the **ActiveX Explorer**.

The client application is now ready. To run the application, you can either build the .exe using the Project, Build command, or you can run the client in debug mode by selecting Debug, Go.

Troubleshooting

This section describes a problem you might encounter when running the tutorial COM application, along with what you can do to remedy it.

OLE Automation Run-time Error

```
Error Code at invocation of : 80002000e
```

This error message is typically triggered because the new or revised COM server .dll file was not registered. You must register the COM server .dll file.

Exercises

The following exercises can help give you a better sense of how COM applications are structured and can show you how to get COM applications to function properly using SQLWindows:

- In IslandSALESOrderEntry.app, there is a function called ValidOrder defined under cIslandSalesForm: frmPLACEORDER.

  Exercise: Remove the ValidOrder function from the client, add it to the COM server, and then call the function from the client to perform the original task. You must regenerate the COM server Type Library, reregister the server, and generate a new proxy class using ActiveX Explorer.

- To limit the complexity of the tutorial COM application, all of the Interfaces were located in the same COM server. For a real-world application, you might not structure the application in this manner.

  Exercise: Separate each Interface and associated CoClass into its own COM server and then rework the COM client to invoke the COM objects from the separated COM servers.

- The ISLAND database includes multiple sales contacts for each company. However, the tutorial application does not allow you to pick which contact to place an order with.

  Exercise: Add the functionality needed to allow a user to list and select which sales contact to place an order with based on the Company ID.

- A common need for most businesses is the ability to keep track of inventory.

  Exercise: Create a new COM client application for monitoring product inventory levels in the ISLAND database. Include a function that allows you to add/order inventory when you are running low.

Running the COM Client as a Web Application

This section describes how to run the tutorial COM client as a Web application. The application is already Web-enabled and maintains its N-tier architecture in a Web environment. The IslandSALESOrderEntry client uses the Gupta Web Extension (CWE) objects along with the COM objects provided by the IslandSALESInfoSVR COM server. Because it uses the CWE objects, no code changes are needed to deploy the same application in a Windows environment or in a Web environment. Refer to the previous sections for details on how to create a COM-based application for a Windows environment.

For information on how to create GUPTA Web applications, see the *Building Web Applications with GUPTA* document.

Running the COM Application from the Web

If you installed Gupta Team Developer 2005 using the default installation settings and on a machine running a Web server, you only need to follow these steps to run the COM client from the web.

1. Launch your Web server.

2. Launch the Gupta App Manager.

   Run the Gupta AppConsole (**Start**, **Programs**, **GUPTA**, **Team Developer 2005**, **AppConsole**) and expand the WAM server in the left pane to show which application services are defined.

   Select the **WAM server** in the tree view, click the **App Manager** tab, and click **Start App Manager** (if the App Manager is currently stopped).

   Launch the **SQLBase Database Engine**.

3. Launch **Microsoft Internet Explorer** and open the following URL (the URL is case sensitive):

   http://*webservername*/scripts/ cwisapi.dll?Service=IslandSALESOrderEntry

   webservername is the name of the web server accessible by the Gupta WebAppManager Server.

   This URL varies depending on the Web Server you are using. For more information, check the documentation for your Web Server.

   The Web version of the IslandSALESOrderEntry application opens in the browser window. It functions in a manner similar to the Windows version.

Running a New COM Application on the Web

This section describes the steps you must complete in order to run a Web-enabled COM client such as IslandSALESOrderEntry.exe on the Web using the Gupta WebAppManager Server. Most of these steps are completed automatically for the tutorial COM application when it is installed from the TD installation CD.

This section assumes you already have a working Gupta Web-enabled COM application. For information on how to create Gupta Web applications, see the Building Web Applications with Gupta manual.

To run a new Gupta Web-enabled COM application from a Web browser, follow these steps:

1. Install your Web server; for example, Microsoft IIS.

   You can use any Microsoft Windows compatible Web server to run your Gupta COM clients.

2. Install Gupta Team Developer 2005.

   The installation includes the Gupta Web Extensions (CWE) and the Gupta WebAppManager (WAM) Server. The WAM executables are copied to the \scripts directory of the Web server and a new directory, called CWD, is created under \wwwroot.

3. Make sure that your COM application functions properly.

   The COM server should be registered and should be accessible by the COM client. The COM client application should be in a directory accessible to the WAM server.

4. Copy the graphics for the Web application to a directory accessible to the WAM server.

5. Create the WAM application service.

    Run the Gupta AppConsole (Start, Programs, GUPTA, Team Developer 2005, AppConsole) and expand the WAM server to show the defined application services.

    To create a new WAM application service:

    a. In the tree view, select the **WAM server**.
    b. Click the **New Application** tab in the right window pane.
    c. Enter the service name; for example, NewCOMApp (this name is case sensitive).
    d. Click **Browse** and navigate to your COM client application (the .exe file). Select the application and click **Open**.
    e. You can change the default settings, but it should not be necessary. Click **OK**.
    f. Select the **WAM server** in the tree view, click the **App Manager** tab, and click S**tart App Manager**. Confirm that the NewCOMApp service is running by selecting it in the tree view.

6. Create a WEB Server alias to your Web Server for the Gupta graphics files used by your COM client, as follows:

    a. In Windows NT 4.0, select **Start**, **Programs**, **Windows NT 4.0 Option Pack**, **Microsoft Internet Information Server**, **Internet Service Manager**.

        The Microsoft Management Console is displayed.

    b. Navigate to Default Web Site in the tree view and then right-click **Default Web Site** and select **New**, **Virtual Directory**. The New Virtual Directory Wizard is displayed.
    c. Enter the alias name; for example, graphics. Click **Next**.
    d. In the next screen, enter the directory path for the directory that contains your web graphic files. Click **Next**.
    e. Select access permissions for the directory. Make sure to at least allow Read access. Click **Finish**.

7. To run the application from a Web browser, launch your web browser and navigate to the following URL:

    http://*webservername*/scripts/cwisapi.dll?Service=*NewCOMApp*

    webservername is the name of your Web Server

    NewCOMApp is the name of your WAM application service

    This URL varies depending on the Web Server you are using. For more information, see the Building Web Applications with GUPTA document.

The application is invoked on the web server and the opening screen is displayed.

Developing Web Applications

This section provides some information to help you develop Web/COM applications.

Building Web Applications

The Gupta Web Extensions include a class library of objects called the Web QuickObjects used to build Web applications. The objects in the Web QuickObjects class library appear as standard Windows objects in the SQLWindows Layout tab. They also behave like standard Windows objects when you run the application in debug mode or as a stand-alone executable. However, when they are deployed with the Web App Manager, they generate Web content such as HTML, XML, and JavaScript in response to Web requests.

The following steps show how to generate the Web content for the IslandSALESOrderEntry.app in SQLWindows. The content is displayed in a window in raw form and in a browser as it would appear to a user.

1. Open the **IslandSALESOrderEntry.app** application in SQLWindows, right-click **frmWebManager** in the tree view (in the Windows folder).

2. Select **cWebMgrForm Properties** from the pop-up menu.

   The frmWebManager Properties dialog box is displayed.

3. Check the **HTML Generation Test Mode** checkbox on the Properties dialog and click **OK**.

4. Select **Debug**, **Go**.

   The Web Manager Form window is displayed. This window is normally hidden.

5. Select **frmSALESORDER** in the combo box and click **Show Web String**.

   The following things happen:

   ▸ The Island Sales Order Information window is displayed.
   ▸ The Web Manager Form window displays the Web page content that would be sent to the user's web browser.
   ▸ A separate window with an Internet Explorer control appears, displaying the Web content as it would appear to the user. It is not functional, but it helps you verify that the Web page layout is satisfactory.

Building a New Web Application in SQLWindows

To build a new Web application, you start with a new Web application template (newweb.app). This template already includes a Web Manager Form and an object of class cWebMgrForm. The Web Manager Form automatically responds to requests from the Web App Manager, and routes them to the appropriate object in the application. All Web applications need a Web Manager Form, except those that serve raw Web content using the cWebRaw class object. However, the most common Web applications are built with the cWebDedicatedForm or cWebReusableForm class objects.

Dedicated versus Reusable Web Applications

The Gupta Web Extensions support two ways to build Web applications: dedicated and reusable. In a Dedicated application, the Web App Manager creates a separate application thread to service each user. By creating a separate application thread for each user, state and session information is managed independently for each user. This approach is relatively simple to implement, however it does consume more system resources, and is best suited to environments where speed of development is more important than scalability.

In a Reusable application (IslandSALESOrderEntry is a Reusable application), the Web AppManager allows many users to share the same application thread. However, the developer must design the application in a stateless manner. You cannot assume that the application is in any particular state when a user enters it, because other users may have left it in an undetermined state. You must save each user's state after the application finishes handling a request, and restore that user's state when they make a new request.

Stateless applications. Since IslandSALESOrderEntry is a stateless application, it does not maintain a permanent connection to the database or the COM server.

When each client function call is completed, all database or COM server handles are released.

Building a Dedicated application. To build a Dedicated application, you start with a cWebDedicatedForm Window. You can also use Modal Dialog Boxes.

Building a Reusable application. To build a Reusable application, you start with a cWebReusableForm window. To ensure that the application is reusable, never use Modal Dialog Boxes. When a Modal Dialog Box is created, the application behaves as a Dedicated application.

Updating the client .exe

Whenever you update the client .exe file, you must stop the Application service in the App Manager of the WAM server before you can begin to use the updated application. After updating the .exe file, restart the Application service.

Debugging a Web Application

To help you debug your application, activate the SQLWindows Enable Playback feature. When enabled, all communication between various parts of the web application is written to a log file. A close examination of the log should reveal where the error or errors occurred.

To enable the playback feature, follow these steps:

1. Select **Project**, **Build Settings**.

2. On the Build Target tab, click the **Enable Playback** checkbox.

The Build Settings, Record/Playback feature of SQLWindows is especially useful for debugging applications built upon COM, MTS, COM+, and ASP.

Note: There is a significant performance penalty when this feature is enabled. Be careful to disable this feature on shipping code.

Chapter 5 – Using Team Developer and COM+

COM allows you to separate the business logic of an application from the application interface. The business logic is moved to COM objects running in COM servers. This application architecture makes it possible to allow large numbers of client applications to access a single centralized COM server remotely. COM+ helps to make centralized COM servers work more efficiently and effectively.

For TD developed COM servers, COM+ acts as a communications broker between COM objects. It handles requests from remote COM clients to the COM objects it manages, along with requests between COM objects. COM+ can also handle transactions for COM clients created in environments such as ASP.

Microsoft created COM+ (formerly Microsoft Transaction Server or MTS) in an attempt to simplify the development of large distributed applications. It is part of a category of programs known as middleware, multi-tier, or N-tier applications.

This chapter is divided into the following sections:

- Running the tutorial COM server in COM+

- Converting the COM tutorial application to use COM+

- Creating an ASP client for a COM server

Note: To run the COM+ tutorial application, you must be running Gupta Team Developer 2005 or higher on Windows NT 4.0, 200, 2003, or XP. Windows NT 4.0 systems must have the Microsoft Transaction Server Option Pack 4 installed.

Running the Tutorial COM Server in COM+

This section describes how to run the COM+ versions (provided with TD) of the tutorial COM client and server.

The files are called:

- IslandSALESInfoSVRMTS.dll

- IslandSALESOrderEntryMTS.exe

Note: The provided samples still retain the "MTS" acronym for purposes of backward compatibility with earlier versions of Team Developer. MTS is the old acronym (from Windows NT and Microsoft Transaction Server) for what is now COM+ in more recent versions of Windows.

As with the standard COM server, the COM+-enabled COM server is automatically registered when it is installed from the TD installation CD. In order to use the COM+-enabled server from within COM+, you only need to install the server as an application in COM+.

Installing the Tutorial COM+ Server in COM+

The steps that follow show how to install the tutorial COM+ server as an application in COM+. For detailed instructions on how to use COM+, consult the Microsoft documentation.

Note: These steps describe the menus as they are specified in Windows 2000. The Windows NT menus are somewhat different. Those menus can be found in Appendix A.

To install the COM+ server as an application in COM+, follow these steps:

1. Close SQLWindows, SQLBase, and any other application that might be using the tutorial COM server.

2. Select Start, Settings, Control Panel, and then choose Administrative Tools and Component Services.

3. Expand the Component Services tree to, Computers, My Computer, COM+ Applications.

4. Select COM+ Applications, right-click, and select New, Application.

5. In the Application Wizard, click Create an Empty Application and name the new application IslandMTS. Click Next.

6. Use the default settings in the Set Application Identity dialog box. Click Next. On the next screen, Click Finish.

7. Expand the new IslandMTS package in the tree view.

8. Select Components, right-click, and select New, Component. The Component Wizard is displayed. Click Next.

9. Click Import components that are already registered.

10. Select the following components:

    ▸ IslandSALESInfoSVRMTS.ISOARPostServerMTS
    ▸ IslandSALESInfoSVRMTS.ISOInfoServerMTS
    ▸ IslandSALESInfoSVRMTS.ISOProductServerMTS
    ▸ IslandSALESInfoSVRMTS.ISOSalesServerMTS
    ▸ IslandSALESInfoSVRMTS.ISOInvoiceServerMTS
    ▸ IslandSALESInfoSVRMTS.ISOListOrdersMTS
    ▸ IslandSALESInfoSVRMTS.ISOListLineItemsMTS

    By checking the Details checkbox, you can confirm that the components you select are coming from the correct DLL. (The default is ...\GUPTA\Tutorial\IslandSALESInfoSVRMTS.DLL.)

    Click **Next**.

11. On the next screen, click **Finish**.
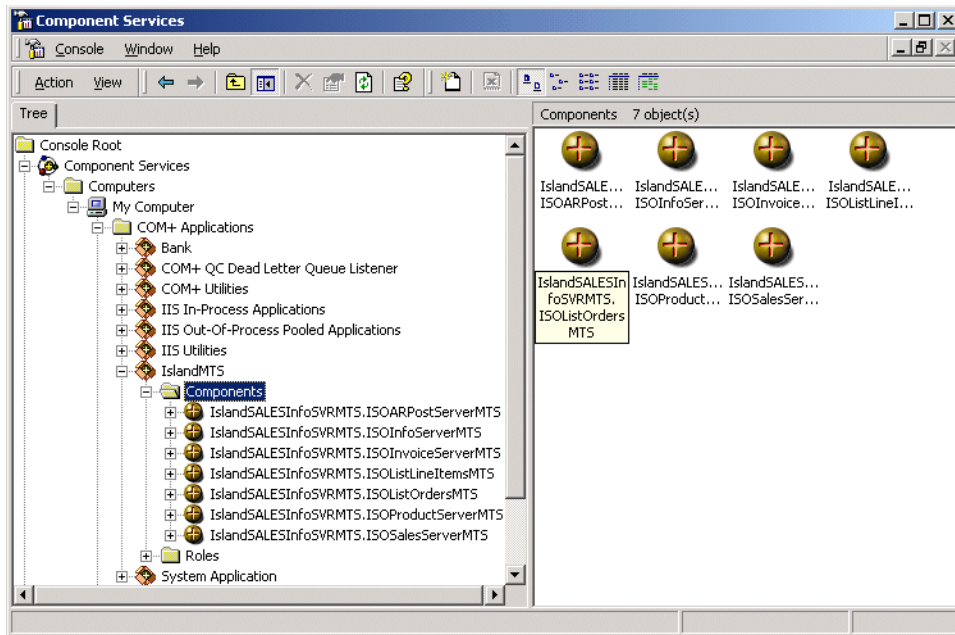
The COM+ Server objects are displayed in the right window pane.

Now that you have installed the COM+ server components in COM+, COM+ can act as a broker for the objects when called by the COM+ client application. Open IslandSALESOrderEntryMTS.app and run the application in debug mode (or launch IslandSALESOrderEntryMTS.exe). If you have the Microsoft Management Console open while you are running the client, the ball graphic spins for each object that is currently being used by the client.

Note: Each time you revise an COM/COM+ server, you should delete the old COM+ server package and then reinstall the revised COM server in a new COM+ server package.

Converting the COM Tutorial Application to Use COM+

This section describes how to modify IslandSALESInfoSVR.app and IslandSALESOrderEntry.app so that they can use COM+. You are shown how to create IslandSALESInfoSVRMTS.app and IslandSALESOrderEntryMTS.app (provided with the TD installation) based on the tutorial COM .app files.

Note: The procedures described in this section overwrite IslandSALESInfoSVRMTS.app and IslandSALESOrderEntryMTS.app in the \GUPTA\Tutorial directory. You can of course reinstall these files from the installation CD if necessary.

Converting the COM Server to Use COM+

To modify the COM tutorial server so that it can run in COM+, follow these steps:

1. Shut down SQLBase, IIS, WAM, or any other system that might be using the COM tutorial servers, IslandSALESInfoSVR.dll, and IslandSALESInfoSVRMTS.dll.

2. Open **IslandSALESInfoSVRMTS.app** from \GUPTA\Tutorial in SQLWindows.

3. Unregister **IslandSALESInfoSVRMTS.dll** by selecting Project, Un-Register Server.

4. Close **IslandSALESInfoSVRMTS.app** in SQLWindows.

5. Navigate to \GUPTA\AXLibs using Windows Explorer and delete **IslandSALESInfoSVRMTS 1.0 Type Library.apl**.

6. Open **IslandSALESInfoSVR.app** in SQLWindows.

7. Unregister **IslandSALESInfoSVR.dll** by selecting Project, Un-Register Server.

8. Save **IslandSALESInfoSVR.app** as **IslandSALESInfoSVRMTS.app** overwriting the existing file in the \Tutorial directory.

Create new COM+ CoClasses

To create a new COM+ (MTS) CoClass for each of the interfaces in IslandSALESInfoSVRMTS.app, follow these steps:

1. Open **IslandSALESInfoSVRMTS.app** in SQLWindows.

2. Select **Component**, **Wizards**.

3. Select the **COM Class icon** and click **Start**.

4. Select **MTS CoClass** and click **Next**.

5. Enter **ISOSalesServerMTS** in the Name field in the CoClass box.

6. Derive ISOSalesServerMTS from an existing interface (created for the original COM server application).

   In the Derives From box, click the **Existing Interface** radio button.

   Click **Select**. The Interface Picker dialog box is displayed.
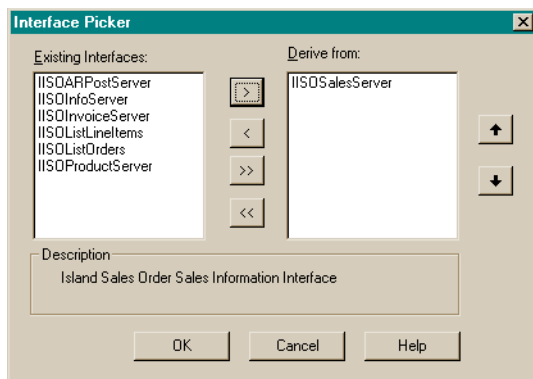
7. Select **IISOSalesServer** from the Existing Interfaces list and click the **right arrow**. The selected item moves to the Derives from list.

   Click **OK** and then click **Next**.

8. Click **Next**. The COM Class Wizard Summary is displayed. Verify that the information displayed is correct. Click **Finish**.

9. Repeat the preceding steps to create the following CoClasses for the MTS tutorial application:

   ▸ ISOInfoServerMTS Derived from IISOInfoServer
   ▸ ISOProductServerMTS
   ▸ Derived from IISOProductServer.
   ▸ ISOARPostServerMTS
   ▸ Derived from IISOARPostServer.
   ▸ ISOInvoiceServerMTS Derived from IISOInvoiceServer

- ISOListOrdersMTS Derived from IISOListOrders
- ISOListLineItemsMTS Derived from IISOListLineItems

In the Outline under Global Declarations, Classes, you should now have seven new MTS CoClasses.

10. Regenerate all of the GUIDs in the COM server. Select **Project**, **Regenerate GUIDs**. Check all of the items listed and click **OK**.
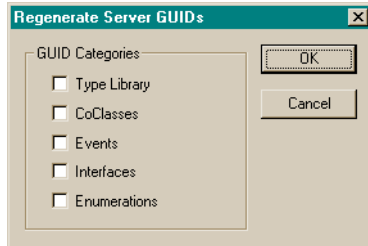
Copy the code for the events to the COM+ CoClasses

If you open one of the new COM+ CoClasses in the Outline—ISOSalesServerMTS, for example—you notice that the code for the events present in the original COM CoClass is missing. You must copy these events in the Outline from the old COM CoClasses to the new COM+ CoClasses.

To copy the events from the old COM CoClasses to the new COM+ CoClasses, follow these steps:

1. Open the **Outline** for **IslandSALESInfoSVRMTS.app** to Global Declarations, Class Definitions.

   The original COM interfaces and CoClasses are displayed along with the included classes and the new COM+ CoClasses.

2. Open **CoClass: ISOSalesServer**, **Events**. Copy the three events:
   - evRecordNotFound
   - evResultsNotYetReady
   - evSearchError

3. Paste the events under **CoClass: ISOSalesServerMTS**, **Events**.

4. Complete the steps in this section for the following MTS CoClasses:
   - ISOInfoServerMTS
   - ISOProductServerMTS
   - ISOARPostServerMTS
   - ISOInvoiceServerMTS

Note: ISOListOrdersMTS and ISOListLineItemsMTS have no events

5. There are a number of references in the application to the original COM CoClasses. These references must be changed to the new COM+ CoClasses.

   Search for each instance of the original CoClass names and replace them with the new names:

   Rename from:
   - ISOSalesServer
   - ISOInfoServer
   - ISOProductServer
   - ISOARPostServer
   - ISOInvoiceServer
   - ISOListOrders

- ▶ ISOListLineItems

Rename to:

- ▶ ISOSalesServerMTS
- ▶ ISOInfoServerMTS
- ▶ ISOProductServerMTS
- ▶ ISOARPostServerMTS
- ▶ ISOInvoiceServerMTS
- ▶ ISOListOrdersMTS
- ▶ ISOListLineItemsMTS

You can use the Edit, Replace dialog box to complete this process quickly.
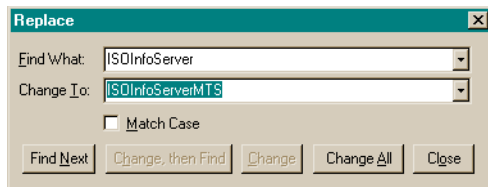
Note: Do not change the references to any of the interfaces.

6. When you finish copying the events from the old CoClasses to the new COM+ CoClasses, comment out each of the COM CoClasses.

! CoClass: ISOInfoServer

! CoClass: ISOProductServer

! CoClass: ISOARPostServer

! CoClass: ISOSalesServer

! CoClass: ISOInvoiceServer

! CoClass: ISOListOrders

! CoClass: ISOListLineItems

To comment a line of SAL code, select the line, right-click, and select **Comment Items** from the popup menu. The following graphic shows how your application window should appear once you complete the preceding steps.
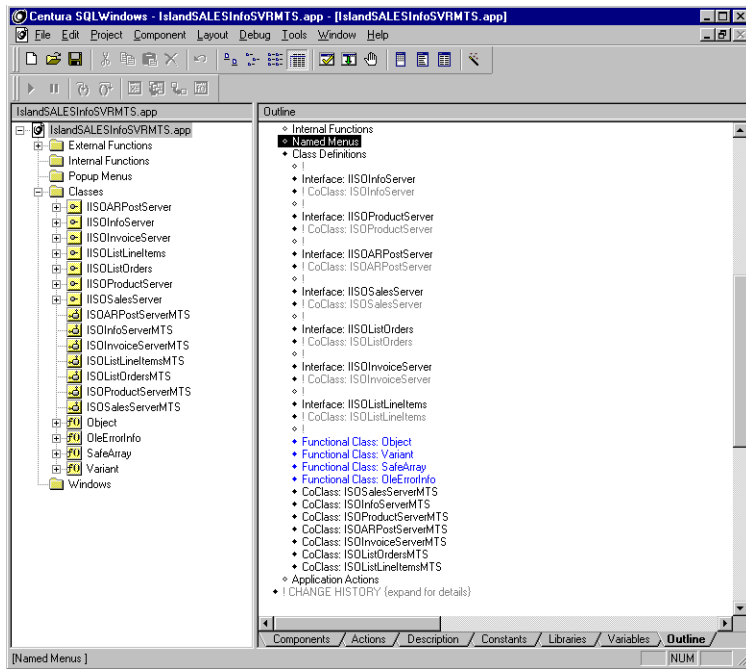
In the tree view shown in the left window pane, each Interface should now have a single corresponding CoClass.

7. Save the file.

Building the COM+ server

To build the COM+ server, follow these steps:

1. Select **Project**, **Build Setting**.

2. In the Build Settings dialog box, select the **Build Target** tab, set the Target Type to **MTS COM Server**.

3. Rename the Target Name IslandSALESInfoSVRMTS.dll.

4. Click the **Version** tab in the Build Settings dialog.

   In the Product Name field, add MTS to the name so that it reads **MTS ISOIServer**.

5. Click **OK** and save the application.

6. Build the application (select Project, Build). The Build Information dialog box is displayed.

   Click **OK**. A message box should open stating that the build was successful.

7. Register the Server by selecting **Project**, Register Server. You should get a message that the server registered successfully.

Note: If you receive an error message when you attempt to register the server, it may be that a process on the machine still has a handle to the old COM server. To ensure that all of these connections are free, reboot the system.

You have now changed the tutorial COM Server so that it can function from within an COM+ Server. In the next section, you change the tutorial COM client application so that it can use the COM+ Server.

Modifying the Client to Use the Server

To modify the COM tutorial application to work with the COM+ server, follow these steps:

1. Open **IslandSALESOrderEntry.app** from the Tutorial directory.

2. Save the file as **IslandSALESOrderEntryMTS.app**, overwriting the existing file in the Tutorial directory.

3. In the Outline under Libraries, comment out **IslandSALESInfoSVR 1.0 Type Library**.

4. Select **Tools**, **ActiveX Explorer**. The ActiveX Explorer Library Selection dialog box opens.

5. Click **Browse**, select the **IslandSALESInfoSVRMTS.tlb** file located in the Tutorial directory, and click **Open**.

6. A message box is displayed asking whether you would like to register the Type Library. Click **Yes**.

7. If the Interfaces are not displayed, right-click in the Classes window pane and select **Show Interfaces** from the pop-up menu.

8. Right-click in the **Classes** window pane and select **Check All Shown** from the pop-up menu.

9. Right-click in the **Classes** window pane and select **Generate Full** from the pop-up menu.

10. Close the ActiveX Explorer.

11. Open the main TD window and select the **Outline** view. Open **Global Declarations**, **Class Definitions** in the IslandSALESOrderEntryMTS.app application, and rename the following COM Proxy Classes:

    Rename *from*:

    ```
    COM Proxy Class:   ISOInfoServer
    !
    COM Proxy Class:   ISOProductServer
    !
    COM Proxy Class:   ISOARPostServer
    !
    COM Proxy Class:   ISOSalesServer
    !
    COM Proxy Class:   ISOInvoiceServer
    !
    COM Proxy Class:   ISOListOrders
    !
    COM Proxy Class:   ISOListLineItems
    ```

    Rename *to*:

    ```
    COM Proxy Class:   ISOInfoServerMTS
    !
    COM Proxy Class:   ISOProductServerMTS
    !
    COM Proxy Class:   ISOARPostServerMTS
    !
    COM Proxy Class:   ISOSalesServerMTS
    !
    COM Proxy Class:   ISOInvoiceServerMTS
    !
    COM Proxy Class:   ISOListOrdersMTS
    !
    COM Proxy Class:   ISOListLineItemsMTS
    ```

12. Because you have now included the new Type Library, you must change some of the references in the preceding COM Proxy Classes. These COM Proxy Classes make it possible to handle events generated by

the Interfaces on the COM+ server. Each is derived from the COM Proxy Class included as part of the process of generating the Type Library with ActiveX Explorer.

Open **COM Proxy Class: ISOInfoServerMTS**, **Derived From** in the Outline. Change the class reference.

Change *from*:

```
Class: IslandSALESInfoSVR_ISOInfoServer
```

Change *to*:

```
Class: IslandSALESInfoSVRMTS_ISOInfoServerMTS
```

Make the same change to the references of each of the other COM Proxy Classes.

13. Open **Global Declarations**, **Variables** and change the names of the references for the variables.

Change *from*:

```
ISOInfoServer: comIslandINFO
!
ISOProductServer: comIslandPRODUCT
!
ISOARPostServer: comIslandARPOST
!
ISOSalesServer: comIslandSALES
!
ISOInvoiceServer: comIslandINVOICE
!
ISOListOrders: IslandORDERS
!
ISOListLineItems: IslandINVOICEITEMS
```

Change *to*:

```
ISOInfoServerMTS: comIslandINFO
!
ISOProductServerMTS: comIslandPRODUCT
!
ISOARPostServerMTS: comIslandARPOST
!
ISOSalesServerMTS: comIslandSALES
!
ISOInvoiceServerMTS: comIslandINVOICE
!
ISOListOrdersMTS: IslandORDERS
!
ISOListLineItemsMTS: IslandINVOICEITEMS
```

14. Select **Project**, **Build Setting**.

15. Change the Target Name to **IslandSALESOrderEntryMTS.exe** and click **OK**.

16. Save and then build the application. A message box is displayed stating that the build was successful.

17. Confirm that the COM+enabled COM server and client work correctly by selecting **Debug**, **Go**. Walk through the various dialogs of the application to ensure that it still functions normally.

You have now changed the tutorial COM client application so that it can use the COM+ Server. The next step is to install the COM+ Server in the COM+ environment as a new COM+ server application. For more information, see "Installing the Tutorial COM+ Server in COM+". The next section provides some information on how to create an ASP-based COM client that can use the COM+ Server.

Creating an ASP Client for a COM Server

COM servers can be used by any client that conforms to the COM specification, regardless of the programming language it was developed in. Included with the COM tutorial files is a set of ASP pages that replicate much of the functionality of IslandSALESOrderEntry.exe. These samples demonstrate how to use the functionality provided by a TD-developed COM server from an ASP page.

When installed by the TD installation CD, the ASP pages are copied to \wwwroot\GUPTA\Tutorial if the install machine has an Internet Server. If the install machine does not have an Internet Server, the files are installed to \GUPTA\Tutorial\ASP.

System Requirements

The machine running the COM server must meet the following system requirements:

- Microsoft Windows NT 4.0, 2000, or XP

  If you are running Windows NT 4.0, you must install Windows NT 4.0 Option Pack 4.

- Web Server that can handle ASP pages; for example, Microsoft Internet Information Server

- Installation of TD that includes the COM tutorial sample applications

The machine running the ASP client must meet the following system requirements:

- Microsoft Internet Explorer 5.01 or higher

- Forms 2.0 Control Security Patch, fm2paste.exe (available from the Microsoft website in the downloads section)

This Patch fixes a bug in how the ASP pages are displayed in Internet Explorer.

Running the Tutorial ASP Pages

To run the tutorial ASP-based client, follow these steps:

1. Install your Web server; for example, Microsoft IIS, which is installed as part of Windows NT 4.0 Option Pack 4 and included in 2000 and XP. You can use any Windows-based Web server that can handle ASP pages.

2. Install the full version of Gupta Team Developer 2.1 or higher.

   As a part of the TD installation, IslandSALESInfoSVRMTS.dll (the COM+ enabled tutorial COM server) is installed and registered, and the tutorial ASP pages are copied to \wwwroot\GUPTA\Tutorial (this directory varies according to the type of Web Server you are running).

3. Add the COM objects as components of a new COM+ package (see "Installing the Tutorial COM+ Server in COM+").

4. Open **testparameters.app** (located in \GUPTA\Tutorial) in SQLWindows.

   Build and register the server.

5. Launch your Web Server and COM+.

6. Using Microsoft Internet Explorer, open the following URL: http://webservername/GUPTA/Tutorial/

The default web page (default.htm) for the ASP tutorial pages is displayed and provides you with three options:

- Ping Server – Testparameters COM Server

  Clicking this hyperlink takes you to an ASP page that attempts to connect to the COM server on the TD machine. This ASP page is designed to test whether a COM server on the TD machine is accessible or not. The ASP page indicates whether it is successful or not.

- Ping Database – IslandSALESInfoSVR with COM+

  Clicking this hyperlink takes you to an ASP page that attempts to retrieve data from the ISLAND sample database using the COM server through COM+. It indicates whether it is successful or not.

- Island SALES Info Application

  Clicking this hyperlink takes you to the ASP version of the IslandSALESOrderEntry client. The tutorial ASP pages that are provided with TD replicate some of the functionality of the Windows/ CWD version.

Overview of the Tutorial ASP Pages

This section describes each of the ASP pages provided with TD. Like HTML pages, you can open ASP pages in any text editor; Notepad, for example. These pages include numerous comments interspersed throughout the code to help you to better understand how they function.

The following tutorial ASP pages are included with TD:

- CreateNewInvoice.asp

  Creates a new invoice. This ASP page is used by FormPlaceOrder.asp. FormPlaceOrder.asp calls CreateNewInvoice.asp when the user completes the order and clicks Submit Order.

- FormListOrder.asp

  Making this ASP page functional is left as an exercise for the reader. This page is intended to replicate the List Order window in the IslandSALESOrderEntry client application. It should list the invoices associated with the company selected on IslandSalesinfo.asp.

- FormNewInvoice.asp

  This ASP page is left as an exercise for the reader. This page replicates the ISLAND SALES ORDER INVOICE window in the IslandSALESOrderEntry.exe client application. It displays the completed invoice submitted from FormPlaceOrder.asp.

- FormPlaceOrder.asp

  Replicates the Island SALES ORDER ENTRY window in the IslandSALESOrderEntry.exe client application. You can create a new sales invoice for the company selected on IslandSalesinfo.asp.

- GetCompanyIDs.asp

  Retrieves the CompanyIDs from the ISLAND database.

- GetCustomerInfo.asp

  Retrieves company information from the ISLAND database based on the CompanyID passed in from PingDatabase.asp.

- IslandSalesinfo.asp

This ASP page replicates the opening screen of the IslandSALESOrderEntry.exe client application. It allows you to select a company, list contact information on that company, and then either to list the invoices currently associated with that company or to create a new invoice.

- PingDatabase.asp

    Tests whether you can access data in the ISLAND database through the IslandSALESInfoSVR.dll COM server in the COM+ environment. Also makes calls to GetCustomerInfo.asp.

- PingServer.asp

    This ASP page tests whether you can access a COM object on your TD machine from your web browser.

Calling a COM Server from an ASP Page

This section provides information on how to make a function call to the tutorial COM server from an ASP page.

GetCustomerInfo.asp includes code that creates an instance of a COM object and then calls a function from that object. The following is a code sample from GetCustomerInfo.asp that instantiates the ISOInfoServerMTS COM object:

```
'Instantiate SalesInfo Object
```

'response.write "Invoking MTS SAL COM Server from the ASP SalesInfo page!
"

set objSalesServer = Server.CreateObject("IslandSALESInfoSVRMTS.ISOInfoServerMTS.1")

! If the server fails to create, then execution will stop on the preceding line

'response.write "Created ISLAND Sales Order Information Server

- With MTS
"

In the following code sample, the function variables for GetCustomerInfo are set to null and then the function (GetCustomerInfo through the objSalesServer object) is called.

Note: Because all variables in ASP are of data type variant, you should also set to null each data type before calling any COM function with parameters; for example, sString = "", nNumber = 0, and so on.

```
sCompanyName = "" sAddress = "" sCity = ""
sState = "" sZip = "" sCountry = ""
sContactName = ""
'Call Object
vGetCustomerInfo = objSalesServer.GetCustomerInfo( nCompanyID, sCompanyName,
sAddress, sCity,
sState, sZip, sCountry, sContactName)
```

The input parameters (nCompanyID) and the output parameters (sCompanyName, sAddress, and so on) are defined in the COM functions. The nCompanyID input parameter is of data type number. The remaining parameters in this function are receive parameters. Because you declared the variants as string null, the receive values return value is a string.

Chapter 6 – Managing Teams and Objects

This chapter discusses team programming. Team programming helps you manage projects with many programmers, including sharing code and standardizing programs.

This chapter explains how to bring the application (from the \Samples directory) into the Team Object Repository, and how a programmer manages code through check-in and check-out and other version control features. This is accomplished with Team Object Manager, the team programming component.

Managing Teams and Objects

The Gupta multi-programmer development environment enables teams of programmers to work together efficiently to develop client-server applications.

- Entire teams of programmers can use Team Object Manager to share specialization of labor, minimize duplication of effort, and establish consistency within and across projects.

- Managers can track the application development process and produce impact analysis reports.

- Expert programmers can capture their application knowledge and share it with other programmers without exposing them to the complexities of expert application design.

- All programmers can share and reuse portions of a GUPTA SQLWindows application throughout its life cycle.

Team Object Repository

Pivotal to team programming with Gupta is the Team Object Repository. The Repository is a centralized, multi-user database that contains an extended storage area of information about the application database. The Repository also holds project-related information and files, including a current copy of each project file.

Requirements

Team Object Manager must be installed on your machine. For instructions, see the Gupta Team Developer Setup.

The Team Object Repository must be installed. From the Start menu on your machine, select GUPTA\Team Developer\Setup Repository Wizard. The Wizard helps you specify configuration information and install the TEAMOBJM repository. For the following tutorial, specify QuickInstall, specify TEAMOBJM as your repository, and then accept the defaults to install the repository.

Create a Project

Starting Team Object Manager

1. Select Team Object Manager from the Tools menu in SQLWindows.

   The Team Object Manager Login dialog prompts you for a Repository, a user name, and a password.

2. Type the Repository, User Name, and Password into the appropriate fields. Use **TEAMOBJM**, **demo**, and **demo**, respectively.

3. Click **OK**.

Note: Team Object Manager might tell you that you are missing the #STARTER project. If so, follow the defaults in the dialogs that appear to include the #STARTER project.

Creating a New Project

The Team Object Manager desktop appears as follows.
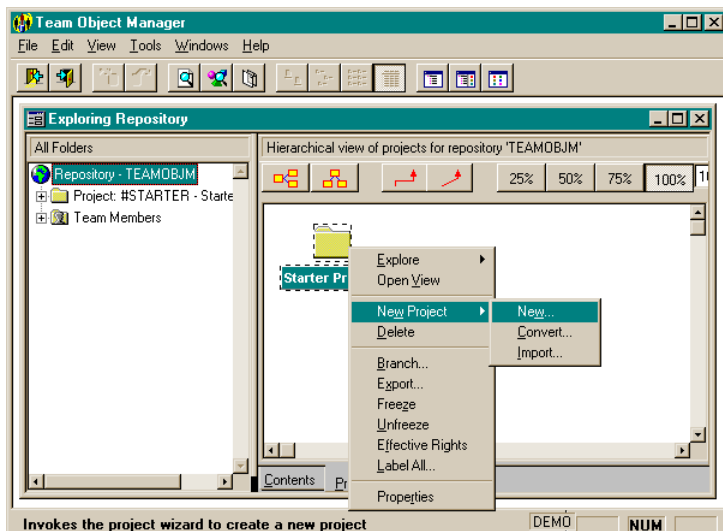
1. Right mouse click on the **Starter Project** icon. The project context menu appears.

2. Select the **New Project / New** option to open the Project Wizard. This will guide you through the steps required to create your new project. The initial dialog in the Project Wizard appears.

3. Click **Next** to continue.
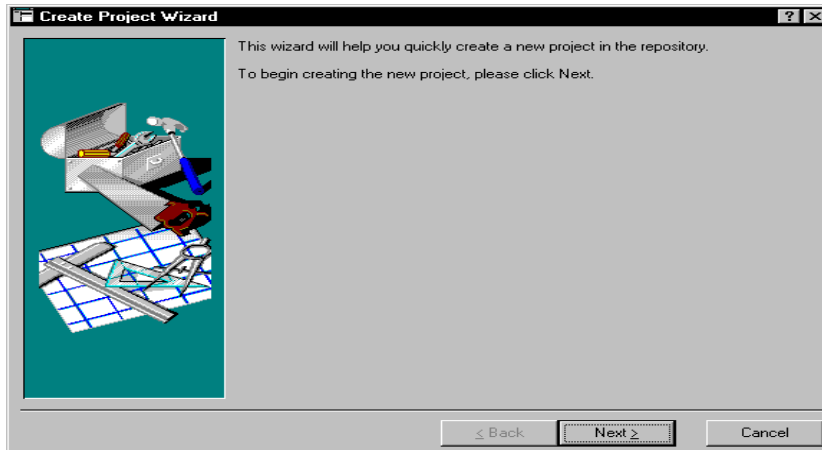
4. The next dialog gives you two options. To use as many pre-set options as possible going forward, select the **Express** radio button and click **Next**.
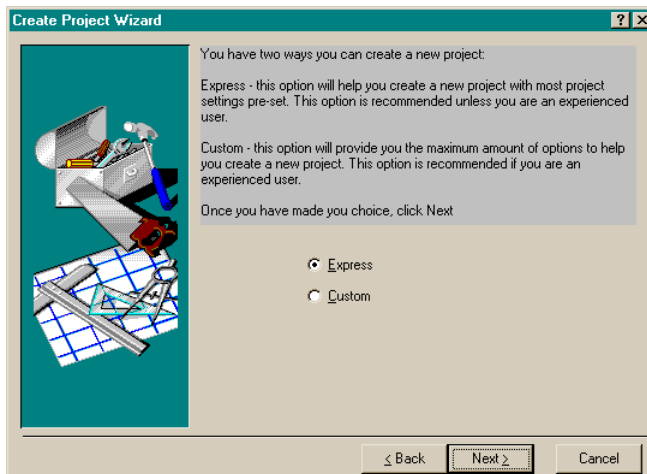
5. Type the project identity into the **Code** field. Name this project **MOLOKAI**. Type a more informative name for the project into the **Name** field. Type any description you want into the Description field, a multiline text field and click **Next**.
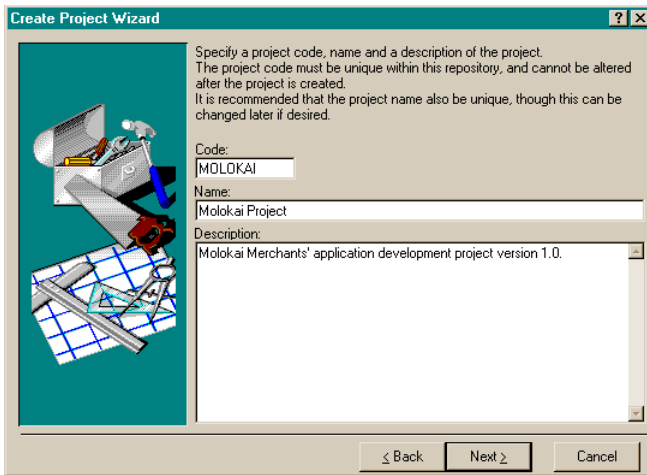
6. In the screen that appears, accept the default in the Project Directory field and click **Next**.
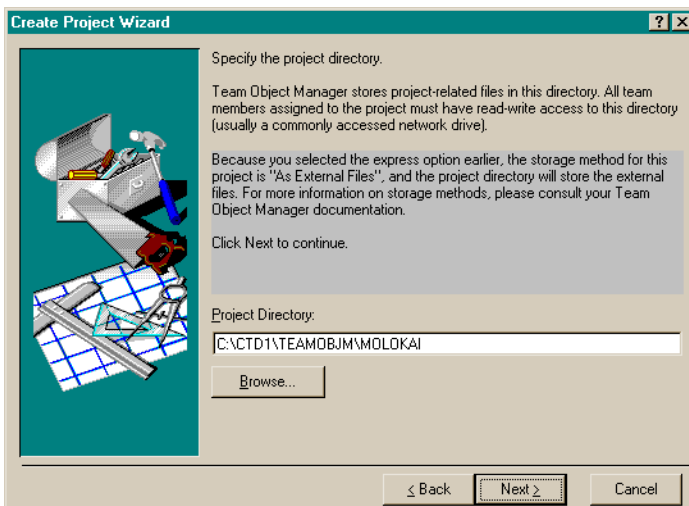
7. Click **Finish**.
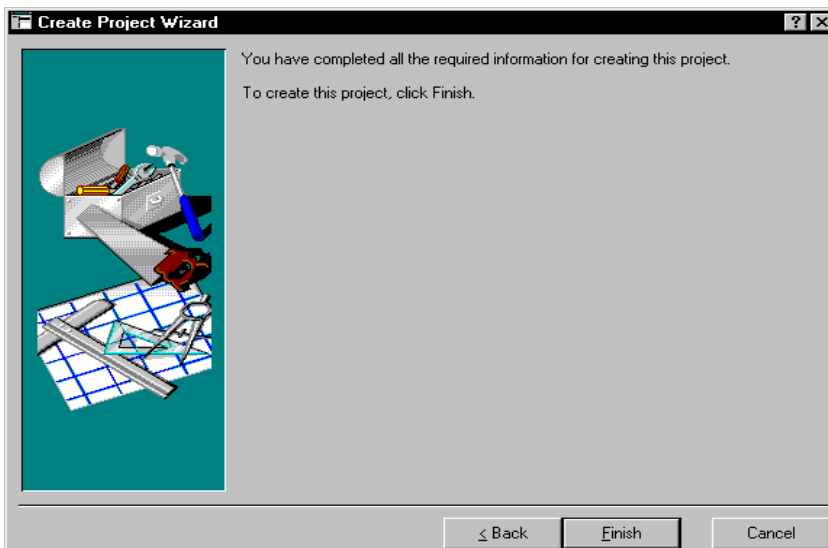
A dialog box appears to let you know the project was successfully created.
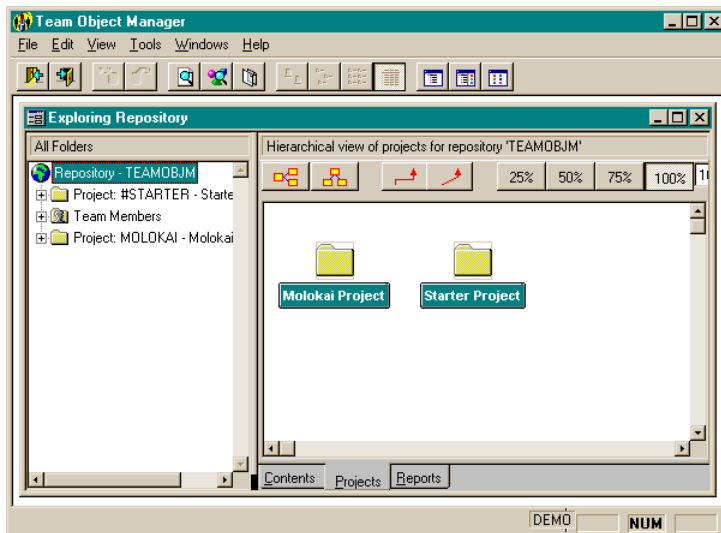
8. Click **OK**.


Bring an Application into the Repository

Add a SQLWindows application to the MOLOKAI project.

Adding qckfinal.app to the Project

1. Click the '+' symbol next to the MOLOKAI project. The project opens to show the objects it contains.
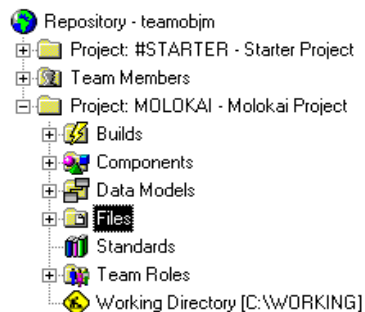
2. Right-click the **Files** icon to display the Files context menu.

3. Select the **Add** option in the Files context menu. This enables you to insert existing files into your project.
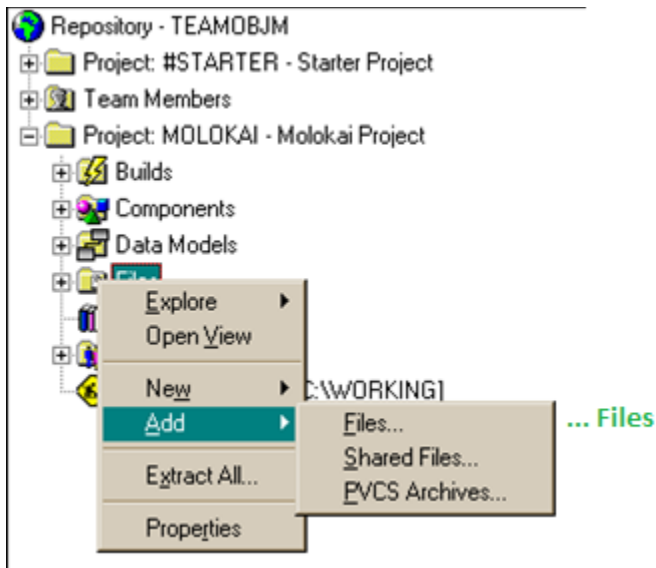
4. Click the **Files** option to bring up the browse dialog.

    Select **qckfinal.app** (in the \Samples directory).

    Click **Add Files** to insert this file into the project.

5. Repeat steps 2 - 4 to insert the following files (usually in the \GUPTA directory): FORWARD.BMP, LASTREC.BMP, NEWREC.BMP, NEXTREC.BMP, PREVREC.BMP.

Note: Hold down the **Ctrl** key as you select these files to add them all at once.

6. Click the **qckfinal.app** item in the left pane (that was just adde to the project).

    The Team Object Manager desktop looks similar to the following:
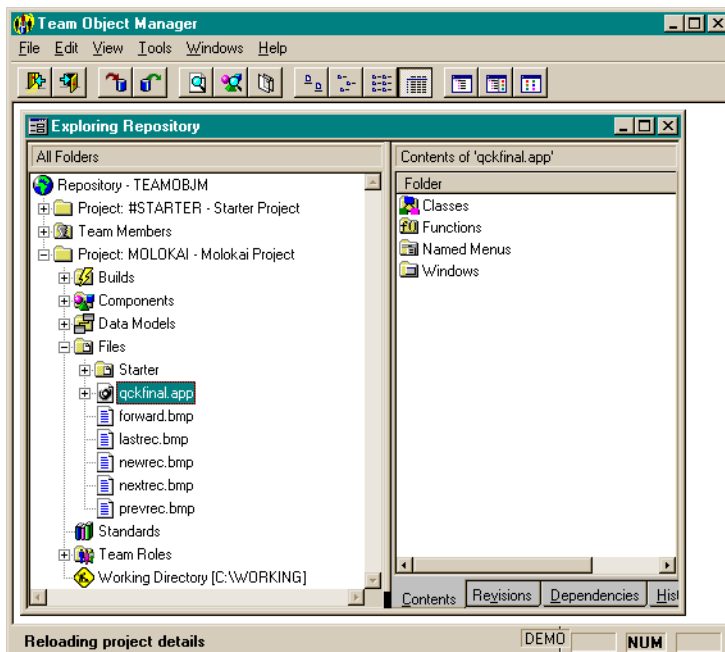
Checking Out a File from the Repository

To check out the **qckfinal.app** from the Repository to your local workstation, follow these steps:

1. Click the **qckfinal.app** item. Eight tabs are displayed in the view window.

2. Right-click the **qckfinal.app** icon to bring up the context menu. Select the **Scan, Scan for Dependencies** option. This links the application file with the bitmaps you included in the project.
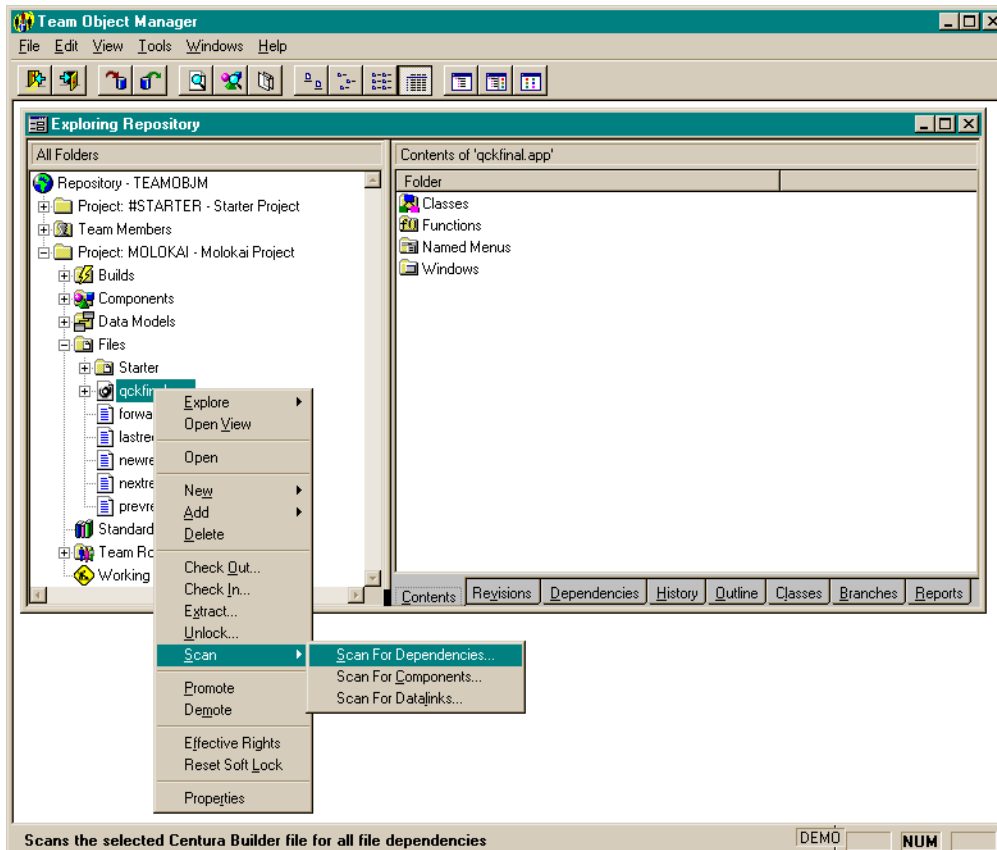

A warning box is displayed to let you know that the application file requires other files to be included in the project which are not present.
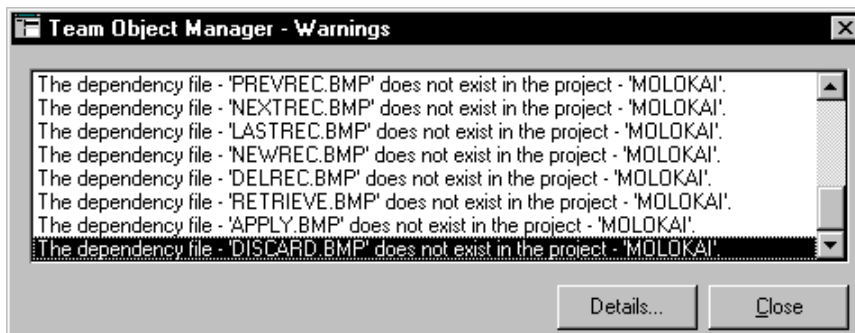
3. Click **Close**.

4. When the scan for dependencies finishes, select the **Scan for Components** option from the same menu. This updates the Repository with information about all the component parts (functions, windows, classes and named menus) that are included in your project.

5. Select the **Check Out...** option. This brings up the Check out file dialog.

6. Click **OK** to start the check out process.
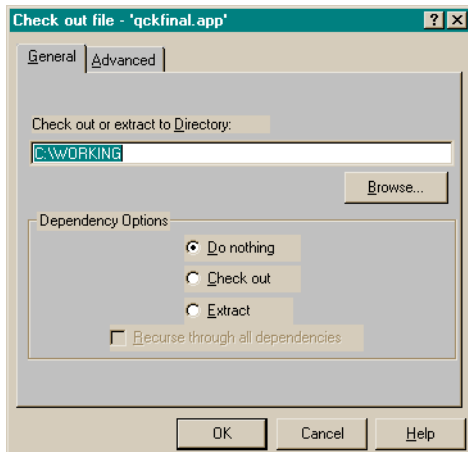
Note: If the Checkout directory does not exist, you will be asked if you want to create it. Click **Yes**.

Editing the Checked-Out File

Once you check out a file, your username appears in square brackets beside the file name in the left pane. You can now edit the checked out file.

1. Select qckfinal.app and click the right mouse button to bring up the context menu.

2. Select the **Open** option to edit qckfinal.app.

While the file is checked out, you can edit it, integrate it with other files, and continue your development cycle. The next exercise shows how to make changes to qckfinal.app and use the Diff/Merge tool to analyze the before and after versions of the file.

You were able to check out the file qckfinal.app from the MOLOKAI project to your local workstation. Once checked out, the file is editable in SQLWindows.

Diff/Merge Tool

Diff/Merge compares two SQLWindows files and reports on the differences. To use this tool, follow these steps.

Change qckfinal.app and Save as qck2.app

1. Within Team Object Manager, select **qckfinal.app** from the Files object in the left pane.

   Use the right mouse button to bring up the context menu.

   Select the **Open** option to edit **qckfinal.app** with SQLWindows.

2. Using SQLWindows, add a new form window to the application by right-clicking **qckfinal.app** in the left pane to open the context menu.

   Select **New / Form Window** from the menu.
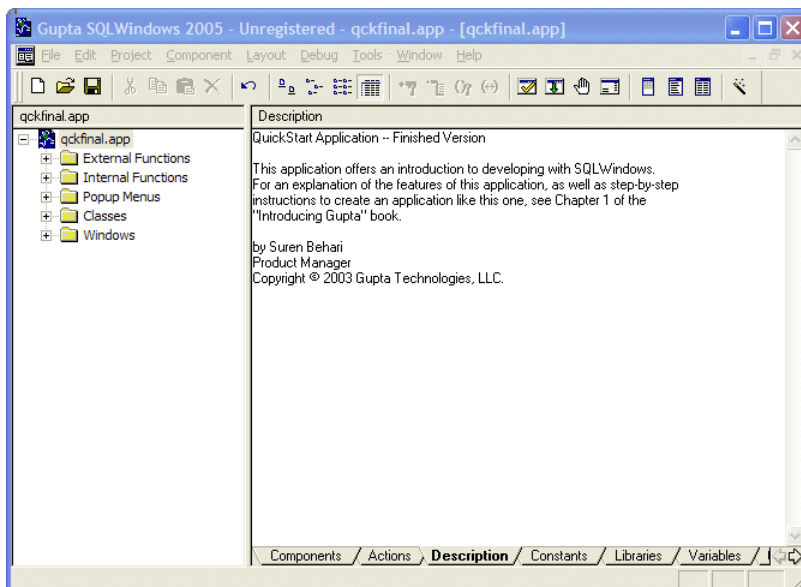
3. Call the new form window **frmMyForm**.
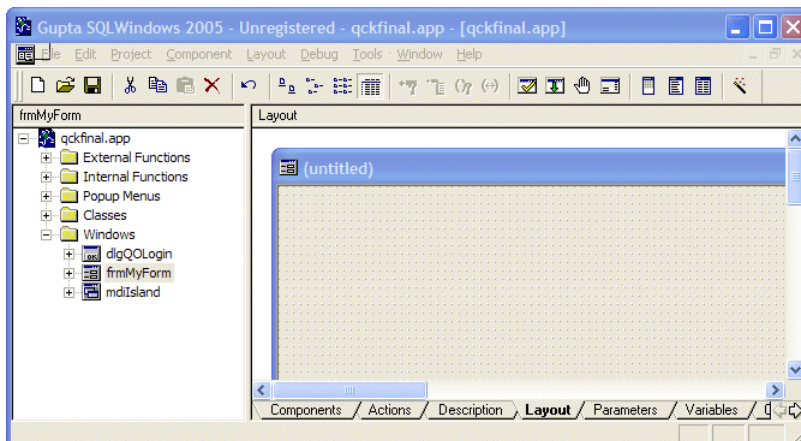
4. Click **mdiIsland** in the left pane.

   Delete the bitmap object from the mdiIsland form.

5. Select the Island Outfitter Background text object and move it to the left.


6. Save your changes in a new file called **QCK2.APP** by opening the **File** menu and selecting the **Save As** option.

7. Close SQLWindows and return to Team Object Manager.

Compare the Two Files

1. Within Team Object Manager, select the **Tools** menu and select the **Diff/Merge** option.
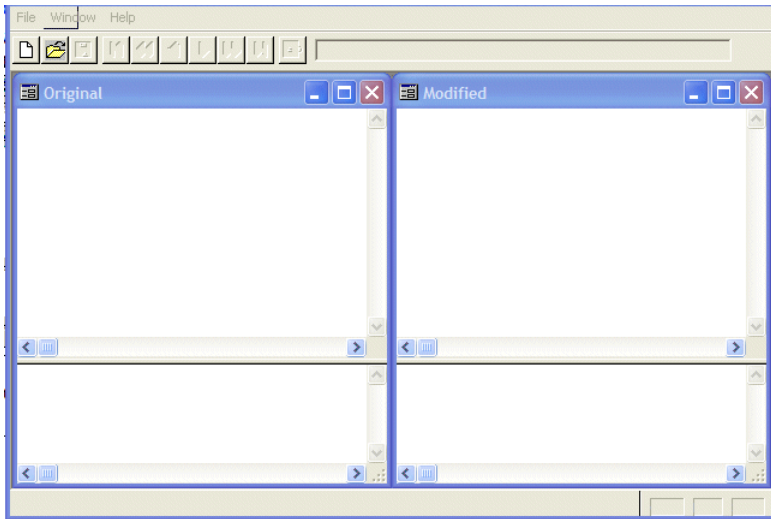
2. Start a new file comparison by clicking the [icon] button in the tool bar.

3. Specify the names of the two files to be compared, **qckfinal.app** and **qck2.app**.
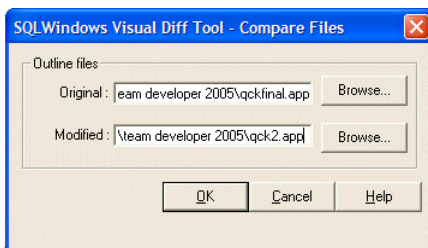

4. Click **OK** to start the file comparison.

When the comparison completes, the Diff/ Merge tool shows the changes in the original and modified files. The original file contents are shown on the left side, the new file contents on the right side. The changes are shown as deletions, modifications, and additions.
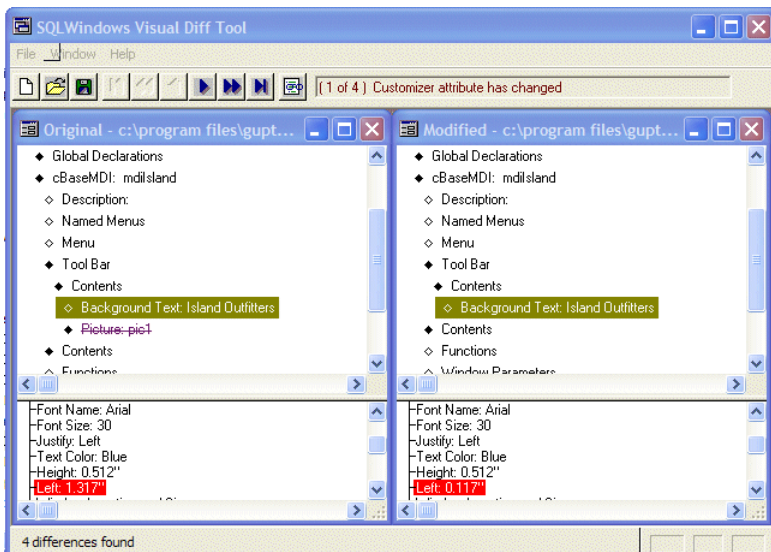
5. Press the **F4** key to open the Go To Difference dialog. This dialog contains an ordered listing of each difference between the original and modified files. By clicking on one of the list entries you can go straight to the place where the change was made.
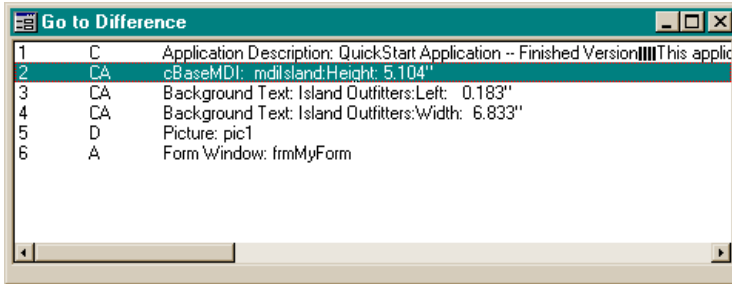

Each variation between the two files is listed in the dialog together with a code that identifies the type of difference; for example, C = Changed Text, D = Deletion, CA = Changed Attribute, M = Move, A = Add.

6. Close the Diff/Merge tool. Do not save the analysis report contents.

Checking a File Back into the Repository

1. Check the **qckfinal.app** file into the Repository.

2. Return to Team Object Manager. Right-click **qckfinal.app** and select the **Check In** option. This brings up the Check In file dialog.
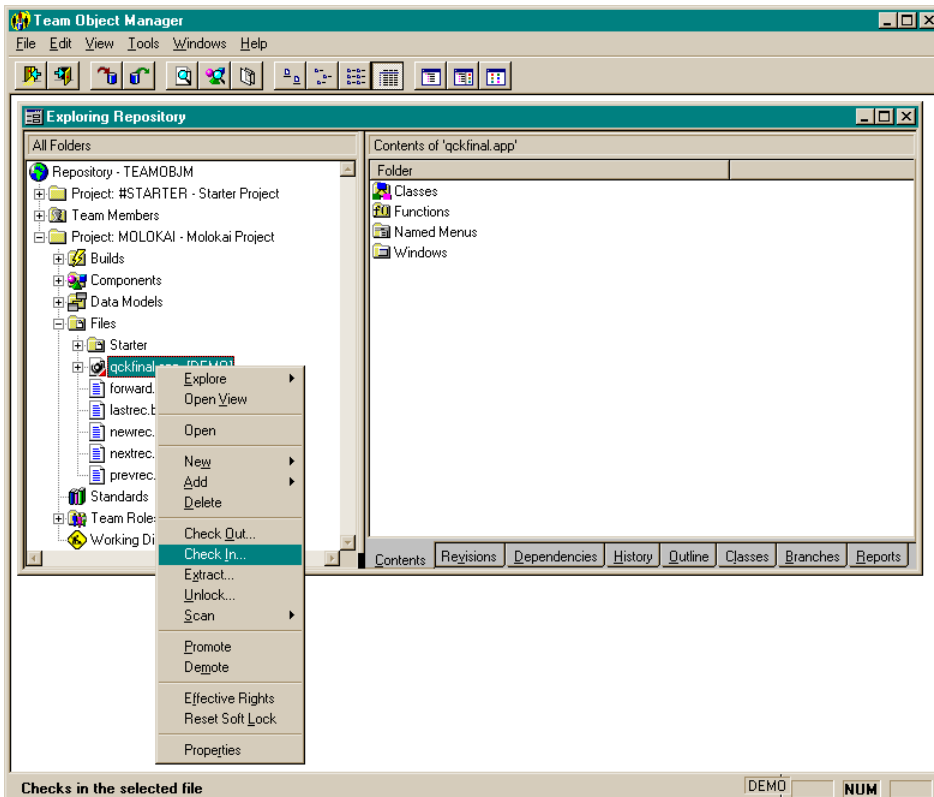
Note: Because you saved your changes to another file, a warning appears to let you know that the file was not modified and asks if you would like to unlock the file. Click **No**.
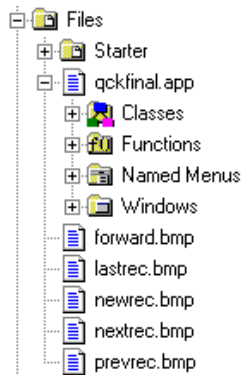
3. Type an explanatory comment into the Notes field and click **OK**.

The file is checked in and now there are two revisions of the file. The earlier one is stored as a delta file for space efficiency.

Short Tour of the Team Object Manager Interface

1. Click the **+** symbol beside the qckfinal.app in the tree view (left pane).

The four objects below the .app file represent the types of objects the application contains.

2. Open the **Classes** and **Windows** objects by clicking + to see what objects are present.
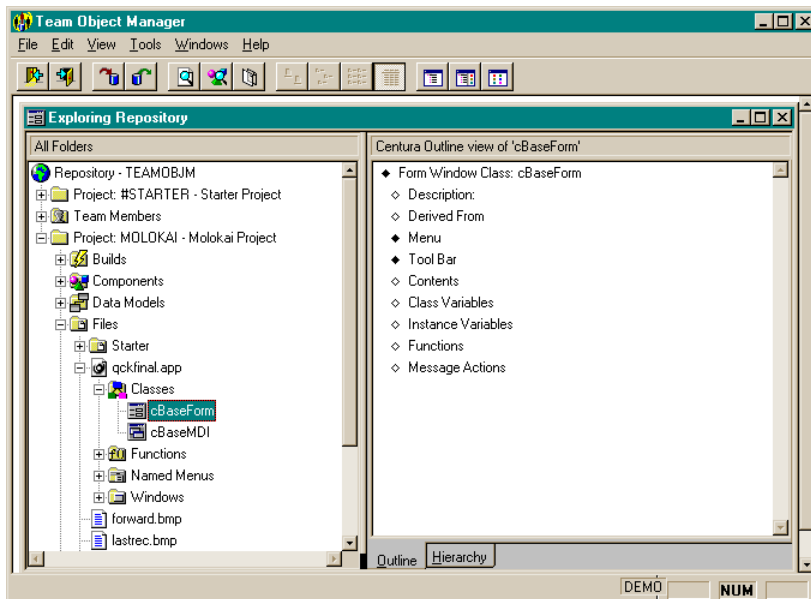
3. Click the **+** symbol next to the Team Roles object. This gives you a view of the different user classifications that Team Object Manager provides as a default.

114

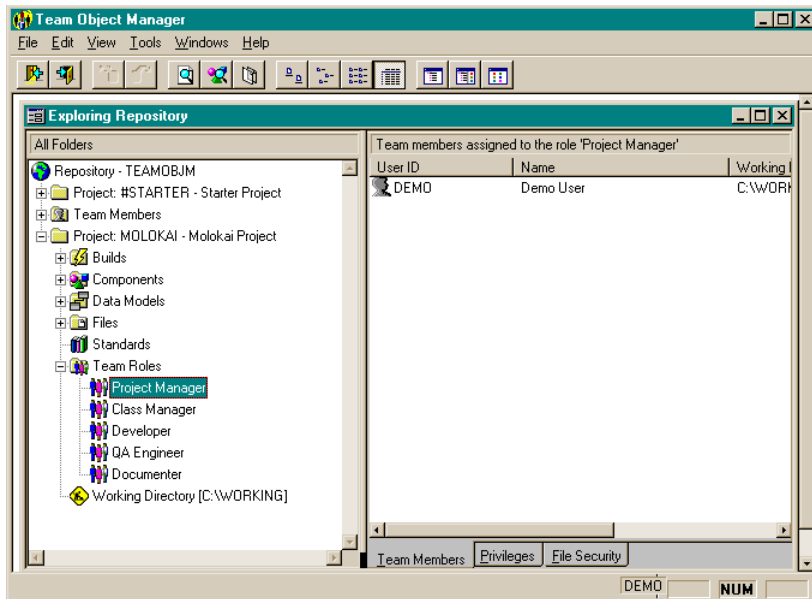Team Object Manager Team Roles are fully configurable, offering maximum flexibility when structuring your projects.

Team Object Manager provides an intuitive, project-oriented user interface that makes it possible for you to check files in and out of the Team Object Repository. It has other powerful features that are fully detailed in the *Managing Teams and Objects with GUPTA* manual supplied with the GUPTA Bookcase or found in the online book collection.

Chapter 7 – More Information

Team Developer is a full-featured, robust development environment. To help you learn about and use all the components, the entire documentation suite is available on your CD in the online book collection. To view the books, select GUPTA Books Online from the GUPTA program group.

Gupta Books Online

GUPTA Books Online includes manuals on the following topics:

- Application Development
- SQLBase
- Connectivity

Application Development

Book	Contents
Introducing GUPTA Team Developer	How to use GUPTA Team Developer
New Features in Team Developer	Features that are new to this version of Team Developer
Developing with SQLWindows	Description and examples of how to use the interface features
SQLWindows Function Reference	Syntax, descriptions, and examples of GUPTA functions
Building Web Applications with GUPTA	How to write, deploy, and manage Web applications
Extending the GUPTA Development Environment	Discovering the SAL and C++ functions in the CDK that help you extend your applications
Connecting Gupta Objects to Databases	How to use Gupta's routers to connect to non-Gupta databases
Business Reporting	How to create reports from GUPTA SQLWindows using the Report Builder component
Using and Extending QuickObjects	How to create complex and sophisticated applications using QuickObjects
Managing Teams and Objects with GUPTA Team Developer	Understanding conceptual information and how to use Team Object Manager, the multi- programmer developer tool
Localizing and Customizing GUPTA Applications	How to translate applications into other languages

SQLBase

Book	Contents
SQLBase Starter Guide	Installing and configuring GUPTA's SQLBase
SQLBase Guide to New Features	Descriptions of features that are new to this particular version of SQLBase
SQLBase Database Administrator's Guide	Managing your SQLBase database, and design databases and applications
SQLBase Application Programming Interface Reference	SQLBase SQL/API, a set of functions you can call to access a SQL database
SQLBase SQL Language Reference	Entering SQL commands in a Windows environment and performing database management functions
SQLBase SQLTalk Command Reference	SQLTalk commands
Connecting to SQLBase	Using SQLBase drivers and data providers to connect any client application in any language to SQLBase
SQLConsole Guide	Using SQLConsole to simplify database administration, manage database objects, monitor performance, and automate database maintenance
SQLBase Advanced Topics Guide	SQLBase advanced topics, including database design, SQLBase internals, and SQLBase query optimizer

Connectivity

Book	Contents
Connecting GUPTA Objects to Databases	Connecting your GUPTA applications to one or more relational databases, such as Oracle, Microsoft SQL Server, System 11.x, Informix, and others
Connecting to SQLBase	Using SQLBase drivers and providers for JDBC, ODBC, OLE DB, and .NET, with any compatible application development tool
SQLHost Installation and Operation Guide	Installing and configuring SQLHost software on a workstation and a SQLGateway machine
SQLHost Application Services Developer's Guide	Using SQLHost/Application Services (SQLHost/AS) to create server applications
SQLHost Client Developer's Guide	Designing and developing client applications for SQLHost

Appendix – Installing the Tutorial COM/MTS Server in MTS

You can install the tutorial COM/MTS server as a package in MTS and Windows NT. For detailed instructions on how to use MTS, consult the Microsoft documentation.

The following steps describe the menus as they are specified in Windows NT. The Windows 2000 menus are somewhat different, and are described in chapter 5.

To install the COM/MTS server as a package in MTS, follow these steps:

1. Close SQLWindows, SQLBase, and any other application that might be using the tutorial COM server.

2. Select **Start** > **Programs** > **Windows NT 4.0 Option Pack** > **Microsoft Transaction Server** > **Transaction Server Explorer**.

   Expand the MTS tree: Microsoft Transaction Server, Computers, My Computer, Packages Installed.

   Select **Packages Installed**.

   Right-click and select **New**, **Package**.

   In the Package Wizard, click **Create an Empty Package,** name it **IslandMTS**, and click **Next**.

3. Use the default settings in the Set Package Identity dialog box and click **Finish**.

   Expand the new MTS Island package in the tree view.

   Select **Components**.

   Right-click and select **New**, **Component**. The Component Wizard is displayed.

4. Click **Import components** that are already registered.

5. Select the following components:

   ‣ IslandSALESInfoSVRMTS.ISOARPostServerMTS
   ‣ IslandSALESInfoSVRMTS.ISOInfoServerMTS
   ‣ IslandSALESInfoSVRMTS.ISOProductServerMTS
   ‣ IslandSALESInfoSVRMTS.ISOSalesServerMTS
   ‣ IslandSALESInfoSVRMTS.ISOInvoiceServerMTS
   ‣ IslandSALESInfoSVRMTS.ISOListOrdersMTS
   ‣ IslandSALESInfoSVRMTS.ISOListLineItemsMTS

   By checking the Details checkbox, you can confirm that the components you select are coming from the correct DLL. The default is \GUPTA\Tutorial\IslandSALESInfoSVRMTS.DLL.

6. Click **Finish**.
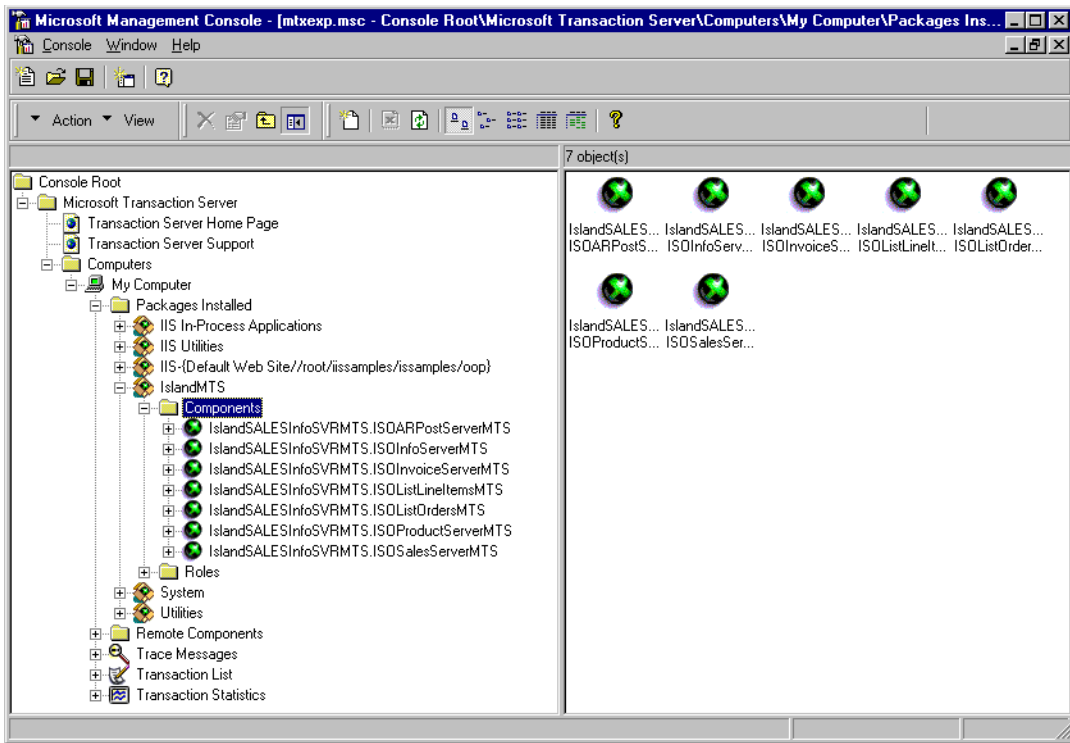
The MTS Server objects are displayed in the right window pane.

Now that the COM/MTS server components are installed in MTS, MTS can act as a broker for the objects when called by the COM/MTS client application.

7. Open **IslandSALESOrderEntryMTS.app** and run the application in debug mode (or launch IslandSALESOrderEntryMTS.exe).

If the Microsoft Management Console is open while the client is running, the ball graphic spins for each object that is currently being used by the client.

Each time you revise an COM/MTS server, you should delete the old MTS server package and reinstall the revised COM server in a new MTS server package.