CHAPTER 6: REPORTING

Objectives

The objectives are:

- Using report event triggers.
- Using special report functions.
- Creating processing-only reports.

Introduction

The seminar module thus far includes the master tables and forms, a means to create new seminar registrations, and routines to post the registrations. These features are integrated into the standard application so they can be accessed from the Main Menu. The next step is to create reports for the module. This will allow reports containing seminar registration information to be generated.

Microsoft Official Training Materials for Microsoft Dynamics ™
Your use of this content is subject to your current services agreement

6-1

Prerequisite Information

Before analyzing and implementing the report functionality covered in this chapter, there are some important concepts that should be reviewed.

Report Triggers

There are many events that happen when running a report. As a result, reports support a number of event triggers. The following report elements all have triggers:

- The report itself
- The data items
- The sections
- The request form
- The controls

It is important to understand the order in which some of the more frequently used triggers fire. The following list traces the firing order when a report runs.

1. When the user initiates the running of the report, the **OnInitReport** trigger is called. This trigger can perform processing that is necessary before any part of the report can run. It can also stop the report.

2. If the **OnInitReport** does not end the processing of the report, the request form for the report runs, if defined. The user can choose to cancel the report from the request form.

3. If the user continues, the **OnPreReport** trigger is called. At this point, no data has been processed. Like the OnInitReport trigger, the OnPreReport trigger can terminate report processing.

4. Provided that the processing of the report is not ended in the **OnPreReport** trigger, the first data item is processed.

5. Immediately before the first record is retrieved, the **OnPreDataItem** trigger is called. Likewise, the **OnPostDataItem** trigger is called after the last record has been processed.

6. Between these two triggers, the records of the data item are processed. Processing a record means executing the record triggers and outputting sections. C/SIDE also determines whether the current record should cause outputting of a special section: header, footer, group header or group footer.

7. If there is an indented data item, a data item run is initiated for this data item. Data items can be nested ten levels deep.

8. When there are no more records to be processed in a data item, control returns to the point from which the processing was initiated. For an indented data item this is the next record of the data item on the next higher level. If the data item is already on the highest level (indentation is zero), control returns to the report.

9. After the first data item has been processed, the next data item – if one exists – is processed in the same way.

10. When there are no more data items, the **OnPostReport** trigger is called. Use this trigger to do any post processing that is necessary, for example, cleaning up by removing temporary files.

Report Functions

Certain functions can only be used in reports. These functions can be useful for complex reports.

CurrReport.SKIP: Use this function to skip the current record of the current data item. If a record is skipped, it is not included in totals and it is not printed. Skipping a record in a report is much slower than never reading it at all, so use filters as much as possible.

CurrReport.BREAK: Use this function to skip the rest of the processing of the data item currently being processing. The report resumes processing the next data item. All data items indented under the one that caused the break are also skipped.

CurrReport.QUIT: This function skips the rest of the report. It is not an error, however. It is a normal ending for a report.

CurrReport.PAGENO: Use this function to return the current page number of a report and/or to set a new page number.

CurrReport.CREATETOTALS: Use this function to maintain totals for a variable in the same way as totals are maintained for fields by using the TotalFields property. This function must be used in the **OnPreDataItem** trigger of the data item in the sections where the totals are displayed.

CurrReport.TOTALSCAUSEDBY: Use this function to determine which field caused a break to occur. The return value is the field number of the field that the data item is grouped on that changed and caused a Group Header or Group Footer section to print. This function is usually used in the OnPreSection trigger of Group Header and Group Footer sections. This function must always be used when grouping more than one field for a single data item.

CurrReport.NEWPAGE: Use this function to force a page break when printing a report. This is usually found in the data item triggers.

CurrReport.PREVIEW: Use this function to determine whether a report prints in preview mode.

CurrReport.SHOWOUTPUT: Use this function to return the current setting of whether a section should be outputted or not, and to change this setting. This function should only be used in the OnPreSection trigger of a section. If TRUE is passed to the function, nothing changes and the section prints as normal. If FALSE is passed to the function, the section is not printed for this iteration of the data item.

Processing–Only Reports

A processing-only report is one that does not print but instead changes table data. Changing table data is not limited to processing-only reports; however, as printing reports can also change records, this section applies to those reports as well. It is possible to specify a report to be "Processing Only" by changing the **ProcessingOnly** property of the **Report** object. The report functions as it is supposed to (processing data items), but it does not print any sections.

When the **ProcessingOnly** property is set, the request form for the report changes slightly: the **Print** and **Preview** buttons are replaced with an **OK** button. The **Cancel** and **Help** buttons remain unchanged.

Once the **ProcessingOnly** property is set, use the following guidelines to write processing-only reports:

- Decide which tables are read – these are the data items.
- Most of the code will go into the **OnAfterGetRecord** trigger.
- Do not forget the INSERT or MODIFY functions.
- Use a dialog to show the user the progress, and allow the user to cancel the report.

There are a few advantages to using a report to process data rather than a codeunit. One is that the request form functionality that allows the user to select options and filters for data items is readily available in a report, but difficult to program in a codeunit. Also, using the features of the Report Designer ensures consistency. Finally, instead of writing code to open tables and to retrieve records, report data items can be used.

Reporting Lab Overview

Description

The client's functional requirements describe their reporting needs in the following way:

The customer requires that a number of reports and statistics be available. These include:

- A list of the participants registered for a seminar.
- A list of the participants that have successfully completed each seminar, for the purposes of providing certificates. (The certificates themselves will also be produced by the system.)
- The total costs for each seminar, broken down according to what is and what is not chargeable to the customer.
- Statistics for different time periods, for example, for a month, for last year, for this year, and up to the current date.

There are three main reports that can be created to fulfill these requirements:

- A participant list
- A certificate confirmation
- A processing-only report that posts invoices

Each of these reports is implemented in a lab in this chapter.

Participant List Reporting

Solution Analysis

The client's functional requirements require that a Participant List report be available. This is merely a list of participants that are enrolled for a given seminar. This report should be available from both the main **Seminar Menu** and the **Seminar Registration** form.

Solution Design

To implement this report, it is necessary to create:

- The report itself.
- The request form to set the parameters of the report.
- The controls to access the report from a form.
- A form from which seminar reports can be selected.

GUI Design

The **Seminar Report Selection** form showing in Figure 6-1 displays the available seminar reports.
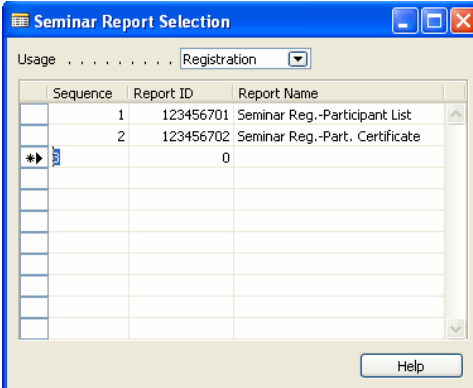

FIGURE 6–1: THE SEMINAR REPORT SELECTION FORM (123456723)

SEMINAR MENU: MODIFY THIS MENU BY ADDING THE FOLLOWING MENU ITEM TO THE REPORTS GROUP (UNDER ORDER PROCESSING):

Menu Type	Menu Name	Group	Comment
Item	Seminar Reg.-Participant List	Reports	Opens report 123456701 Seminar Reg.-Participant List.
Item	Seminar Reg.-Part. Certificate	Reports	Opens report 123456702 Seminar Reg.-Part. Certificate.

Add the following menu item to the Setup Group:

Menu Type	Menu Name	Group	Comments
Item	Report Selections	Setup	Opens form 123456723 Seminar Report Selection.

The **Seminar Registration** form should be modified as shown in Figure 6-2 by adding a **Print** command button from which to start the **Participant List** report.

FIGURE 6–2: THE SEMINAR REGISTRATION FORM (123456710)

Functional Design

The information for the report comes from the **Seminar Registration Header** and **Seminar Registration Line** tables.

When running this report, the user selects which **Seminar Registration Headers** are included. For each **Seminar Registration Header**, the program then prints information from each corresponding **Seminar Registration Line**.

Table Design

Implementation of the **Participant List** report will require that one new table be created, which will be called **Seminar Report Selections**. Also, the **Seminar Registration Header** table must be modified.Create one new table:

Lab 6.1 - Creating the Participant List Report

Begin development by creating the **Participant List** report and then create the means by which seminar reports can be selected from a form.

1. First, add a field to the **Seminar Registration Header** table (123456710) as follows:

No.	Field Name	Type	Length	Comment
40	No. Printed	Integer		Must not be editable.

In the next tasks, create a codeunit to increment the **No. Printed** field just added to the **Seminar Registration Header** table.

2. Create the **Seminar Registration-Printed** codeunit (123456702). Set the property to specify **Seminar Registration Header** as the source table for this codeunit.

3. Enter code in the appropriate trigger so that when the program runs this codeunit, it finds the **Seminar Registration Header** record, increases the **No. Printed** by 1, and modifies and commits the table.

Now create the actual report in the next tasks.

4. Use the wizard to create the **Seminar Reg.-Participant List** form-type report (123456701) based on the **Seminar Registration Header** table (123456710) table. Add the following fields: **No.**, **<Separator>**, **Seminar Code**, **Seminar Name**, **Starting Date**, **Duration**, **Instructor Name** and **Room Name** to the report. Save the report.

5. Set the properties for the **Seminar Registration Header** data item so that it is sorted by **No.** and so that the filter fields the user can fill in when running the report are **No.**, **Seminar Code**, **No. Printed**.

6. Add a data item called **CopyLoop** for the **Integer** virtual table.

   – Set the property so that the data item indents to the first level.
   – Set the **DataItemTableView** property so that the table sorts by Number.
   – Set the property so that a new page is created for each record.

7. Add a data item called **PageLoop** for the Integer virtual table.

   – Set the property so that the data item indents to the second level.
   – Set the property so that the table sorts by Number where the Number is 1.

8. Add a data item for the **Seminar Registration Line** table.

   – Set the property so that the data item indents to the third level.
   – Set the property so that the data item sorts by **Document No.** and **Line No.**
   – Set the **DataItemLinkReference** and the **DataItemLink** properties so that the data item links to the **Seminar Registration Header** data item.

9. Set the property for the report so the caption is **Seminar Reg.- Participant List**.

10. Add the following sections to the report:

    – Two headers for the PageLoop data item.
    – A header and body section for the **Seminar Registration Line** data item.

11. Delete all sections except the ones just added.

12. Move the fields from the **Seminar Registration Header** header section to the first **PageLoop** header section and all the fields from the **Seminar Registration Header** body section to the second **PageLoop** header section.

    – Set the property for the **Seminar Registration Line** header section so that it prints on every page.
    – Set the property for the first **PageLoop** header section so that the section prints on every page.
    – Set the property for the second **PageLoop** header section so that the section prints on every page.

13. Add the **Bill-to Customer No**., **Participant Contact No**. and **Participant Name** fields to the **Seminar Registration Line** body section. Move the labels for these fields to the **Seminar Registration Line** header section.

14. In the second **PageLoop** header section, there are text boxes that identify the seminar registration and seminar information. Because the fields have moved from a different section, the program does not know from which table the data for the text boxes is taken. Set the SourceExpr property for each text box in this section so that the table is specified along with the field name.

Microsoft Official Training Materials for Microsoft Dynamics ™
Your use of this content is subject to your current services agreement

6-9

15. Define the following global variables for the report:

Name	DataType	Subtype	Length
SeminarCountPrinted	Codeunit	Seminar Registration-Printed	
NoOfCopies	Integer		
NoOfLoops	Integer		

16. Enter code in the appropriate trigger so that after the program gets the record of the Seminar Registration Header table, the program calculates the **Instructor Name** field.

17. Enter code in the appropriate trigger so that before the **CopyLoop** data item is run, the program sets the **NoOfLoops** to one more than the absolute value of the **NoOfCopies** and filters the table to the records between 1 and the **NoOfLoops**.

18. Enter code in the appropriate trigger so that after the program gets the record for the **CopyLoop** data item, the program sets the page number of the current report to 1.

19. Enter code in the appropriate trigger so that after the **CopyLoop** data item, if the current report is not previewed, the program runs the **Seminar Registration-Printed** codeunit for the **Seminar Registration Header** record.

20. On the request form for the report, add a text box with the caption "No. of Copies" with the source expression of the **NoOfCopies** variable.

21. Set the properties for the request form so that the form saves the values after it is closed.

Now the table and form can be created, from which the report can be selected and run.

22. Create the **Seminar Report Selections** table (123456705) with the following fields:

No.	Field Name	Type	Length	Comment
1	**Usage**	Options		Options: S.Registration.
2	**Sequence**	Code	10	Numeric field.
3	**Report ID**	Integer		Relation to the **ID** field of the Object table where the Type=Report.

No.	Field Name	Type	Length	Comment
4	**Report Name**	Text	80	FlowField; CalcFormula looks up the Object Caption field of the AllObjWithCaption table where the Object Type=Report and the Object ID= the field number of the Report ID. Must not be editable.

– The primary key for this table is **Usage**, **Sequence**.

23. Enter code in the appropriate trigger so that when the user enters or changes the value in the **Report ID** field, the program calculates the Report Name value.

24. Define a new function called **NewRecord**.

25. Enter code in the function trigger so that the function filters the **Seminar Report Selection** table to the corresponding Usage. If the function can find the last record in the record set and if the **Sequence** for the record is not blank, the function increments the **Sequence**. If the function cannot find the last record or if the **Sequence** is blank, the function sets the **Sequence** to 1.

26. Create the **Seminar Report Selection** form (123456723) with the **Sequence**, **Report ID** and **Report Name** fields as shown in the GUI design. Set the properties for the form so that it saves the values when the form is closed.

27. Set the properties for the **Report ID** text box so that the lookup form is the Objects form.

28. Set the properties for the **Report Name** text box so that there is no drill down and so that the lookup form is the Objects form.

29. Define a global variable called **ReportUsage** with the data type **Option** and a single option of **Registration**.

30. Define a function called **SetUsageFilter**. Enter code in the function trigger so that the function sets the filtergroup to 2, and if the **ReportUsage** is **Registration**, the function filters the table to where Usage is **S. Registration**. At the end, the function resets the filtergroup to 0.

31. Add a text box labeled Usage to the top of the form as shown in the GUI design. Set the properties of the text box so that the option is **Registration** and the source expression is the **ReportUsage** variable.

32. Enter code in the appropriate trigger so that the program runs the **SetUsageFilter** function when the user enters or changes the value in the **Usage** text box and when the program opens the form.

33. Enter code in the appropriate trigger so that after the program validates the **Usage** text box, the program updates the form.

34. Enter code in the appropriate trigger so that when the form opens a new record, the program runs the **NewRecord** function.

35. Now menu items can be added to the main **Seminar Menu** to allow the user to run the reports. Add the **Report Selections** menu item to the **Setup** menu item in the **Seminar Menu** as shown in the GUI design.

36. Add the reports as shown in the GUI design into the **Partner Menu Suite** under the **Order Processing – Reports** group.

37. The user should be allowed to run the participant list for a particular seminar registration from the registration form. To enable this, add a command button to the form that will run a codeunit to print the report. Develop the codeunit first. Create the **Seminar Document-Print** codeunit (123456703).

38. Define a function called **PrintSeminarRegistrationHeader** that takes one parameter. This parameter is a record variable of the **Seminar Registration Header** table, called **SemRegHeader**.

39. Enter code in the function trigger so that the function filters the **SemRegHeader** to the No. of the **SemRegHeader** record. The function filters **the Seminar Report Selection** table to those records with a **Usage** of S. Registration and a **Report ID** that is not 0. The function finds the first record in this set, and for each record in the set, modally runs the report corresponding to the **Report ID**.

40. Add the **Print** command button to the **Seminar Registration** form (123456710) as shown in the GUI design. Enter code in the appropriate trigger so that when the user clicks this button, the program runs the **PrintSeminarRegistrationHeader** function of the **Seminar Document-Print** codeunit.

Testing

1. If there are no **Seminar Registration** records, create one by selecting SEMINARS→ORDER PROCESSING→REGISTRATIONS from the main menu. Fill in all of the fields in the **General** and **Seminar Rooms** tabs and add at least two participants.

2. Use the **Seminar Report Selection** window to set up the **Seminar Reg.-Participant List** as the report to be run for **Registration**. To do so, select SEMINARS→SETUP→REPORT SELECTIONS from the main menu. Select Registration as the **Usage** and 123456701 on the first line for the **Report ID**. The report name should be filled in automatically.

3. Open the **Seminar Registration** form again and view the record prepared in step 1. Click the **Print** button.

4. The request form should open with the No. set to the current registration. Select **Print** or **Preview** to print and check the report.

5. Try printing the report with different parameters and registration data to verify that it is working correctly.

Certificate Confirmation

Solution Analysis

Upon completion of some seminars, participants receive a seminar certificate. These certificates are generated in the form of a report. This report should only include the participants whose seminar registration is posted.

Solution Design

This report must create a page for each participant in a given seminar registration. This will require data from the **Seminar Registration Header** and **Seminar Registration Line** tables.

For each **Seminar Registration Header** selected by the user in the request form, the report should print one certificate for each participant, with information from the **Seminar Name**, **Starting Date** and **Instructor Name** fields on the **Seminar Registration Header** table and from the **Participant Name** field on the **Seminar Registration Line** table.

The client has provided a model for how this report should appear. This sample report is in the Appendix B of this manual.

Lab 6.2 - Creating the Certificate Confirmation Report

1. Create report 123456702 **Seminar Reg.-Part. Certificate** with the data items **Seminar Registration Header** and **Seminar Registration Line**.

2. Set the properties for the **Seminar Registration Header** data item so that the table is sorted by **No**. The user should be able to filter on the **No**., **Starting Date** and **Seminar Code** fields when running the report. The data item should only print if there are records to be printed.

3. Set the properties for the **Seminar Registration Line** data item so that it is indented to the first level and so that the data item is sorted by **Document No**. and **Line No**. Set the link property and set the property so that there is a new page for each record.

4. Define a global record variable of the **Company Information** table, called **CompanyInfo**.

5. Enter code in the appropriate trigger so that before the program runs the **Seminar Registration Header** data item, it gets the **CompanyInfo** record and calculates the **Picture**.

6. Enter code in the appropriate trigger so that after the program gets the record for the **Seminar Registration Header** data item, it calculates the **Instructor Name**.

7. Enter code in the appropriate trigger so that after the program gets the record for the **Seminar Registration Line** data item, it skips to the next record if there is no **Participant Name** in this record.

8. The sections used in this report are two body sections and one footer section, all for the **Seminar Registration Line** data item. Set the property for the footer section so that the section prints at the bottom of every page.

9. Place a picture box in the first body section as shown in the sample report in Appendix B. The source for the picture box is the **Picture** field of the **CompanyInfo** record.

10. Place the following labels on the report for the fixed text of the report as shown in the sample report in Appendix B:

    – The "Participant Certificate" title, with a font size of 30 and bold.
    – The "has participated in seminar" phrase, with a font size of 15.

11. Set the captions for the labels so that they display the correct information.

12. Place text boxes on the report for the variable text of the report:

    – The participant name, with a font size of 25 and bold.
    – The seminar name, with a font size of 20 and bold.
    – The "on <date>" phrase, with a font size of 15.
    – The instructor's name, with a font size of 9.

13. Set the source expressions for the text boxes so that they display the correct information.

14. Add a shape to the report in the footer section. Set the properties for the shape so that the shape style is that of a horizontal line.

Testing

1. Select SEMINARS→ORDER PROCESSING→REPORTS→SEMINAR REG.-PART. CERTIFICATION from the main menu.

2. Select a registration and **Print** or **Preview** to verify that the report is functioning correctly.

Invoice Posting

Solution Analysis

When a seminar is finished, customers are invoiced for the participation of their registered participants. These invoices are implemented as reports, and should allow for multiple participants can include project and resource information.

This report must create an invoice for each customer within a filter specified by the user, with lines for each participant in a registered seminar (that falls within the filter set by the user). The user should also have the option of either creating the invoices or both creating and posting them at the same time.

Solution Design

Because invoices for posted seminars to be created, and because there can be additional charges, use the **Seminar Ledger Entries** to get the information to run this report. The report creates Sales Headers and Sales Lines as appropriate, and if the user has selected the option to post the invoices as well, the report runs the posting process for sales invoices.

Lab 6.3 - Creating the Invoice Posting Report

This report is different from the previous two in that it is not a report to be printed but is instead a processing-only report.

1. Create the **Create Seminar Invoices** report (123456700). Set the property to specify that this report is for processing-only.

2. Create a data item for the **Seminar Ledger Entry** table and set the properties for the data item as follows:

   – Specify the view of the data item to be sorted by **Bill-to Customer No.**, **Seminar Registration No.**, **Charge Type** and **Participant Contact No.**, where the **Remaining Amount** is not 0.
   – Specify that the user should be able to filter on the **Bill-to Customer No.**, **Seminar No.** and **Posting Date**.
   – Specify the variable name of the data item as **SeminarLedgerEntry**.

3. Define the following global variables for the report:

Name	DataType	Subtype	Length
SalesHeader	Record	Sales Header	
SalesLine	Record	Sales Line	
SalesSetup	Record	Sales & Receivables Setup	
GLSetup	Record	General Ledger Setup	
Cust	Record	Customer	
Job	Record	Job	
JobLedgEntry	Record	Job Ledger Entry	
ApplJobLedgEntry	Record	Job Ledger Entry	
JobPostingGr	Record	Job Posting Group	
CurrExchRate	Record	Currency Exchange Rate	
SalesCalcDisc	Codeunit	Sales-Calc. Discount	
SalesPost	Codeunit	Sales-Post	
JobLedgEntrySign	Code		10
Window	Dialog		
PostingDateReq	Date		

Name	DataType	Subtype	Length
DocDateReq	Date		
CalcInvDisc	Boolean		
PostInv	Boolean		
NextLineNo	Integer		
NoOfSalesInvErrors	Integer		
NoOfSalesInv	Integer		

4. Define the following text constants for the codeunit:

Name	ConstValue
Text000	Please enter the posting date.
Text001	Please enter the document date.
Text002	Creating Seminar Invoices...\\
Text003	Customer No. #1##########\
Text004	Registration No. #2##########\
Text005	The number of invoice(s) created is %1.
Text006	Not all the invoices were posted. A total of %1 invoices were not posted.
Text007	There is nothing to invoice.

5. Define a function called **FinalizeSalesInvHeader**. Set the property to specify that this is a local function.

6. Define a function called **InsertSalesInvHeader**. Set the property to specify that this is a local function.

7. Enter code in the **FinalizeSalesInvHeader** function trigger so that the function performs the following tasks:

    – If the **CalcInvDisc** variable is TRUE, the function runs the **Sales-Calc. Discount** codeunit with the **Sales Line** record and finds the **Sales Header**.
    – The function then performs a commit.
    – The function clears the **SalesCalcDisc** and **SalesPost** variables and increments the **NoOfSalesInv** by one.
    – If the **PostInv** variable is TRUE, the function clears the **SalesPost** variable. If running the **Sales-Post** codeunit with the **SalesHeader** returns FALSE, the function should increment the **NoOfSalesInvErrors** by one.

8. Enter code into the **InsertSalesInvHeader** function trigger so that the trigger performs the following tasks:

   – Initializes a new Sales Header record with a **Document Type** of Invoice and a blank **No**. and inserts it into the database.
   – Validates that the **Sell-To Customer No**. of the new record equals the **Bill-to Customer No**. of the ledger entry.
   – If the **Bill-to Customer No**. value is not the same as that of the **Sell-To Customer No**., the program validates that the **Bill-to Customer No.** of the new record equals the **Bill-to Customer No.** of the ledger entry.
   – Validates that the **Posting Date** of the new record is the **PostingDateReq**.
   – Validates that the **Document Date** of the new record is the **DocDateReq**.
   – Validates that the **Currency Code** of the new record is blank.
   – Modifies and commits the new record.
   – Sets the **NextLineNo** variable to 10000.

9. Enter code into the appropriate trigger so that just before the data item is run, the program performs the following tasks:

   – Shows an error if the **PostingDateReq** or **DocDateReq** is empty.
   – Opens a dialog window with the text constants Text002, Text003, and Text004.

10. Enter code in the appropriate trigger so that after the program gets a record for the data item, the program performs the following tasks:

   – Gets the **Job Ledger Entry** corresponding to the **Seminar Ledger Entry** record.
   – If the **No**. of the **Customer** record is not the same as the **Bill-to Customer No**., the program gets the **Customer** record corresponding to the **Bill-to Customer No**.
   – If the **Customer** is not blocked for All or is only blocked for Invoice, the program proceeds with the tasks that follow.
   – If the **Bill-to Customer No**. is different from that on the current Sales Header record, the program updates the dialog window with the new **Bill-to Customer No**. If the **No**. on the **Sales Header** is not blank, it runs the **FinalizeSalesInvHeader** function. The program then runs the **InsertSalesInvHeader** function and sets the **Document Type** and **Document No**. fields on the **Sales Line** to be those of the **Sales Header**.
   – Updates the dialog window with the **Sales Registration No**.
   – Sets the **Type** on the **Sales Line** according to the **Type** on the **Job Ledger** Entry.

- – If the **Type** on the **Job Ledger Entry** is G/L Account, the program tests that the **Job Posting Group** on the corresponding **Job** record is not blank. The program tests that the **G/L Exp. Sales Acc**. field on the corresponding Job Posting Group record is not blank. The program sets the **No.** of the Sales Line to the G/L Exp. Sales Acc. value.
- – If the **Type** on the Job Ledger Entry is not G/L Account, the program sets the **No**. of the Sales Line to the **No.** on the Job Ledger Entry.
- – Fills the Sales Line fields as follows: **Document Type** and **Document No**. from the Sales Header, **Line No**. from the NextLineNo variable, **Description** from the Seminar Ledger Entry, **Work Type Code** and **Unit of Measure Code** from the Job Ledger Entry, and **Unit Price** from dividing the **Total Price** from the Job Ledger Entry by the **Quantity**.
- – If the **Currency Code** on the Sales Header is not blank, the program tests that the **Currency Factor** is blank and calculates the **Unit Price** for the Sales Line by running the ExchangeAmtLCYToFCY function of the Currency Exchange Rate table.
- – Fill the Sales Line fields as follows: **Unit Cost (LCY)** from the Total Cost of the Job Ledger Entry divided by the Quantity, and the **Quantity**, **Job No**., **Phase Code**, **Task Code**, and **Step Code** fields from the Job Ledger Entry.
- – If the **Total Price** on the JobLedgEntry is greater than 0, the program sets the JobLedgEntrySign to '+' and filters the ApplJobLedgEntry records to those where the **Applies-To ID** matches that of the JobLedgEntry, with a '-' sign at the end (meaning JobLedgEntry."Applies-To ID" + '-'). For these records, the program modifies the Applies-To ID to blank. If the Total Price is less than 0, the program performs the same steps with opposite signs.
- – Sets the **Applies-To ID** of the Job Ledger Entry record to the **Entry No**. plus the JobLedgEntrySign and modifies the record.
- – Sets the **Job Applies-To ID** to the **Applies-To ID** of the Job Ledger Entry record.
- – Sets the **Apply** and **Close (Job)** field on the SalesLine to TRUE.
- – Inserts the Sales Line record.
- – Increments the NextLineNo by 10000.
- – If the Customer record is blocked for All or Invoice, increments the NoOfSalesInvErrors by one.

11. Enter code in the appropriate trigger so that when the program posts the data item, it performs the following tasks:

    – Closes the dialog window.
    – If the **No.** of the Sales Header is not blank, runs the FinalizeSalesInvHeader function. The program then displays a message showing the number of errors that occurred, if any, or the number of invoices created.
    – If the **No.** of the Sales Header is blank, the program shows a message saying that there is nothing to invoice.

12. Place two new text boxes and labels on the request form. The sources for these text boxes are the variables PostingDateReq and DocDateReq.

13. Place two new check boxes and labels on the request form. The sources for these check boxes are the variables CalcInvDisc and PostInv.

14. Set the property for the request form so that the form saves the values after the form is closed.

15. Enter code in the appropriate trigger so that when the request form is opened, if the PostingDateReq and DocDateReq variables are not filled, they are set to the work date. Set the CalcInvDisc value to the value of the **Calc. Inv. Discount** field in the Sales Setup.

Testing

To test this portion of the seminar module, complete the steps described in previous exercises. In particular, it is important that the **Job** and **Customer** associated with the registrations being tested have been set up correctly.

1. Select SEMINARS→PERIODIC ACTIVITIES→PERIODIC ACTIVITIES from the main menu.

2. The **Create Seminar Invoices** request form should open. Enter data so that one or more posted registrations meet the criteria. (If there are no posted registrations, create one.) Click **OK** to run the processing-only report.

3. Look at the **Sales Invoices** that were created. See that the header and lines were created correctly based on the Seminar information. Expect to have one invoice per Bill-to Customer with a line for each contact registered. Check that the data flowed correctly from the registration (and the associated jobs, etc.) to the invoice.

4. Try running the **Create Seminar Invoices** process with the **Post Invoices Option** marked and unmarked and verify that the **Sales Invoices** are posted as indicated.

Conclusion

In this chapter, three reports were created. The first two, **Participant List** and **Certificate Confirmation** were normal reports. The third report was a processing-only report to enable the creation of invoices for customers with participants in completed seminars.

In the next chapter, statistics will be added to the Seminar module.

Test Your Knowledge

Review Questions

1. Place these triggers in the order in which they would "fire" in normal report processing: **OnPostDataItem**, **OnPreReport**, **OnAfterGetRecord**, **OnPreDataItem**, OnPostReport, **OnInitReport**.

2. What report function would you use to skip the processing of one record in a data item?

3. What are the advantages of using processing-only reports in some situations as opposed to codeunits?

4. What does the following line of code do and in what trigger would you expect it to appear?

```
CurrReport.SHOWOUTPUT(CurrReport.TOTALSCAUSEDBY =
FIELDNO("Posting Date"));
```

5. What do you have to do to print FlowField data when running a report? What trigger would you use for this code?

6. In what trigger do you first have access to the values the user entered on the request form?

Quick Interaction: Lessons Learned

Take a moment to note three key points you have learned from this chapter:

1.

_____

_____

_____

2.

_____

_____

_____

3.

_____

_____

_____