Symitar

Education & Technical Publications

PowerOn 101

Participant Guide

Rev. 10/11/2022

Contents

PowerOn 101 Overview

This Introduction to PowerOn course is designed for people with little or no Symitar experience. It covers everything a new specfile writer needs to know to get started using PowerOn. This means, however, that participants with more Symitar experience may find they already know some of the concepts and procedures we teach in this class. If you are such a participant, rest assured that you will learn many new and useful things in this course, even if you already know the basics.

Objective

Learn to write PowerOn specfiles using basic divisions, functions, and statements.

What is PowerOn?

PowerOn is the function of the Symitar software that enables the creation and production of custom-designed reports, file maintenance files, etc. A program written to generate a report or file using the PowerOn language is called a specfile.

The Introduction to PowerOn course covers the following topics and concepts in the order listed below:

- Pre-Work Review
- The TARGET and SELECT Divisions
- EasyWriter
- Test Data Creation
- Guidelines for Writing Specfiles
- PRINT, NEWLINE, and TITLE
- Field Values
- The COL Statement
- Headers, Trailers, and REPEATCHR
- Standard Totals and Subtotals
- The SUPPRESS Statement
- The LETTER Division
- Conditions (IF...THEN...ELSE)
- Read Literals
- The FORMAT Function
- Boolean Rules of Precedence
- Demand Specfiles and Mathematical Expressions
- Arithmetic Functions
- Enter Functions
- Date Functions
- FOR EACH Loops

- TOTAL Division
- Basic HTML
- HTML Display

Pre-Work Review

In this lesson, we will briefly review the objectives of each of the five pre-work lessons and answer any questions you may have about them.

Lesson 1: PowerOn Overview

- Define the following terms:
  - PowerOn: The name of the programming language
  - PowerOn Control: The Symitar function that executes the instructions in a specfile
  - Specfile: A file that contains instructions that enable PowerOn to produce custom output. Specfiles can be created from scratch or by using the EasyWriter wizard.
- List the elements that make up the structure of the PowerOn language:
  - Syntax
  - Semantics
  - Punctuation
- Explain the difference between batch and demand modes:
  - Batch looks at records across the entire database.
  - Demand looks at the contents of a single record.
- Describe six types of PowerOn output or tasks:
  - Validation
  - Loan calculations and APY calculations
  - File maintenance (FM) scripting
  - Audio response/home banking interactive specfiles
  - Collect information in dialog boxes
  - Display information to the screen in HTML format (Windows® interface)

Lesson 2: The Symitar Record Structure

- List the data files you can access with PowerOn.
- Briefly describe the information found in each data file.
- Define the following database terms:
  - Field: The smallest unit of data in Symitar; a single piece of information about a person, place, or thing
  - Record: A collection of related fields
  - File: A collection of related records
- Describe, or draw a diagram that describes, the structure of the Account file.

- Given the name of an Account file record, briefly describe the information stored in that record.

Lesson 3: Programming Terms

- List the seven data types and indicate the following for each data type:
    - Natural (or default) sort order
    - Justification (left or right)
- Define and identify an example of each of the following:
    - Variable: A temporary storage location for data used in a program
    - Array: A collection of variables stored sequentially in memory and accessed using an index value
    - Literal: A value entered in the code that does not change
    - Special literal: A special name used to pull values from the system
    - Read literal: A function used to prompt for and accept input in batch mode
    - Enter function: A function used to prompt for and accept input in demand mode
    - Expression: A PowerOn statement that evaluates to a single value
    - Function: A built-in procedure performed on a data item; distinguished from other keywords by the fact that a function takes at least one argument, enclosed in parentheses
    - Statement: A line of PowerOn code

Lesson 4: Specfile Division

- List the PowerOn divisions in the order they must appear in a specfile:
    - Specfile type (optional)
    - TARGET (required)
    - DEFINE (optional)
    - SETUP (optional)
    - SELECT (optional)
    - SORT (optional)
    - PRINT or LETTER (required)
    - TOTAL (optional)
    - PROCEDURE (optional)
- Indicate whether each division is required for Symitar to process your specfile.
- Briefly describe what type of instructions are processed in each division.

Lesson 5: Symitar Basics

- Start Symitar.
- Log on to the IBM® host.
- Log on to a Symitar directory with a User ID.
- Navigate between work areas.
- Identify the buttons in the PowerOn Control work area to create, revise, and save specfiles.
- Log off from Symitar.

Activity: Jeopardy

Your instructor will now divide you into teams. Members of each team will face off to answer questions about the material we have just reviewed in a Jeopardy-style game. Each correct answer earns a point for your team.

Your instructor will display the question on the screen and read it aloud. The first competitor to ring the bell gets to answer the question. As in Jeopardy, if the first contestant's answer is incorrect, the other contestant has an opportunity to answer the question correctly to earn a point for his or her team. This means it is to your team's advantage for you to "ring in" as soon as you know the answer to the question, but only if you are sure you know the answer.

The TARGET and SELECT Divisions

Your first task when you write any specfile is to determine which record to target. To be able to determine which record to target, you must understand how PowerOn processes the instructions you provide in each division, particularly the SELECT and PRINT divisions.

In this lesson, we will talk about the relationship between the TARGET and SELECT divisions and how that relationship affects the output in your PRINT (or LETTER) division.

Objectives

At the end of this lesson, you will be able to:

- Define the term record path
- Explain how the record path affects the choice of a target record
- Describe how and when you can access the primary Name record when it is not in the targeted record's record path
- Explain how the TARGET, SELECT, and PRINT divisions interact
- Determine which record to target when writing a specfile and explain why

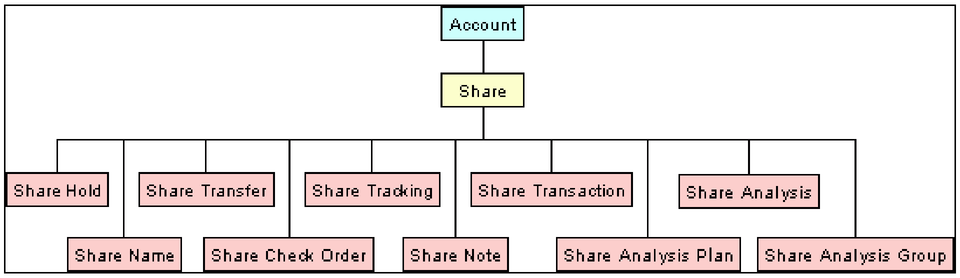
Record Paths

In your pre-work assignment, you learned that Symitar organizes records into the equivalent of family trees. The first (or top) record in any file is the file's parent record. For example, the Account record is the parent record in the Account file. A single Account record represents each member account in Symitar. All other records belonging to a member account are children, grandchildren, or great-grandchildren of the Account record.

A record path is an outgrowth of the family tree concept. To continue using the genealogical analogy, the record path is the direct line of descent (and ascent) of a single record. For example, the Share record has the following record path:

- Account (parent)
    - Share
        - Share Hold (child)
        - Share Name (child)
        - Share Transfer (child)
        - Share Check Order (child)
        - Share Note (child)
        - Share Tracking (child)
        - Share Transaction (child)
        - Share Analysis Plan (child)
        - Share Analysis (child)
        - Share Analysis Group (child)

The following illustration is another, more graphical way to represent this record path:



Understanding the record path is important because the system can look up and down in a record path, but not sideways. In other words, when you target a record in a specfile, the system can only gather information from that record's ancestor records (parent and grandparent) and its descendant records (children and grandchildren). All other records are "sideways" in the family tree, and therefore cannot be seen.

In the previous example, if you target the Share record, you can select targeted Share records based on information in the Account record and all the child records of the share, but not based on information in any other record in the Account file.

> **TIP**
> You can always access information in the primary Name record in the SORT and PRINT divisions. You cannot, however, access the primary Name record in the SELECT division if you target one of its siblings. For more information, see *Primary Name Record* later in this lesson.

Primary Name Record

Because the Name record is not a parent to any other record, the only time it is in the available record path is when the target record is ACCOUNT or NAME. Many reports based on the Account file are more meaningful when you include information from the Name record (such as name and address).

Because of this, the system handles access to the primary Name record in a special way. The primary Name record is the mandatory Name record the system creates when you set up an account. It has the **Name Type** field set to **(0) Primary**.

If you don't target the Name record and you don't use a field from the Name record in the SELECT division, the system makes the information from the primary Name record available to you for sorting and printing. You don't need to have any special instructions in the specfile; just go ahead and use a field from the primary Name record as a sort key or as part of a PRINT statement.

However, if you target the Account record and use a field from the Name record in the SELECT division to select accounts, the information from the primary Name record is not necessarily available. The Name record available for sorting and printing in this case is the Name record that qualified the Account record for inclusion on the report; that Name record can be a joint Name record or a mailing address Name record.

How the TARGET, SELECT, and PRINT Divisions Interact

Be aware that in qualifying the target record, the selection criteria look only for the first true condition per target record. The sole purpose of the SELECT division is to decide whether a target record qualifies for the report.

When you target a record, the system starts with the first record of that type and looks at each one in the database. As it does, it checks the record against the criteria in the SELECT division. As soon as the system determines that the record meets the criteria, it places the targeted record in a temporary file for further processing in the PRINT division and moves on to the next record of that type in the database.

Once the system finishes selecting targeted records, the PRINT division is then executed one time for each record in the temporary file. This means that you get at least one printed line of information on your report for every selected target record.

Suppose that you need to create a report that lists all loans paid by automatic transfer. There are three possible target records in the Account file to produce this report:

- Account
- Loan
- Loan Transfer

The following pages illustrate how the TARGET, SELECT, and PRINT divisions interact when each of these targets is used to produce the report.

TARGET=ACCOUNT

The following pages illustrate how the TARGET, SELECT, and PRINT divisions interact when each of these targets is used to produce the report.

This is the code with the Account record as the target record:

```
TARGET=ACCOUNT
SELECT
 ACCOUNT:CLOSEDATE='--/--/--' AND
 LOAN:CLOSEDATE='--/--/--' AND
 (LOAN TRANSFER:EXPIRATIONDATE='--/--/--' OR
 LOAN TRANSFER:EXPIRATIONDATE>=SYSTEMDATE)
END
PRINT TITLE="Loans Paid by Automatic Transfer"
 HEADER="Account #   ID"
 HEADER="----------------"
 COL=001 ACCOUNT:NUMBER
 COL=013 LOAN:ID
END
```

The following illustration shows how the system selects records for the report based on these specifications. The highlighted records are the only ones the system uses to determine which target records to select for further processing.

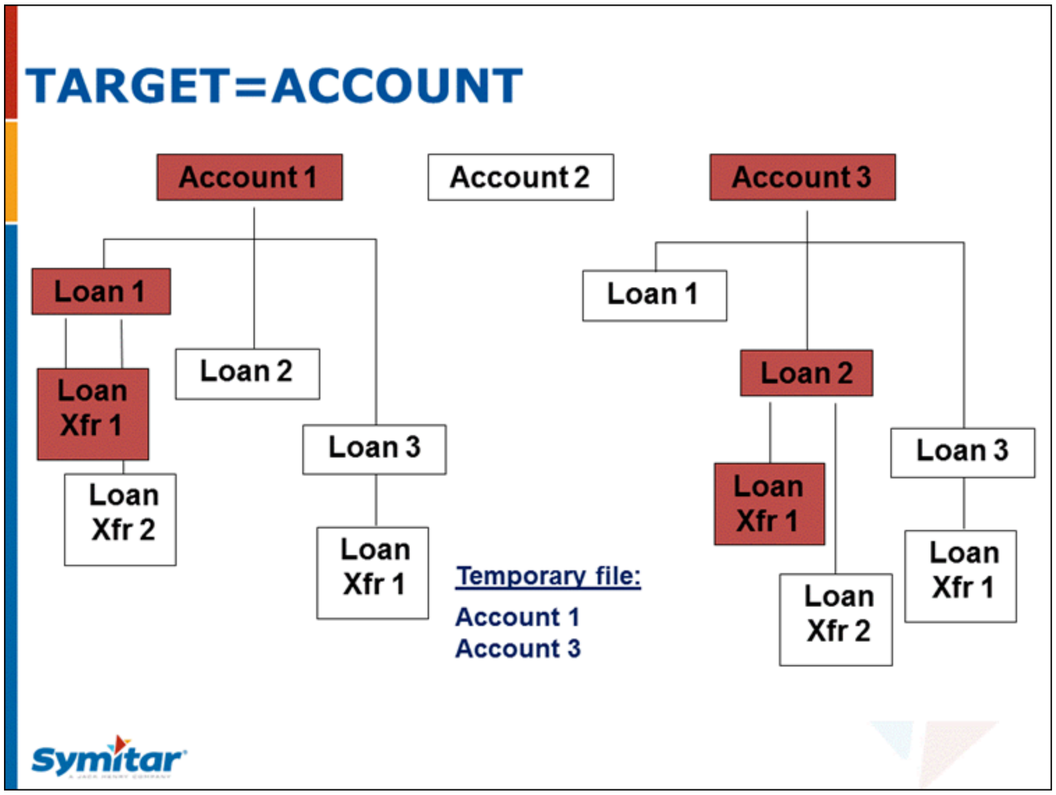


Figure 1: TARGET=ACCOUNT

The report generated by the file looks like this:

```
Loans Paid by Automatic Transfer
Account #    ID
-----------------
```

```
0000000001   0001
0000000003   0002
```

As you can see, the report produced with the Account as the target record does not achieve the desired result. The report misses two loans paid by automatic transfer: Loan 3 in Account 1 and Loan 3 in Account 3.

TARGET=LOAN

This is the code with the Loan record as the target record.

```
TARGET=LOAN
  SELECT
   ACCOUNT:CLOSEDATE='--/--/--' AND
   LOAN:CLOSEDATE='--/--/--' AND
  (LOAN TRANSFER:EXPIRATIONDATE='--/--/--' OR
   LOAN TRANSFER:EXPIRATIONDATE>=SYSTEMDATE)
  END
  PRINT TITLE="Loans Paid by Automatic Transfer"
   HEADER="Account #   ID"
   HEADER="----------------"
   COL=001 ACCOUNT:NUMBER
   COL=013 LOAN:ID
  END
```

The following illustration shows how the system selects records for the report based on these specifications. The highlighted records are the only ones the system uses to determine which target records to select for further processing.

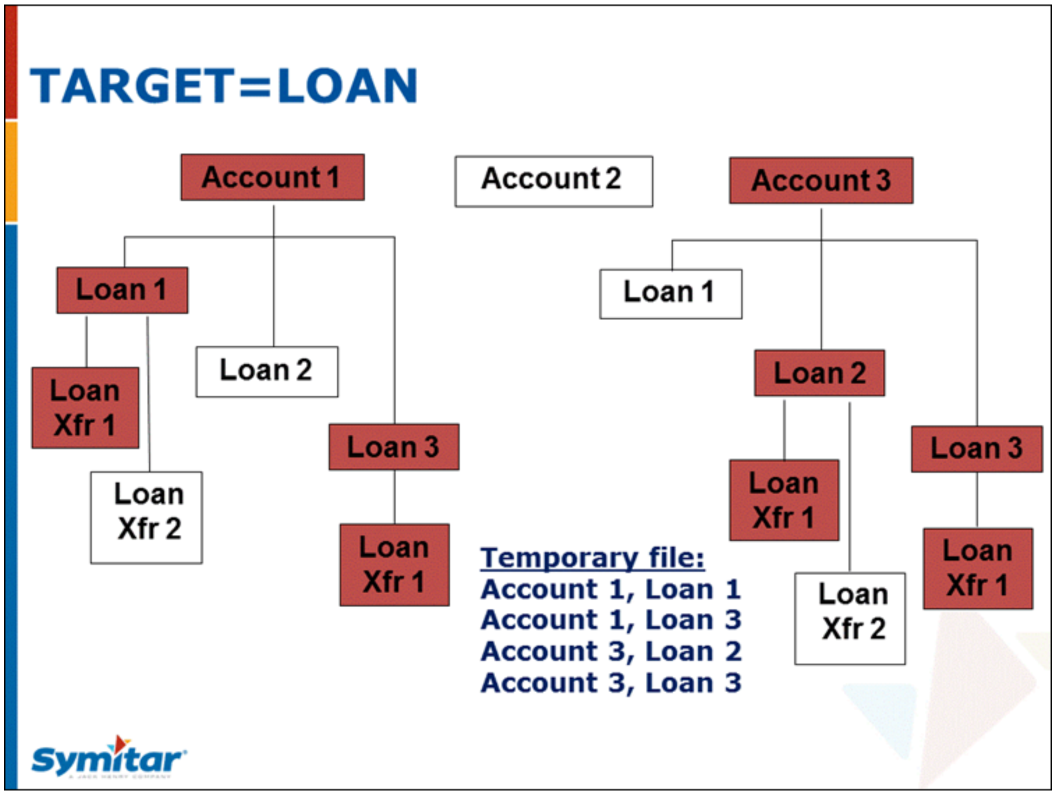


Figure 2: TARGET=LOAN

The report generated by the specfile looks like this:

```
Loans Paid by Automatic Transfer
Account #    ID
-----------------
0000000001   0001
0000000001   0003
0000000003   0002
0000000003   0003
```

This specfile achieves the desired result because it lists each Loan record that has an automatic Transfer record beneath it.

TARGET=LOAN TRANSFER

This is the code with the Loan Transfer record as the target record.

```
TARGET=LOAN TRANSFER
  SELECT
   ACCOUNT:CLOSEDATE='--/--/--' AND
   LOAN:CLOSEDATE='--/--/--' AND
  (LOAN TRANSFER:EXPIRATIONDATE='--/--/--' OR
   LOAN TRANSFER:EXPIRATIONDATE>=SYSTEMDATE)
  END
  PRINT TITLE="Loans Paid by Automatic Transfer"
   HEADER="Account #   ID"
   HEADER="----------------"
   COL=001 ACCOUNT:NUMBER
   COL=013 LOAN:ID
  END
```

The following illustration shows how the system selects records for the report based on these specifications. The highlighted records are the only ones the system uses to determine which target records to select for further processing.

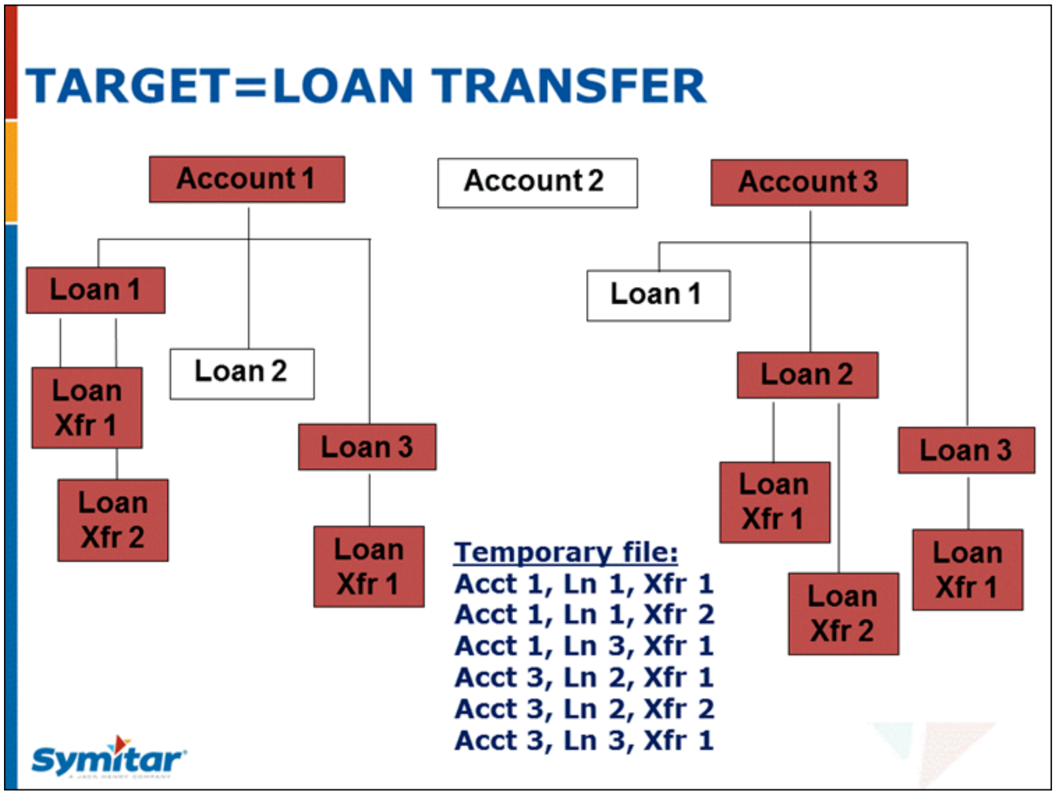


Figure 3: TARGET=LOAN TRANSFER

The report generated by the specfile looks like this:

```
Loans Paid by Automatic Transfer

Account #    ID
-----------------
0000000001   0001
0000000001   0001
0000000001   0003
```

```
0000000003   0002
0000000003   0002
0000000003   0003
```

This report achieves the desired result because it lists all the loans with automatic transfers. You'll notice, however, that it lists two loans twice (Loan 1 in Account 1 and Loan 2 in Account 3). That is because each of these loans has two Loan Transfer records beneath it. As a result, the Loan Transfer record is probably not the best target record for this report.

Activity: TARGET and SELECT

Your instructor will break you into several groups for this activity. The following pages provide three descriptions of proposed specfiles and the code for each specfile with the TARGET undefined. Your instructor will assign one specfile case study to each group.

As a group, analyze the description of the specfile to determine which record in the Account file will make the best target to achieve the desired results. You may want to draw a diagram or illustration to show which record makes the best target and why. You can create diagrams similar to those in this workbook, or any other type of illustration you find helpful.

When all groups have finished, your instructor will ask a member of your group to present your case study to the class.

Specfile #1

The writer of the following specfile wants a listing of all open, non-zero-balance shares with at least one active, unexpired check hold. A check hold is indicated by the existence of a Share Hold record with the **Hold Type** field set to **(1) Check Hold**. Determine which record to target for this report.

```
TARGET=

SELECT
 ACCOUNT:CLOSEDATE='--/--/--' AND
 SHARE:CLOSEDATE='--/--/--' AND
 SHARE:BALANCE>$0.00 AND
 SHARE HOLD:TYPE=1 AND
(SHARE HOLD:EXPIRATIONDATE='--/--/--' OR
 SHARE HOLD:EXPIRATIONDATE>=SYSTEMDATE)
END

PRINT TITLE="Share with Check Holds"
 COL=001 ACCOUNT:NUMBER
 COL=015 NAME:SHORTNAME
 COL=035 SHARE:ID
 COL=045 SHARE:TYPE
 COL=065 SHARE:BALANCE
 NEWLINE
END
```

Specfile #2

The writer of the following specfile wants a list of accounts that have:

- At least one open share draft account; a Share record with the **Share Code** field set to **(1) Draft**.
- At least one unexpired overdraft transfer source; a Share Transfer record with the **Transfer Type** field set to **(0) Overdraft**.

The report should print the ID of the first share draft found and the ID of the overdraft transfer source of the first Share Transfer record found. Determine which record to target for this report.

```
TARGET=

SELECT
 ACCOUNT:CLOSEDATE='--/--/--' AND
 SHARE:CLOSEDATE='--/--/--' AND
 SHARE:SHARECODE=1 AND
 SHARE TRANSFER:TYPE=0 AND
(SHARE TRANSFER:EXPIRATIONDATE='--/--/--' OR
 SHARE TRANSFER:EXPIRATIONDATE>=SYSTEMDATE)
END

PRINT TITLE="Accounts with Drafts and Overdraft Protection"
 COL=001 ACCOUNT:NUMBER
 COL=015 NAME:SHORTNAME
 COL=035 SHARE:ID
 COL=043 SHARE TRANSFER:ID
 NEWLINE
END
```

Specfile #3

The writer of the following specfile needs a list of the names and Social Security Numbers of all share-level joint owners; Share Name records with the **Name Type** field set to **(1) Joint**. The report should also provide the account number and ID of the share for which the person is a joint owner. Determine which record to target.

```
TARGET=

SELECT
 ACCOUNT:CLOSEDATE='--/--/--' AND
 SHARE:CLOSEDATE='--/--/--' AND
 SHARE NAME:TYPE= 1
END

PRINT TITLE="Share Level Joint Owner Names and SSNs"
 COL=001 ACCOUNT:NUMBER
 COL=015 SHARE:ID
 COL=023 SHARE NAME:SHORTNAME
 COL=045 SHARE NAME:SSN
 NEWLINE
END
```

EasyWriter

When you need to generate a simple report quickly and would rather not write your specfile from scratch, you can use Symitar EasyWriter to write the specfile for you.

Objectives

At the end of this lesson, you will be able to do the following:

- Create a simple batch specfile using EasyWriter
- Run a specfile in batch mode
- View the resulting report
- Analyze a specfile created by EasyWriter

What is EasyWriter?

EasyWriter is a menu-driven interface (or “wizard”) to the PowerOn program that prompts you for the data you want to include on your report and stores the information in a snapshot file. It includes a drag-and-drop interface that lets you select the fields that will appear on your report and place them wherever you would like on the page. From the snapshot file, EasyWriter writes your specfile for you.

EasyWriter is a terrific tool and a time saver under certain circumstances, but it does have some limitations. It cannot fulfill all your PowerOn needs. First, let’s talk about what EasyWriter can do.

EasyWriter Can

- Access data in any PowerOn data file
- Select records for your report based on
  - Information in a record
  - The presence or absence of a record
- Sort the output of your report
- Lay out your report so that columns of information do not overlap
- Display a mock output of your report before you run it
- Print a detailed line of information for each selected record
- Provide a total dollar amount of all records on the report

EasyWriter Cannot

- Select records that meet some selection criteria but not others (EasyWriter always uses AND when you specify multiple selection criteria)
- Perform FOR EACH loops (this means that once a specfile finds a record that meets the selection criteria, it goes on to the next target record)
- Generate more than one line of information per selected record
- Perform mathematical calculations (except dollar amount totals)
- Write special types of specfiles (validation, Windows®, etc.)
- Produce a summary report (EasyWriter must print one line of information for each record selected)
- Print anything on a report except headers and data from a field in the database

Accessing EasyWriter

You can access the EasyWriter work area in one of the following ways:

- From the menu bar, select **Navigate** > **Marketing** > **EasyWriter**.

- Click **EasyWriter**.

Once you access EasyWriter, a series of dialog boxes prompts you through the creation or revision of your snapshot file.

Running a Batch Mode Specfile

EasyWriter is typically used to create batch mode specfiles. For this reason, we will run the specfile you create during this lesson as a batch specfile.

About this task

EasyWriter assumes that you want to write a batch specfile rather than a demand specfile. For this reason, if your specfile prompts a user for any information, EasyWriter uses the batch mode read literals to generate the prompts. Therefore, if you want to use a specfile written by EasyWriter in demand mode and that specfile prompts the user for information, you must revise it to use ENTER functions before installing it for demand use

Steps

1. From the menu bar, select **Navigate** > **Information Systems** > **Batch Control**, or click **Batch Control**.
2. Click the arrow next to the **Program** drop-down field, scroll down and select **PowerOn**.
3. At the **Specification File (newline when done)** prompt, enter the name of the specfile you want to run. This prompt is repeated up to 50 times or until you leave it blank and press **Enter**.
4. If the specfile prompts the user to enter data, the prompts appear one at a time. Answer each prompt appropriately.
5. Symitar displays the Batch Options prompt. Batch options allow you to receive a message when you complete the job file, choose the queue priority, and set the start date and time. If you do not want to select batch options, accept the default of **No** at the **Batch Options** prompt. The **Batch Queue** prompt appears.
6. Enter the number of the batch queue where Symitar should run the job file. In most cases, you should accept the default batch queue (**0**). The following message appears: `Preparing to run Batch Program REPWRITER in Queue 0 in SYMXXX`

   The message is followed by the **Okay** prompt.
7. Answer **Yes** to run the program or **No** to cancel it.

Results

After your specfile runs, the resulting report ends up in the Print Control work area.

Viewing the Batch Specfile Report

A batch specfile normally produces a report that ends up in the Print Control work area. From Print Control, you can view, print, and delete the reports you create with batch specfiles.

Before you begin

Access Print Control in one of these ways:

- From the menu bar, select **Navigate** > **Information Systems** > **Print Control**.
- Click **Print Control**.

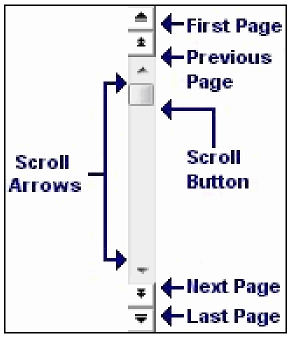
Steps

1. To display a list of reports, click the down arrow at the **Select** prompt and select an option from the drop-down list. For this exercise, **Today's Reports** or **Last Few Reports** are the best options. After choosing your option, click **OK**.

   If you choose **Last Few Reports**, you must specify the **Quantity** (or number) of reports to list, and then click **OK**.

   A list of reports meeting your search criteria appears in chronological order (oldest report first). You can change this to reverse chronological order (newest report first) by clicking the Time column header.
2. To view a report, select it from the list, and then click **View Report** on the Print Control work area toolbar.
   The contents of the report appear on the bottom half of the screen in a scrollable window.

   You can use the buttons on the right side of the window to navigate through the report, as shown below:



Activity: Using EasyWriter

Working as a group, use EasyWriter to create a specfile named <initials>.EZPRAC and titled *New Accounts with E-Mail Addresses* that does the following:

- Selects all accounts that meet the following criteria:
    - Are open
    - Were opened on or after a date entered by the person running the specfile
    - Have a non-blank email address in the primary Name record
- Sorts the report first by the account's **Open Date** field and then by account **Type** field
- Prints the member's account number, short name, account type, email address, and account open date

When you are done creating your EasyWriter snapshot file, go to the Batch Control work area and run it, and then view the results in the Print Control work area.

Activity: Parsing an EasyWriter Specfile

Now that you have written a specfile using EasyWriter, it is time to have a look at the PowerOn code EasyWriter created. Parsing (reading through this code and making sense of it) is an excellent way to begin learning PowerOn syntax and keywords.

Steps

1. Click **PowerOn Control**.
2. Click **Open**.

   The *Open Specfile* window appears.
3. Type the name (**<initials>.EZPRAC**) in the **File Name** field.

   > **TIP**
   > File names are case-sensitive. Use uppercase and lowercase letters exactly as they appear in the name of the file you want to revise.

   The specfile opens in a text editor window.

Test Data Creation

In this lesson, you will learn how to set up data to test a specfile's operation. You will also gain more familiarity with the Symitar database and with procedures for creating and modifying records.

You have now created your first custom report using EasyWriter. The report looks good and seems accurate. But how do you know it really is accurate? Are you sure that the report lists every record it should list and none that it should not?

The only way to verify the report's accuracy is to create data to test the specfile's operation. Creating test data and running your specfiles to verify that they operate as you expect is essential to good specfile writing.

Objectives

At the end of this lesson, you will be able to do the following:

- Determine what data to use to create or modify to test a specfile's operation and accuracy
- Create or modify records appropriately to test a specfile
- Verify test results

How Test Data Works

Just as a laboratory researcher uses control subjects to ensure the validity of test results, you can use control data to ensure the validity of your specfiles results.

Properly defined control data achieves two things:

- It proves that your report lists all the records it is intended to list
- It shows that your report does not list any records it is not intended to list

Creating test data requires you to modify the database. For obvious reasons, this is not something you should do in a live credit union database. You should always write and test specfiles in a test directory before running them in the live directory.

For the time being, the only part of your specfiles that requires testing is the selection criteria. Since the selection criteria determine which records are selected for the report and which are not, you must be sure that the criteria work as you intend before you can consider the report complete and correct.

Test Selection Criteria

When your specfile contains selection criteria (a SELECT division), you should create or revise both the records that you know should be and the records that you know should not be selected by your specfile.

For example, assume that your specfile targets the Share record. Your selection criteria read as follows:

```
SELECT
  SHARE:CLOSEDATE='--/--/--' AND
 (SHARE:TYPE>=0 AND SHARE:TYPE<=20) AND
  ANYSERVICE(SHARE,20)
END
```

This set of criteria should select a share for further processing only if the share is open, its **Type** field is set to a value between 0 and 20, and its **Service Code** field is set to **20**.

To test these criteria, you must create or revise Share records so that you have the following:

- Several shares that meet all the criteria and should therefore appear on the report. Although it’s acceptable to create just one record that you know should appear on the report, it is a good idea to create several. The presence or absence of all or only some of the records can alert you to other problems with your specfile, particularly a problem with the target record.
- At least one share in the database that does not meet each one of the following criteria:
    - One closed share that has the **Type** field set to a value between 0 and 20 and the **Service Code** field set to **20**
    - One open share that has the **Service Code** field set to **20** and the **Type** field set to a value that is not between 0 and 20
    - One open share that has the **Type** field set to a value between 0 and 20, that does not have the **Service Code** field set to **20**

When you run the specfile, check the resulting report to verify that it does not include any of the shares you set up to fail at least one of the criteria, and make sure that it does include the records you set up to meet all of the criteria.

Create and Revise Account Records

Throughout this class, you will be creating and revising records to test your specfiles. Following are basic procedures for creating and revising member Account records using the Account Manager work area.

Account Creation Wizard

You must use a special function of Symitar called the Account Creation Wizard to create a new member Account record. The Account Creation Wizard guides you step-by-step through the process of creating a new account. Many of the fields and selection options displayed in the Account Creation Wizard are credit union-defined.

Following are a few points to remember when using the Account Creation Wizard:

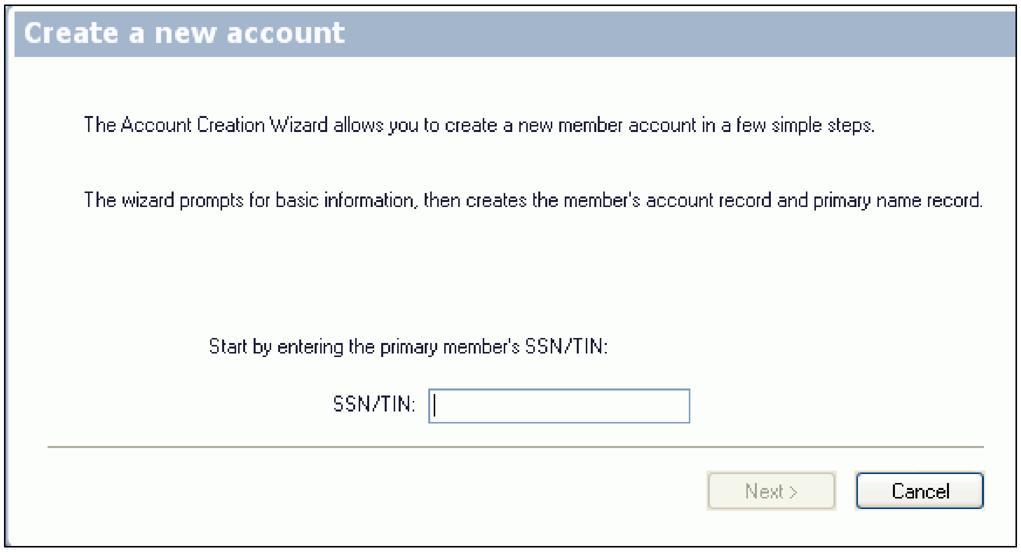
- Each window in the Account Creation Wizard has a **Cancel** button. You can click Cancel to stop creating the new account, and the you return to the **Account** prompt in the Account Manager work area.
- While creating the new account, you must make entries in record fields. To get information on the entries to make at a specific field, choose one of the following:
  - Press **Shift+F1** to view credit union-defined help for the field.
  - Look up the field in the*Programming > Files, Records, & Fields* section of *Symitar eDocs*.

Creating an Account

Create an account using the Account Creation Wizard.

Steps

1. From the menu bar, select **Navigate** > **Member Services** > **Account Manager**, or click **Account Manager**.
2. Click **Create a new account**.



Create a new account

The Account Creation Wizard allows you to create a new member account in a few simple steps.

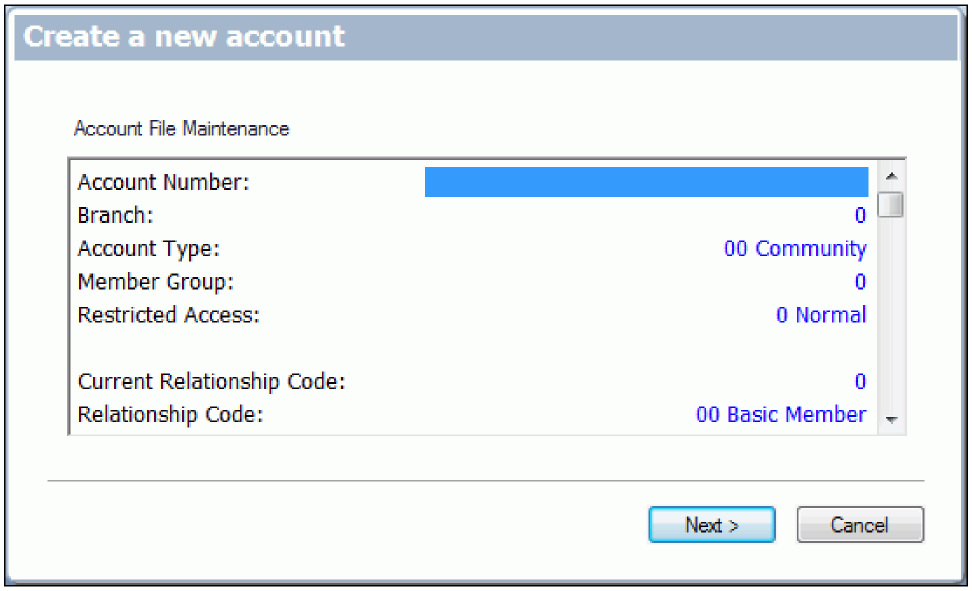
The wizard prompts for basic information, then creates the member's account record and primary name record.

Start by entering the primary member's SSN/TIN:

SSN/TIN:

   The Account Creation Wizard opens and prompts for the member's Social Security Number.
3. Enter the Social Security number, and then click **Next** to continue.
   The system prompts for the account type.
4. Use the drop-down list to select an account type. Click **Next** to continue.
   Symitar prompts for the information required to create the Account record.

5. Enter the information required at each field.


Create a new account

Account File Maintenance

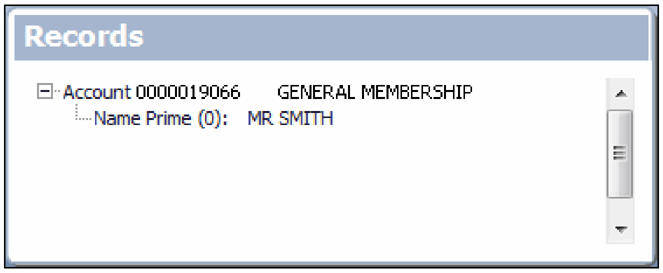
Account Number:	
Branch:	0
Account Type:	00 Community
Member Group:	0
Restricted Access:	0 Normal
Current Relationship Code:	0
Relationship Code:	00 Basic Member

TIP
The fields that appear during Account record creation are credit union-defined. The fields you see may be different from those shown here.

6. Symitar prompts for the information required to create the account's primary Name record. Enter the information. Use the scroll bar to display additional fields. Click **Next** to continue.
7. Symitar prompts for an **Account Number**. Choose one of the following: •

   - Click **Finish** to accept the default.
   - Enter an account number and click **Finish**.

   After you click **Finish** at the *Account Number* window, Symitar creates the Account and primary Name records. The Account Record Tree appears.


Records

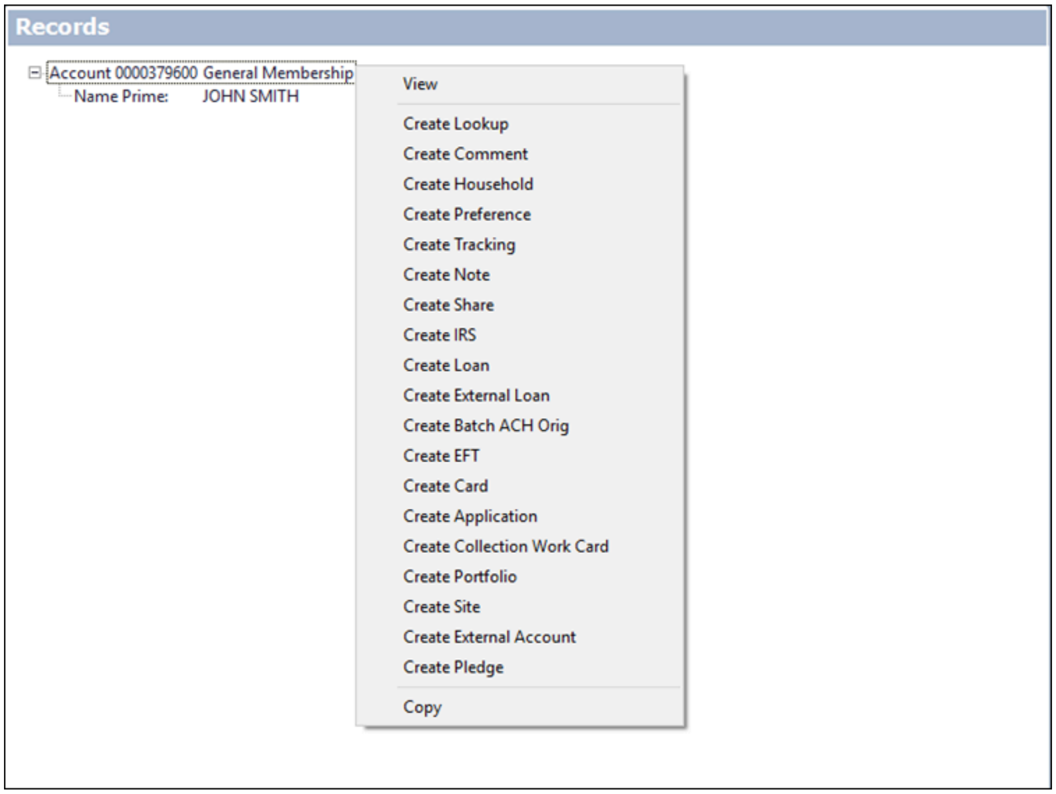
Account 0000019066 GENERAL MEMBERSHIP
Name Prime (0): MR SMITH

Create Additional Records and Child Records

You can create additional records and child records for the account.

For example, the Account record does not store information about shares held by a member. You must create at least one Share record to store this information.

To see a list of the records you can create for the account, right-click the Account record. The list of child records appears.



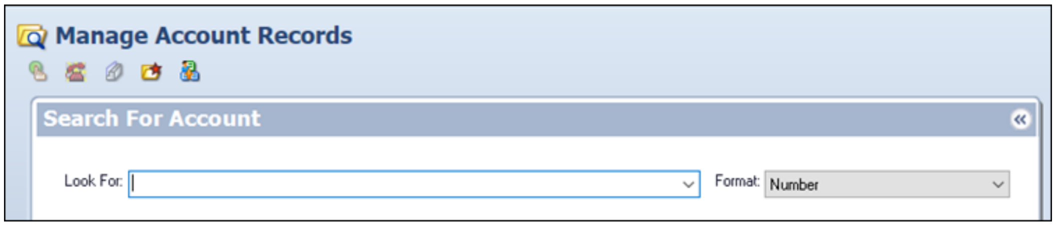
Select any item from this list to create a record. For example, to create a Share record, select **Create Share**. Symitar prompts you for the information required to create the Share record.

Revising Existing Records

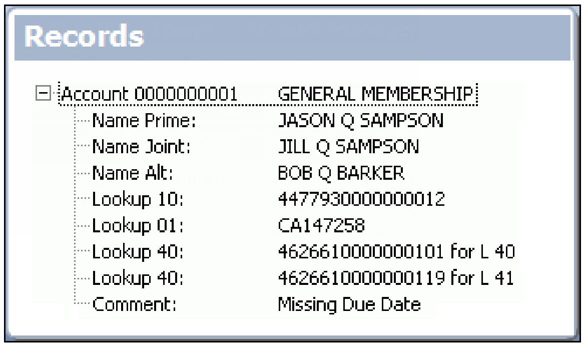
You can revise any record in a member account by following this procedure.

Steps

1. Go to the Account Manager work area.
2. Enter the member's account number in the **Look For** box, and then press **Enter**.



The Account Record Tree for the member's account appears.



Account 0000000001	GENERAL MEMBERSHIP
Name Prime:	JASON Q SAMPSON
Name Joint:	JILL Q SAMPSON
Name Alt:	BOB Q BARKER
Lookup 10:	4477930000000012
Lookup 01:	CA147258
Lookup 40:	4626610000000101 for L 40
Lookup 40:	4626610000000119 for L 41
Comment:	Missing Due Date

3. Select the record you want to modify from the Account Record Tree. The current field values for the selected record appear.
4. To make changes, click **Revise** in the upper left corner of the window.
5. To edit a field, click the field and make a new entry. You can make entries using the keyboard, or in some cases, by selecting an option from a drop-down list. When you are done, click **OK** to save your changes, or click **Cancel** to discard them.

Activity: Creating Test Data

You need to create some test data to check the accuracy of the specfile you wrote (called **<initials>.EZPRAC**) using the EasyWriter in the last lesson.

The SELECT division for your specfile should look something like this:

```
SELECT
 ACCOUNT:CLOSEDATE='--/--/--' AND
 ACCOUNT:OPENDATE>=INPUTVAR001 AND
 NAME:EMAIL<>""
END
```

As a group, discuss which records you need to adequately test the selection criteria. When you have decided which records you need, each of you should create or modify records as necessary to test the specfile, and then run the specfile again in batch mode. Review the resulting report to ensure that the report lists all the records it should and none of the records it should not.

Guidelines for Writing Specfiles

Before you write any specfiles of your own, there are a few specfile writing guidelines and procedures you need to know.

Objectives

At the end of this lesson, you will be able to do the following:

- Set standards for naming a specfile.
- Place comments in a specfile and explain why it's important to do so.
- Expressing literals
- Write a specfile that conforms to standard formatting conventions.
- Check a specfile for errors.

Naming Conventions

We strongly recommend that your credit union devise some conventions for assigning names to specfiles.

Clear naming conventions can make it much easier for all users to find the specfile they need.

- Specfile names can be up to 31 characters long.
- You can use all characters on your keyboard except * - + ’ " , \ and /.
- For batch mode specfiles, the name should correspond to the title that is printed at the top of the resulting report, abbreviated as necessary.
- For demand mode specfiles, the name should give the user an idea of what the specfile does, but do not start it with a number.
- To save an old version of a specfile when you make changes, rename the specfile with the suffix .BU (for backup).
- Specfiles that come in the Jack Henry–delivered PowerOn Specfile Library use one of the following prefixes:
  - **RB.** for batch mode specfiles
  - **RD.** for demand mode specfiles

Do not use RB. or RD. prefixes for your own specfiles. If you modify one of these specfiles, make a copy with a new name. Otherwise, you will overwrite your specfiles when you load a new release.

Setting naming conventions and sticking with them will not only help you remember the names of your specfiles, it will also help you find specfiles others have written.

Place Comments in a Specfile

When you write a specfile, it is clear to you why you are writing it and what it is supposed to do. But that might not be so obvious to someone who runs across your specfile later (or even to you a year after you have written it). To avoid this, you should always place a comment at the top of your specfile that explains what it does, who created the specfile, and the last modified date.

You can place a comment that provides information to another user anywhere in a specfile. Enclosing the text in brackets ([ ]) prevents PowerOn from trying to process that information.

The most common use of comments is at the top of your specfile, where you explain what the specfile does. You can also embed comments elsewhere in your specfile to help another user understand what you have done at any point in the specfile.

Here are some examples of comments in a specfile:

```
[This specfile finds all members with EFT records belonging
to payroll group 900 and creates a report listing these
accounts. Accounting uses this report to verify the paper
listing received from the company. Created by John Smith
Last modified 01/21/17.]

TARGET=ACCOUNT

SELECT
 ACCOUNT:CLOSEDATE='--/--/--' AND
 EFT:GROUP=900
END

SORT
 NAME:SSN        [Accounting needs report in SSN order]
END

etc.
```

Express Literals

A literal is a string of characters, digits, or other symbols that the specfile must use or Symitar must print exactly as it appears, symbol by symbol.

Literals have specific formats based on their data type:

- Character literals must be enclosed in double quotes (for example, "Monthly Report" or "Members with Combined Loan Balances > $1,000,000.00").
- Code literals (a series of digits) must not use a decimal point or commas (for example, 83).
- Date literal must be in the format MM/DD/YY or MM/DD/YYYY (must use slashes) and must be enclosed within single quotes (for example, '05/24/18'). Using a two-digit year will give the correct year for 1951 through 2050 (for example: '12/31/18' or '12/31/2018').
- Floating point literal is a number in scientific notation. A positive or negative exponential value follows the letter E. The mantissa has at least one digit and can have a decimal point. The entire literal cannot be less than nine characters long (for example, +6.2E-004) or more than 23 characters long (for example, -2.671837293847383E+012). You place the decimal point as appropriate, relative to the value of the exponent. For example: 1.5385E+6
- Money literals must use a dollar sign, commas, two decimal places, and decimal point when necessary (for example, $6,838,383.99).
- Number literals (a series of digits) must not use commas (for example, 8374623).
- Rate literal must have a decimal point, where appropriate. The rate literal can have no more than three digits to the right of the decimal point, and must have at least one digit to the left of the decimal (for example, 5.4, 5.40, 5.400).

Formatting Conventions

Just as you put comments in your specfile to help another person understand what your specfile is doing, there are also certain conventions for formatting a specfile that help other users read it and understand its logic.

- Use all CAPITAL LETTERS except for information that is printed on your report or displayed on your screen. For example:

```
PRINT TITLE="Negative Balance Shares"
```

- Use one space between keywords like PRINT and TITLE, as shown previously, but don't use spaces on either side of an arithmetic (+, -, *, /) or relational (<, >, =, <=, >=) operator. For example:

```
SHARE:TYPE<=2
AGGBAL=SHARE:BALANCE+AGGBAL
```

- Always leave one blank line after the end of each division. For example:

```
TARGET=ACCOUNT

 SELECT
  ACCOUNT:OPENDATE='--/--/--'
 END
```

- Use a one-space indentation to differentiate the contents of a division from its END. For example:

```
DEFINE
  AGGSHAREBAL=MONEY
  AGGLOANBAL=MONEY
 END
```

- Use additional one-space indentations to differentiate IF…THEN and ELSE clauses from other commands in the same division. For example:

```
PRINT TITLE="Zero or Negative Shares"
  PRINT SHARE:ID
  IF SHARE:BALANCE<$0.00 THEN
   PRINT "Zero"
  ELSE
   PRINT SHARE:BALANCE
 END
```

- Use minimum one-space indentations with DO…END clauses and their contents to highlight the statements that are logically executed within each DO/END pair. For example:

```
PRINT TITLE="Tier 2 Shares"
  PRINT ACCOUNT:NUMBER
  NEWLINE
  FOR EACH SHARE
   DO
      PRINT SHARE:ID
      PRINT " "
      IF SHARE:BALANCE>$2,500.00 THEN
       PRINT "TIER 2"
      NEWLINE
   END
 END
```

- If any statement or expression in your specfile is too long to fit on one line, insert a break at a natural point and continue on the next line. For example:

```
IF ACCOUNT:CLOSEDATE='--/--/--' AND
```

```
   SHARE:CLOSEDATE='--/--/--' THEN...
```

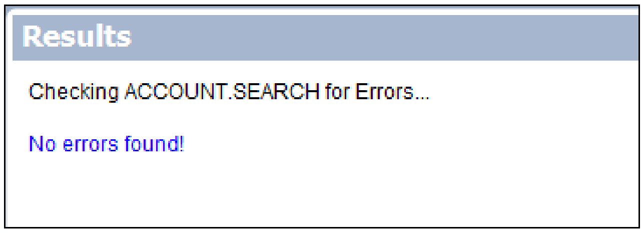
Checking a Specfile for Errors

Once you have named your specfile, written it, and formatted it according to the proper conventions, your final step is to ensure that it contains no obvious syntax errors which will prevent the system from running it.

Steps

1. Click **Save** on the PowerOn Control toolbar.
   If this is the first time you have ever saved the specfile, the *Save Specfile As* window appears.
2. Enter the name of your specfile in the **File Name** field, and then click **Select**.
3. Click **Check a Specfile for Errors**.

   If the specfile syntax is correct, a message such as the following appears:

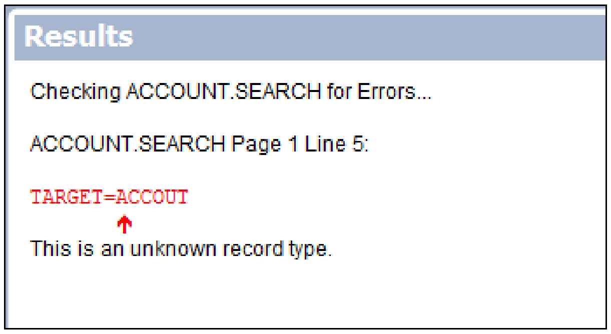
   

   Results

   ```
   Checking ACCOUNT.SEARCH for Errors...

   No errors found!
   ```

   If the specfile syntax is not correct, the system does the following:

   Results

   ```
   Checking ACCOUNT.SEARCH for Errors...

   ACCOUNT.SEARCH Page 1 Line 5:

   TARGET=ACCOUT
            ↑
   This is an unknown record type.
   ```

   Symitar finds and displays one error at a time.
4. For each error the checker finds, edit the lines as needed. Click **Save** when editing is completed.
5. Repeat the checking procedure, revising mistakes until you receive the *No Errors...* message.

   CAUTION
   A specfile that has no errors will not necessarily work the way you expect. The system can find places where you have used improper keywords or punctuation or have written an incomplete statement

or expression, but it cannot determine whether a statement or expression will function as you intend.

PRINT, NEWLINE, and TITLE

Before you write your first specfile, you need to know how to print data. There are two different ways to print data on a report: non-columnar and columnar format.

In this lesson, you will learn how to use the PRINT statement option, which lets you print out data in non-columnar format. You will also learn two other essential print statements: NEWLINE and TITLE.

Objectives

At the end of this lesson, you will be able to:

- Write a specfile which uses the following divisions:
  - TARGET
  - SELECT
  - PRINT
- Use the following print statement options:
  - PRINT
  - NEWLINE
  - TITLE

Sample Specfile

First, let's look at a sample specfile that uses all of these statements so you can see them in action. The following specfile generates a report that lists all member names and addresses in standard address format.

```
TARGET=ACCOUNT

SELECT
 ACCOUNT:CLOSEDATE='--/--/--'
END

PRINT TITLE="Open Account Listing"
 PRINT NAME:LONGNAME
 NEWLINE
 PRINT NAME:STREET
 NEWLINE
 PRINT NAME:CITY
 PRINT ", "
 PRINT NAME:STATE
 PRINT "  "
 PRINT NAME:ZIPCODE
 NEWLINE
 NEWLINE
END
```

The resulting report looks like this:

```

EDWIN JAMES
1485 RAINPATTERN COURT
PITTSBURGH, PA  15632

BELINDA ROGERS
6 FRONT STREET
DANA POINT, CA  92629

MEGAN JENSEN
1219 FALLING WATER
OAK PARK, IL  60357
```

PRINT Definition and Syntax

The PRINT statement tells PowerOn® to print the contents of a data item (or any text you enclose in quotation marks) beginning in the next available column. When you use the PRINT statement, all data types are left justified.

The syntax for PRINT is as follows:

```
PRINT data item/"text"
```

The PRINT option is useful when you want to place the parts of an address on one line with specific spacing and punctuation between them. For example, if you write the following code:

```
PRINT NAME:CITY
PRINT ", "
PRINT NAME:STATE
PRINT "  "
PRINT NAME:ZIPCODE
```

The line of the report is printed as follows:

```
CHULA VISTA, CA  91913
```

This is useful because you do not need to know how many characters are in each field to space the information properly.

NEWLINE Definition and Syntax

The NEWLINE statement tells PowerOn to insert a carriage return and continue any additional PRINT or COL statements on the next line. You must provide a carriage return after printing 132 characters. If you attempt to print more than 132 characters on a single line, your specfile will pass the error-checker but will terminate when you try to run it.

PowerOn automatically inserts a carriage return whenever it begins processing a new target record. If your specfile prints only one line of data for each target record, you do not need to add any NEWLINE statements unless you want to add blank lines between the target records.

The syntax for NEWLINE is…

```
NEWLINE
```

For example, if you write the following code

```
PRINT NAME:LONGNAME
NEWLINE
PRINT NAME:STREET
NEWLINE
PRINT NAME:CITY+", "+NAME:STATE+"  "+NAME:ZIPCODE
NEWLINE NEWLINE
PRINT "DEAR "+NAME:TITLE+" "+NAME:LAST+":"
```

The output looks like this

```
MR. HOWARD G. PARKER
2121 FIRST AVE
SAN DIEGO, CA  92101

DEAR MR PARKER:
```

The code also illustrates the concept of character addition. If you are printing a series of items and all of them have a CHARACTER data type, you can string them together with plus symbols (+) and print them all using a single PRINT statement.

TITLE Definition and Syntax

The PRINT and NEWLINE statements previously described may appear in some specfiles and not in others. You can use the ones you prefer to lay out information on your report the way you want it. There is, however, one statement that you must use in every single specfile when you create your PRINT division: the TITLE statement.

When you start the PRINT division, you must always follow the word PRINT with the word TITLE= and a name for your report enclosed in quotation marks, as shown here:

```
PRINT TITLE="My Report"
```

This format tells PowerOn that this occurrence of the word PRINT marks the beginning of the PRINT division. Otherwise, PowerOn cannot distinguish between the PRINT division marker and the PRINT statement.

For batch specfiles, the title you assign to your report is the name that appears in the print queue when you access the report in Print Control. The title is also printed in the top center of the report itself, provided that you set up headers. (We will talk about creating headers in the next chapter.)

For demand specfiles, the title of the report does not appear in the output, but you still need to assign a title to the report.

Activity: PRINT, NEWLINE, and TITLE

Specfile name: <initials>.NAMES

Your Marketing department wants a report that lists the long name, street address, city, state, and Zip Code of every member with an open account. Between each item, they would like you to print a comma (,).

They will use FTP to transfer the resulting report to a PC and upload it to an Excel® spreadsheet. The commas between the fields allow Excel to determine the column the data will be placed in.

You will need these record types and mnemonics:

- ACCOUNT:CLOSEDATE
- NAME:LONGNAME
- NAME:STREET
- NAME:CITY
- NAME:STATE
- NAME:ZIPCODE

The resulting report looks like this:

```
PHILLIP JOHNSON,1485 RAINBOW COURT,PITTSBURGH,PA,15632
BETSY ELISSA CLARK,606 D STREET,DANA POINT,CA,92629
CHAD JOHN FAIRMAN,1219 FALLING WATER,OAK PARK,IL,60357
BELINDA FOWLER,1827 NANTUCKET AVE,SAN DIEGO,CA,92104
```

Once you are finished writing your specfile and checking it for errors, be sure to create or locate some test data to ensure that your specfile is working properly.

Field Values

When you write specfiles, you almost always need to get information from fields in the database.

To use the database effectively, you need to know three things about every field you use in your specfiles:

- Its mnemonic
- Its data type and size
- Its valid values and their meanings

Objectives

At the end of this lesson, you will be able to:

- Define the following terms:
    - Record type
    - Mnemonic
- Perform a field name inquiry
- Determine which mnemonic to use when referencing a specified field in a specified record
- For each field in the system, determine:
    - Data type and size
    - Valid values and their meanings

Record Types and Mnemonics

Each field within a record must have a unique name. In some cases, several fields may share a name, but each has a different subfield number to distinguish it. For example, there are eight **Service Code** subfields in the Share record: **SERVICE CODE:001**, **SERVICE CODE:002**, and so on.

That said, different records often contain fields with the same name. For example, both the Loan record and the Share record have a **Balance** field. This means that whenever you refer to a field, you must provide PowerOn with the complete “address” of that field. A field’s address consists of the record type and the field mnemonic.

- The record type is the name of the record followed by a colon. For example, **SHARE:** is the record type for a Share record.
- The mnemonic is an easy-to-remember character ID for a data field. For example, the mnemonic for the **Social Security Number** field in the Name record is **SSN**.

When you refer to a field in a specfile, you combine the record type and the mnemonic. Some examples are:

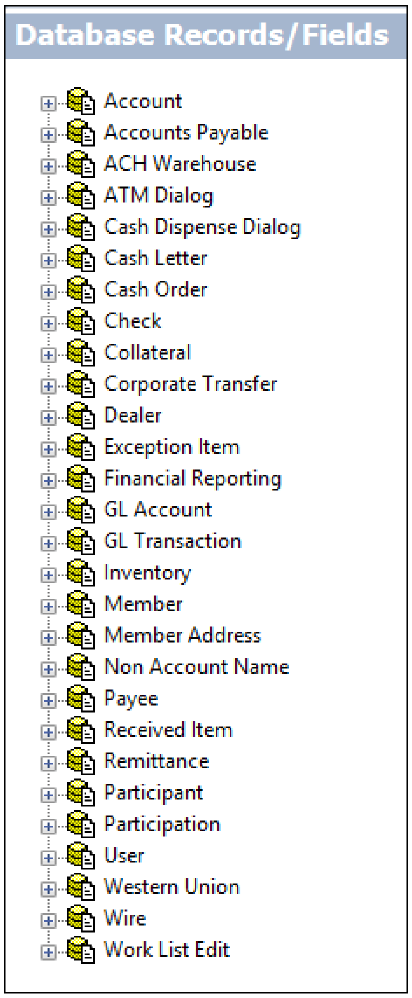
```
SHARE:BALANCE
LOAN:BALANCE
LOAN TRANSFER:AMOUNTOPTION
GLACCOUNT:NUMBER
```

Performing a Field Name Inquiry

You can use the **Field Name Inquiry** function in the PowerOn Control work area to display or print mnemonics and data types.

Steps

1. Click **View Field Names**.
   The *Database Records/Fields* pane appears.

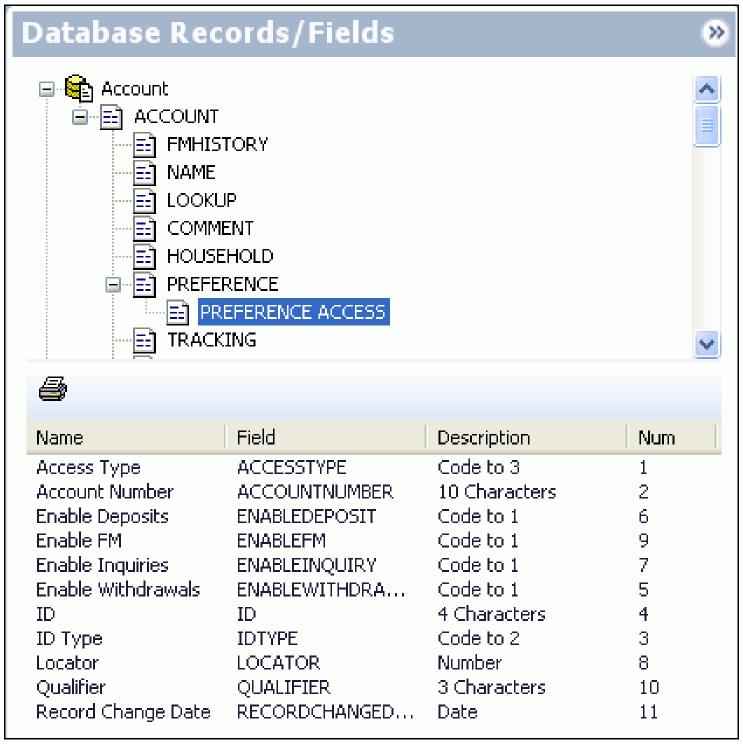


Database Records/Fields

- Account
- Accounts Payable
- ACH Warehouse
- ATM Dialog
- Cash Dispense Dialog
- Cash Letter
- Cash Order
- Check
- Collateral
- Corporate Transfer
- Dealer
- Exception Item
- Financial Reporting
- GL Account
- GL Transaction
- Inventory
- Member
- Member Address
- Non Account Name
- Payee
- Received Item
- Remittance
- Participant
- Participation
- User
- Western Union
- Wire
- Work List Edit

2. To display specific fields by their mnemonics and descriptions, do the following:
    a. Click the **+** (plus sign) at the left of the name of the file from which you want to display fields.
    b. Continue clicking the **+** for each record or subrecord until you reach the record whose fields you want to display.
    c. Select that record name.
    d. The fields from that file or record appear in the lower right pane of the screen, below the file listings.

    This pane lists each field in the record by name, field, description, and number. The example below displays the fields in the Preference Access record:



Name	Field	Description	Num
Access Type	ACCESSTYPE	Code to 3	1
Account Number	ACCOUNTNUMBER	10 Characters	2
Enable Deposits	ENABLEDEPOSIT	Code to 1	6
Enable FM	ENABLEFM	Code to 1	9
Enable Inquiries	ENABLEINQUIRY	Code to 1	7
Enable Withdrawals	ENABLEWITHDRA...	Code to 1	5
ID	ID	4 Characters	4
ID Type	IDTYPE	Code to 2	3
Locator	LOCATOR	Number	8
Qualifier	QUALIFIER	3 Characters	10
Record Change Date	RECORDCHANGED...	Date	11

Name:

The name of the field.

Field:

The mnemonic the system uses to identify each field in a specfile.

Description:

A description of the field's data type. The data type can limit a field's use in a specfile, and you must know the data type when creating a specfile that uses the field.

Number

The field number. This can be used in place of a mnemonic as a field reference in a specfile.

Finding Field Values

You must know which fields in which records contain the information you need for your report before you write your specfile. You must also know the data type of those fields and what the valid values mean.

Steps

1. Determine which record is most likely to contain the information you need. If you need information about shares, try the Share record first. If it doesn't contain what you need, try its child records (Hold, Name, etc.).
2. Use the View Field Names function in the PowerOn Control work area for the record in step 1. Make note of the mnemonic and data type of each field you think may contain the data you need.
3. Use the file maintenance work area (Account Manager, General Ledger Manager, etc.) to create or revise the record chosen in steps 1 and 2, and select the field on your screen.

    - If a selection box with a list of options appears, the field is probably a code field with values preset by Symitar. To determine the code values which correspond to the descriptions listed, refer to Symitar eDocs (see step 4).
    - If a select box does not appear, check to see if there is any user-defined help for the field. If your credit union has a help file for this field, it appears and should list the valid values and their meanings. To display the help file, position your cursor on the field or prompt and then press the **Shift+F1** keys simultaneously to view credit union-defined help.
4. 4. Look up the information in *Symitar eDocs*.
    a. Click **Help** on the Symitar toolbar. Symitar eDocs appears.
    b. Expand the Programming option in the menu on the left-hand side of the screen.
    c. Expand *File, Records, and Fields*.
    d. Continue clicking **+** for each data file or record until you find the record that contains the necessary information. Select that record from the list. The corresponding publication is displayed in the right pane.

Activity: Name That Field!

Your instructor will display a series of questions about field values on the screen. When you're ready to answer the question, raise your hand.

The COL Statement

When you wrote your first specfile, you used the PRINT statement to print data items on the report in non-columnar format. In this lesson, you will learn the COL statement, which prints data in a columnar format.

Objectives

At the end of this lesson, you will be able to:

- Use the COL print statement option
- Explain which data types are left justified and which are right justified by default
- Override the default justification for a data item
- Explain the difference between the share type and the share ID

Sample Specfile

First, let’s look at a sample specfile that uses both the COL and PRINT statements so that you can see the difference between them. The first line of the resulting report provides the member’s account number, full name, and the account open date in columnar format. The second line provides the member’s address.

```
TARGET=ACCOUNT
SELECT
 ACCOUNT:CLOSEDATE='--/--/--' AND
 (SHARE:BALANCE<>$0.00 OR
  LOAN:BALANCE<>$0.00)
END
PRINT TITLE="Open Account Listing"
 COL=001 ACCOUNT:NUMBER
 COL=013 NAME:LONGNAME
 COL=064 ACCOUNT:OPENDATE
 NEWLINE
 COL=004 "Address: "
 PRINT   NAME:STREET+", "+NAME:CITY+ ", "+
         NAME:STATE+"  "+NAME:ZIPCODE
 NEWLINE
END
```

The resulting report looks like this:

```
0000018983  BILL JAMES                                  07/22/93
   Address: 1485 RAINPATTERN COURT, PITTSBURGH, PA  15632
0000123440  BELINDA ROGERS                              11/21/94
   Address: 606 D STREET, DANA POINT, CA  92629
0000123444  MEGAN JENSEN                                07/22/93
   Address: 1219 FALLING WATER, OAK PARK, IL  60357
```

COL Definition and Syntax

COL tells the system to print the contents of a data item (or any text you enclose in quotation marks) in a certain column on the report. You can indicate whether you want the data to start in that column (left justified) or end in that column (right justified). If you do not specify justification, the system automatically left justifies character data and right justifies all other types of data.

The syntax for COL is:

```
COL=column# LEFT/RIGHT data item/"text"
```

For example, if you write this:

```
COL=001 "Type"
COL=006 LEFT SHARE:TYPE
COL=030      SHARE:BALANCE
```

Your report looks like this:

```
123456789012345678901234567890
------------------------------
Type 0010             6,000.00
Type 0050             4,672.32
Type 0060                 5.25-
```

Codes, Types, IDs

One of the most important things you must understand when writing specfiles which use information from Share or Loan records is the difference between the **Share Code** or **Loan Code**, **Type**, and **ID**. Since the specfile you will write for this lesson looks at Share records, we need to discuss this distinction now.

> **IMPORTANT**
> The following information applies to both Share and Loan records.

Shares and loans can be selected by code, type, and ID. The **Share Code** or **Loan Code** is a very general classification, while **Type** gets more specific about the share or loan detail, and finally the **ID** is the most specific grouping you can use.

Share/Loan Codes and Types

When you create any new Share or Loan record, Symitar prompts you to select the Type. The type specifies which template Symitar uses to fill out default values for fields in the record and which screens appear when you create or revise the record. There are up to 10,000 share and loan types (0000–9999).

Before Release 20.01, there were only 100 possible types for shares and loans (0–99). There is now a Miscellaneous Parameter which determines whether types are two or four digits. This parameter is set to two digits by default. Once this parameter has been set to four digits, it cannot be changed back to two digits without the assistance of a Jack Henry programmer. In this class, the parameter is set to four digits.

The Symitar software lets your credit union create these record templates (called default records) to reduce user input errors and ensure that the fields which control how Symitar processes a specific record are properly set.

For example, the **Share Code** field tells Symitar whether the share is a regular savings share, a draft share (which allows checks to clear against that share), a certificate share (with a term and maturity date), or a club share.

Likewise, the **Loan Code** field tells Symitar whether the loan is a closed-end loan, open-end loan, line of credit, credit card, or lease.

Rather than leaving it up to each user to set the **Share Code** field or **Loan Code** field properly when creating a new share or loan, the default record for the share type sets this field automatically.

Similarly, each share type has its own creation screen. This creation screen contains only those fields a user needs to set for a specific kind of Share or Loan record.

For example, a certificate must have its **Maturity Date** field set manually (Symitar calculates the value based on the specified term when your cursor moves into the field). As a result, this field must appear on the creation screen for a certificate share type.

What is an ID?

A share or loan ID is a unique two-character (00–ZZ) or four-character (0000–ZZZZ) identifier for a specific Share or Loan record. Your credit union determines whether IDs are two characters or four characters in length by having a Jack Henry programmer log in to your system and set a system parameter to indicate whether your IDs are two or four positions long. If your credit union does nothing, your IDs will be two characters long. In this class, we will be using a directory in which the IDs are four characters long.

The ID, once assigned to a particular share or loan, can't be changed. The only way to assign a different ID is to close the share or loan with the incorrect ID and open a new one with the correct ID.

The ID should not be confused with the type, although in some cases the type and ID for a particular record may use the same digits.

> **Example:**
> Suppose the **Loan Type** for closed-end loans is **50** and the range of numbers for loan IDs for closed-end loans is 50–59. When you create the first closed-end loan in a member's account, you assign a **Loan Type** of **50** and a **Loan ID** of **50**. A few months later, the member comes back to open a new closed-end loan, but doesn't want to close the original loan. You open a second closed-end loan and assign a **Loan Type** of **50** and a **Loan ID** of **51**.

Activity: The COL Statement

Specfile name: <initials>.COL

Your Member Services department needs a report which lists all open certificate shares. They would like to see the following information on the report:

- The account number
- The member's name in LAST,FIRST MI format
- The **Share ID**
- The **Share Type**
- The **Maturity Date**
- The balance of each certificate

It is up to you to determine the mnemonic, data type, and data size for each of these fields. The resulting report looks like this:

```
0000123441  FOWLER,BELINDA J   0020   0020     08/22/10     3,000.00
0000123441  FOWLER,BELINDA J   0070   0070     02/27/10     5,163.34
0000123442  FOWLER,V. M        0020   0021     04/09/11     3,269.30
0000123456  DOW,TOM            0020   0020     04/20/12     3,274.40
0000123461  RILEY,KATHY C      0020   0020     03/15/11     2,786.92
0000123462  JOHNSON,DONNA M    0020   0020     07/17/11     2,976.35
0000123466  DAVIS,STAN         0020   0020     10/10/13     2,682.23
0000123474  GARCIA,JARED       0070   0070     02/27/10    15,629.65
```

When you've finished writing the specfile, use the **Check a Specfile for Errors** function to verify that you have not made any glaring syntax errors.

- When the specfile has no errors, create or revise records in the database to test your selection criteria.
- When your test data is complete, go to the Batch Control work area and run your specfile. When the job is finished, go to the print queue and view your results. You may need to adjust some spacing to get your columns to line up properly.

Hints

- Be sure to use your initials in the title of your report. Otherwise, you may not be able to distinguish it from anyone else's in the print queue.
- Remember when you space your report that when you use COL, all data types except character are right justified unless you specify otherwise.
- Do not try to set up any headers for this report. You will learn to do this in the next lesson.
- Include plenty of space between columns.

Headers, Trailers, and REPEATCHR

Now that you have written a couple of specfiles without assistance from EasyWriter, you need to learn a few additional statement options to add headers and trailers to the resulting report. You will also learn the REPEATCHR function, which can make it easier to create dashed separator lines.

Objectives

At the end of this lesson, you will be able to:

- Explain why you should add headers to batch reports
- Use the following print statement options:
    - HEADER=
    - HEADERS…END
    - TRAILERS…END
- Use the REPEATCHR function

What Headers Do for You

Headers are used on batch reports to provide a label for each column. When you create headers for a report, the headers are repeated at the top of each page.

Headers also ensure that your report has each of the following:

- A title line at the top
- A column heading line
- A form feed at the end of the report
- Page breaks

There are two ways to create headers in a specfile:

- The HEADER print statement option
- The HEADERS … END subdivision

You cannot mix and match these options; you must use either one or the other. Each method has advantages and disadvantages. The method you use depends on your preferences and the type of report you are creating.

Sample Specfile

First, let's look at two versions of a specfile which uses headers, trailers, and the REPEATCHR function. The specfile lists the account number, loan type, ID, and balance of all open loans in the system and uses the HEADERS…END and TRAILERS…END subsections to create the headers and trailers.

```
TARGET=LOAN

SELECT
 LOAN:CLOSEDATE='--/--/--'
END

PRINT TITLE="Loan Balances"
 HEADERS
  COL=001       "Account #"
  COL=018 RIGHT "Type"
  COL=021       "ID"
  COL=040 RIGHT "Balance"
  NEWLINE
  COL=001 REPEATCHR("-",40)
 END
 TRAILERS
  COL=001 REPEATCHR("=",40)
  NEWLINE
 END
 COL=001 ACCOUNT:NUMBER
 COL=018 LOAN:TYPE
 COL=021 LOAN:ID
 COL=040 LOAN:BALANCE
END
```

The resulting report looks like this:

```
Symitar CU                      Loan Balances

Account #     Type   ID          Balance
----------------------------------------
0000123441    0080   0080         983.76
0000123456    0060   0061       1,517.20
0000123466    0060   0060         199.33
========================================

Totals for Entire Report
Selected Record Count:                                 42
```

Here is the same specfile again, using the HEADER= print statement option to create the headers and trailers:

```
TARGET=LOAN

SELECT
 LOAN:CLOSEDATE='--/--/--'
END

PRINT TITLE="Loan Balances"
               [123456789012345678901234567890123456789012345678901234567890123456789]
 HEADER="Account #   Type  ID             Balance"
 HEADER=REPEATCHR("-",40)
 COL=001 ACCOUNT:NUMBER
 COL=016 LOAN:TYPE
 COL=019 LOAN:ID
 COL=040 LOAN:BALANCE
 NEWLINE
END
```

The resulting report looks like this:

```
Symitar CU                      Loan Balances

Account #     Type   ID          Balance
----------------------------------------
0000123441    0080   0080         983.76
0000123456    0060   0061       1,517.20
0000123466    0060   0060         199.33
----------------------------------------
Totals for Entire Report
Selected Record Count:                                         42
```

HEADER= Definition and Syntax

The HEADER= print statement option identifies a single line of text to be printed as column headings on your report. To line up the text in the headings with the columns below the headings, you must use spaces between the words. The syntax is:

```
HEADER=character expression
```

When you use the HEADER= print statement, the last header line you specify is also printed at the end of the report and after each subtotal. Symitar automatically inserts a return after each HEADER= line.

For example, if you write this code:

```
HEADER="Account #          ID        Balance"
HEADER="------------------------------------"
```

Your report looks like this (the Xs represent mock data):

```
Account #          ID        Balance
------------------------------------
xxxxxxxxxx         xxxx     x,xxx.xx
xxxxxxxxxx         xxxx       xxx.xx
xxxxxxxxxx         xxxx    xx,xxx.xx
------------------------------------
```

HEADERS … END Definition and Syntax

The HEADERS keyword marks the beginning of a subsection within which you place COL, PRINT, and NEWLINE statements to create the column headings for your report. You must indicate the end of the HEADERS subsection with the keyword END.

The syntax is as follows:

```
HEADERS
 print statements (COL, PRINT, NEWLINE)
END
```

The HEADERS subsection gives you more control than the HEADER= print statement. You can create as many lines of header information as you like, and you can easily align your headers with the data below the headings by using the COL statement to indicate exactly where you want each column heading to begin. Use the NEWLINE statement to add a carriage return after each line of header information.

The primary drawback of the HEADERS keyword is that it does not automatically repeat the last line at the end of the report or after subtotals. Instead, you can use the TRAILERS …END subsection which has the same syntax as the HEADERS… END subsection, to create this separator line. If you use the TRAILERS subsection, you must place it immediately after the HEADERS subsection.

For example, if you write this code:

```
HEADERS
 COL=001        "Account #"
 COL=014        "ID"
 COL=030 RIGHT  "Balance"
 NEWLINE
 COL=001 "------------------------------"
END
```

Your report looks like this:

```
Account #     ID         Balance
  --------------------------------
  XXXXXXXXXX    XXXX      X,XXX.XX
  XXXXXXXXXX    XXXX        XXX.XX
  XXXXXXXXXX    XXXX         XX.XX
```

TRAILERS … END Definition and Syntax

Like the HEADERS keyword, the TRAILERS keyword marks the beginning of a subsection in which you write statements that create a separator line between the detail lines and the end of your report. You must indicate the end of the TRAILERS subsection with the keyword END.

The syntax is:

```
TRAILERS
 print statements (COL, PRINT, NEWLINE)
END
```

You use the TRAILERS…END subsection to create the separator lines only if you use HEADERS…END subsection to create your headers. If you use the HEADER= print statement option, the last header line you define is repeated at the end of your report.

If you use the TRAILERS subsection, you must place it immediately after the HEADERS subsection, as shown here:

```
HEADERS
 COL=001        "Account #"
 COL=014        "ID"
 COL=030 RIGHT  "Balance"
 NEWLINE
 COL=001 "------------------------------"
END

TRAILERS
 COL=001 "=============================="
END
```

The resulting report looks like this:

```
Account #     ID         Balance
--------------------------------
xxxxxxxxxx    xxxx      x,xxx.xx
xxxxxxxxxx    xxxx        xxx.xx
xxxxxxxxxx    xxxx         xx.xx
================================
```

REPEATCHR Definition and Syntax

When you are setting up headers and trailers, you usually repeat a character (such as a dash or an asterisk) multiple times to create a separator line. The REPEATCHR function makes it easy for you to create these separating lines.

The REPEATCHR function lets you tell the system to repeat a specified character or string of characters a certain number of times.

The syntax for REPEATCHR is as follows:

```
REPEATCHR("character to repeat",# times)
```

For example, if you write this code:

```
PRINT REPEATCHR("*",10)
```

The system prints the asterisk (*) character 10 times on your report.

Activity: Headers, Trailers, and REPEATCHR

Specfile name: <initials>.HEADERS

Create a new specfile, and then copy the contents of <initials>.COL into the new file. Add headers and trailers to the existing code. If you did not finish your own version of <initials>.COL, you can copy the solution from the file called ED.COL.

The resulting report looks like this:

```
Account#    Member's Name       ID    Type   Mat Date    Balance
----------------------------------------------------------------
0000123456  DOW,TOM            0020   0020   04/20/18   3,274.40
0000123441  FOWLER,BELINDA J   0070   0070   02/27/19   5,163.34
0000123461  RILEY,KATHY C      0020   0020   03/15/19   2,786.92
0000123462  JOHNSON,DONNA M    0070   0070   07/17/18   2,976.35
0000123466  DAVIS,STAN         0020   0020   --/--/--   2,682.23
----------------------------------------------------------------
Selected Record Count:                                         5
```

Hints

- Do not forget that character data, such as column headings, is automatically left justified. If you want character data to be right justified, use the COL print statement option to indicate right justification.
- If you use the HEADERS … END subsection to set up your headers, you must use TRAILERS … END subsection to create the dashed line that separates the detail lines from the automatic record count the end of the report.

Standard Totals and Subtotals

When you write a specfile, it is nice to include totals and subtotals on the report. PowerOn provides built-in statements that let you do this without defining variables or writing a TOTAL division.

Objectives

At the end of this lesson, you will be able to do the following:

- Write a specfile that uses a SORT division.
- Use the SUBTOTAL sort key option.
- Use the TOTAL print statement option.

Sample Specfile

We are going to look at a specfile which uses a SORT division and standard totals and subtotals in three stages.

1. List of all open loans, sorted by **Loan Code**. (The **Loan Code** field has seven values: **(0) Closed End**, **(1) Open End**, **(2) Line of Credit**, **(3) Credit Card**, **(4) Lease**, **(5) Avg Daily Bal LOC**, and **(6) LOC Combination**.)
2. Provide subtotal breaks with counts after printing the detail of loans with each **Loan Code**.
3. Provide totals and subtotals of the **Loan Balance** field.

Example 1

```
TARGET=LOAN

SELECT
 LOAN:CLOSEDATE='--/--/--'
END

SORT
 LOAN:LOANCODE
END

PRINT TITLE="Loan Balances by Loan Code"
 HEADERS
  COL=001       "Account #"
  COL=016 RIGHT "Code"
  COL=022 RIGHT "Type"
  COL=024       "ID"
  COL=040 RIGHT "Balance"
  NEWLINE
  COL=001 REPEATCHR("-",40)
 END
 TRAILERS
  COL=001 REPEATCHR("=",40)
  NEWLINE
 END

 COL=001 ACCOUNT:NUMBER
 COL=016 LOAN:LOANCODE
 COL=022 LOAN:TYPE
 COL=024 LOAN:ID
 COL=040 LOAN:BALANCE
END
```

The resulting report is shown below:

```

Account #   Code  Type  ID          Balance
-------------------------------------------
0000123444     0  0000  0001      12,342.00
0000123475     0  0010  0010       9,823.22
0000123488     0  0001  0001         892.09
0000123458     2  0080  0081           0.00
0000123469     2  0080  0080         983.76
0000123456     3  0060  0061       1,517.20
0000123466     3  0060  0060         199.33
===========================================
Totals for Entire Report
Selected Record Count:                                 7
```

Example 2

In this version, we will add subtotal breaks to the report:

```
TARGET=LOAN
SELECT
 LOAN:CLOSEDATE='--/--/--'
END
SORT
 LOAN:LOANCODE SUBTOTAL="Subtotal for Loan Code:"
END
PRINT TITLE="Loan Balances by Loan Code"
 HEADERS
  COL=001       "Account #"
  COL=016 RIGHT "Code"
  COL=022 RIGHT "Type"
  COL=024       "ID"
  COL=040 RIGHT "Balance"
  NEWLINE
  COL=001 REPEATCHR("-",40)
 END
 TRAILERS
  COL=001 REPEATCHR("=",40)
  NEWLINE
 END

 COL=001 ACCOUNT:NUMBER
 COL=016 LOAN:LOANCODE
 COL=022 LOAN:TYPE
 COL=024 LOAN:ID
 COL=040 LOAN:BALANCE
END
```

The resulting report is shown below.

```

Account #   Code  Type  ID           Balance
--------------------------------------------
0000123458     2  0080  0081            0.00
0000123469     2  0080  0080          983.76
============================================
Subtotal for Loan Code:                                    2
Count:                                                     2
--------------------------------------------
0000123456     3  0060  0061        1,517.20
0000123466     3  0060  0060          199.33
============================================
Subtotal for Loan Code:                                    3
Count:                                                     2
--------------------------------------------
============================================
Totals for Entire Report
Selected Record Count:                                     7
```

Example 3

In the final version, we will add subtotals and totals for the Loan record **Balance** field:

```
TARGET=LOAN

SELECT
 LOAN:CLOSEDATE='--/--/--'
END

SORT
 LOAN:LOANCODE SUBTOTAL="Subtotal for Loan Code:"
END
```

```
PRINT TITLE="Loan Balances by Loan Code"
 HEADERS
  COL=001       "Account #"
  COL=016 RIGHT "Code"
  COL=022 RIGHT "Type"
  COL=024       "ID"
  COL=040 RIGHT "Balance"
  NEWLINE
  COL=001 REPEATCHR("-",40)
 END
 TRAILERS
  COL=001 REPEATCHR("=",40)
  NEWLINE
 END
 COL=001 ACCOUNT:NUMBER
 COL=016 LOAN:LOANCODE
 COL=022 LOAN:TYPE
 COL=024 LOAN:ID
 COL=040 LOAN:BALANCE TOTAL="Total Loan Balance:"
END
```

The resulting report is shown below:

```
Account #   Code  Type  ID            Balance
---------------------------------------------
0000123458     2  0080  0081             0.00
0000123469     2  0080  0080           983.76
=============================================
Subtotal for Loan Code:                                     2
Count:                                                      2
Total Loan Balance:                                    983.76
---------------------------------------------
0000123456     3  0060  0061         1,517.20
0000123466     3  0060  0060           199.33
=============================================
Subtotal for Loan Code:                                     3
Count:                                                      2
Total Loan Balance:                                  1,716.53
---------------------------------------------
==============================================
Totals for Entire Report
Selected Record Count:                                      7
Total Loan Balance:                                 20,501.49
```

TOTAL Definition and Syntax

The TOTAL print statement option identifies a monetary or numeric data item to be totaled and assigns a label to print on the report

The syntax is:

```
print statement TOTAL=character literal
```

The print statement is a statement like PRINT or COL, and the character literal is the string of text which will be used as the label on the report.

The totals accumulated by the TOTAL option are printed in a specific, standard format at the end of your report. For example, if you write the following:

```
PRINT TITLE="Loan Balances"
 COL=001 LOAN:ID
 COL=025 LOAN:BALANCE TOTAL="Total Loan Balance"
END
```

PowerOn prints one detail line consisting of the ID and balance of each selected loan. At the end of the report, the grand totals are printed in the following format:

```
Totals for Entire Report
Selected Record Count:                           247
Total Loan Balance:                       652,435.89
```

If you want to print a total in a format other than the standard format, you must define a variable or array and calculate its value using an arithmetic expression. You will learn to do this later in this course.

SUBTOTAL Definition and Syntax

Just as you can accumulate standard totals without defining a variable, you can also accumulate standard subtotals. You do so by using the SUBTOTAL option in the SORT division to indicate where you want the subtotal breaks to occur.

The SUBTOTAL sort key option lets you create standard subtotals by the specified field. The syntax is:

```
sort key SUBTOTAL=character literal
```

The sort key is the record type and mnemonic of the field at which you want to create your subtotal breaks, and the character literal is the string of text that will be used as the label for the subtotal on the report.

The SUBTOTAL sort key option works in conjunction with the TOTAL print statement option. If you specify a field to total in your PRINT division, Symitar automatically subtotals that field at each subtotal break.

For example, if you write this:

```
SORT
 ACCOUNT:NUMBER SUBTOTAL="Total for Account:"
END

PRINT TITLE="Loan Balances"
 COL=001 LOAN:ID
 COL=025 LOAN:BALANCE TOTAL="Total Loan Balance"
END
```

Then after the detail lines for each loan in a member's account, the subtotals for that account are printed in the following format:

```
Total for Account:                    0021232325
Selected Record Count:                         2
Total Loan Balance:                     2,435.89
```

At the end of the report, the grand totals are printed in the standard format described earlier.

If you want to print subtotals in a format other than the standard format, you must define a variable or array and calculate its value using an arithmetic expression.

Activity: Standard Totals and Subtotals

Specfile name: <initials>.STANDARD.TOTALS

Revise your share certificate specfile (use either ED.HEADERS or <initials>.HEADERS) as follows:

- Sort the report by share type and provide standard subtotals for each share type.
- Provide standard totals at the end of the report for the Share record Balance field.

The resulting report looks like this:

```
Account#   Member's Name      ID     Type   Mat. Date        Balance
----------------------------------------------------------------------
0000123456 DOW,TOM            0020   0020  04/20/2019      3,274.40
0000123461 SMITH,LAUREN C     0020   0020  03/15/2022      2,786.92
0000123462 JOHNSON,DONNA M    0020   0020  07/17/2024      2,976.35
0000123466 DAVIS,STAN         0020   0020  --/--/----      2,682.23
----------------------------------------------------------------------
Total for Share Type:                                            20
Count:                                                            4
Total Share Balance:                                      13,719.90
----------------------------------------------------------------------
0000123441 FOWLER,BELINDA J  0070   0070  02/27/2019      5,163.34
0000123474 RILEY,KATHY        0070   0070  02/27/2019     15,629.65
0000123479 KIRCHER,JOYCE      0070   0070  04/09/2018          0.00
----------------------------------------------------------------------
Total for Share Type:                                            70
Count:                                                            3
Total Share Balance:                                      20,792.99
----------------------------------------------------------------------
----------------------------------------------------------------------
Totals for Entire Report
Selected Record Count:                                            9
Total Share Balance:                                      38,782.19
```

Hints

- You will need to add a SORT division.
- If you forget to total the share balance, your report will have subtotal breaks with share counts, but no dollar amount subtotals or totals.

The SUPPRESS Statement

As you have learned, every specfile (except a validation specfile) must have a PRINT or LETTER division. This does not mean that you must print out detail information for every record your specfile selects for processing. You can, for example, write a PRINT division that suppresses detail lines and prints totals and subtotals only.

Objective

At the end of this lesson, you will be able to use the SUPPRESS statement.

Sample Specfile

Let's look at the specfile we used earlier to provide totals and subtotals of the Loan record **Loan Balance** field sorted by the **Loan Code** field. This time, the report provides only summary information and totals the **Loan Balance**, **Past Due Amount**, and **Interest Due** fields.

```
TARGET=LOAN

SELECT
 LOAN:CLOSEDATE='--/--/--'
END

SORT
 LOAN:LOANCODE SUBTOTAL="Subtotal for Loan Code"
END

PRINT TITLE="Summary Loan Balances by Loan Code"
 HEADER=""
 SUPPRESS LOAN:BALANCE TOTAL="Total Loan Balance:"
 SUPPRESS LOAN:PASTDUEAMOUNT TOTAL="Total Past Due:"
 SUPPRESS LOAN:INTERESTDUE TOTAL="Total Interest Due:"
END
```

The resulting report looks like this:

```
Symitar CU                                    Loan Balances by Loan Code

Subtotal for Loan Code                                                 0
Count:                                                                35
Total Loan Balance:                                           990,953.99
Total Past Due:                                                32,597.82
Total Interest Due:                                             9,385.06

Subtotal for Loan Code                                                 2
Count:                                                                10
Total Loan Balance:                                            16,206.40
Total Past Due:                                                 1,297.44
Total Interest Due:                                               465.97

Totals for Entire Report
Selected Record Count:                                                59
Total Loan Balance:                                         1,028,043.36
Total Past Due:                                                33,973.26
Total Interest Due:                                            10,001.06
```

SUPPRESS Definition and Syntax

The SUPPRESS statement prevents the printing of detail information and prints a total instead.

The SUPPRESS syntax is as follows:

```
SUPPRESS data item TOTAL=literal
```

The TOTAL option that follows the name of the data item tells the system to accumulate a total for that data item. This means that you do not need to write an arithmetic expression to perform the calculation or define a variable to hold the result. The literal that follows the TOTAL option provides a label for the totaled amount. The totals are printed at each subtotal break and at the end of the report.

For example, you can use the SUBTOTAL sort key option to sort by account number and write the following code in your PRINT division:

```
SUPPRESS LOAN:BALANCE TOTAL="Total Loans"
```

Your report looks like this:

```
Total for Account:       0004532396
Count:                            2
Total Loans:              10,345.04

Total for Account:       0004679653
Count:                            1
Total Loans:               6,783.98

Totals for Entire Report
Selected Record Count:            3
Total Loans:              17,129.02
```

CAUTION

When you use the SUPPRESS statement, be sure not to include any NEWLINE statements in your PRINT division. If you do, your report will have one blank line for each selected target record, even though it has no detail information.

Activity: The SUPPRESS Statement

Specfile name: <initials>.SUPPRESS

Create a report that selects only open certificate shares (if you want, you can start with ED.STANDARD.TOTALS or <initials>.STANDARD.TOTALS).

The report should list standard totals and subtotals by type for the Share record **Balance** and **Dividend From Open** (**DIVFROMOPEN**) fields.

The resulting report should look like this:

```
Subtotals for Share Type:                               0020
Count:                                                     8
Total Share Balance:                               25,649.67
Total Dividends from Open:                            426.81

Subtotals for Share Type:                               0021
Count:                                                     3
Total Share Balance:                               14,269.30
Total Dividends from Open:                              0.00

Totals for Entire Report
Selected Record Count:                                    14
Total Share Balance:                               60,711.96
Total Dividends from Open:                          1,135.65
```

Hints

- Make your headers section blank so that you get a title, page breaks, and form feeds, but no column headers.
- You do not need to print account numbers, member names, etc. The only fields you need are the Share record **Balance** and **Dividend From Open** fields.

Replace the COL or PRINT statements with the SUPPRESS statement to suppress detail for the field that follows. Be sure to use the TOTAL="label" statement to assign a label to the subtotals and totals for each field you suppress.

The LETTER Division

Using a LETTER division provides an easy method of creating letters or forms in batch or demand mode. You use the LETTER division in place of the PRINT division. They are not used together. You can think of the LETTER division as a very simplified PRINT division, but use of the LETTER division requires a somewhat different setup and organization of your specfile.

Objectives

At the end of this lesson, you will be able to:

- Use a LETTER division to produce a letter or form in batch and demand modes
- Use bats and mountains to place variable-length or fixed-length text in your letter or form
- Use the CAPITALIZE function
- Explain the limitations of the LETTER division

What is the LETTER Division?

The LETTER division holds a list of expressions to be merged into the body of a letter. Any number of expressions can be listed, but there must be a one-to-one relationship between the number of expressions in the list and the number of placeholders (known as bats and mountains) in the body of the letter. When you run the specfile, the system evaluates each expression found in the LETTER division in order of appearance and prints the result in the next available placeholder in the letter.

You must place the body of the letter or form on a new page of the specfile following the END keyword that ends the LETTER division. To create the page break, type `<!--NEWPAGE-->`.

Sample Specfile

The following specfile generates a letter for members who have at least one share with a balance greater than or equal to $5,000 and less than $20,000.

```
TARGET=ACCOUNT

SELECT
 SHARE:BALANCE>=$5,000.00 AND
 SHARE:BALANCE<$20,000.00
END

LETTER TITLE="Letters for Special Members"
 CAPITALIZE(NAME:FIRST)
 CAPITALIZE(NAME:LAST)
 CAPITALIZE(NAME:STREET)
 CAPITALIZE(NAME:CITY)
 NAME:STATE
 NAME:ZIPCODE
 CAPITALIZE(NAME:FIRST)
 ACCOUNT:NUMBER
 SHARE:ID
 SHARE:BALANCE
 $20,000.00-SHARE:BALANCE
END

<!--NEWPAGE-->

^+^ ^+^
^+^
^+^, ^+^ ^+^

Dear ^+^,

Your Account ^^^^^^^^^^ Share ID ^^^^ has a current
balance of $^+^. We thank you for your faith in
your credit union. With a deposit of only $^+^,
you can purchase a $20,000 Golden Eagle Certificate with
a special dividend rate. Please make your deposit soon!
Thank you,

Mike Davis
Your Credit Union CEO
```

The resulting letter looks like this:

```
Janice Perez
3554 West Lawndale Blvd.
San Jose, CA 95210

Dear Janice,

  Your Account 0000144599 Share ID 0000 has a current
balance of $15,000.00. We thank you for your faith in
your credit union. With a deposit of only $5,000.00,
you can purchase a $20,000 Golden Eagle Certificate
with a special dividend rate. Please make your deposit
soon!

Thank you,



Mike Davis
Your Credit Union CEO
```

LETTER Division Definition & Syntax

The LETTER division of a specfile contains the print instructions for a letter or standard form.

The syntax for LETTER is:

```
LETTER TITLE=character expression
    Expressions
END
```

For example:

```
LETTER TITLE="Monthly Letters"
 ACCOUNT:NUMBER
 SHARE:ID
 SHARE:BALANCE
END
```

^^^ Mountains Definition & Syntax

Mountains (^^^) are symbols used as placeholders for fixed-length data merged into a specfile. Each caret represents one character position.

The syntax for mountains is:

```
^^^
```

For example:

```
 Your Account Number is ^^^^^^^^^^
```

- Since the account number is 10 characters in length and is a fixed-length field, and the user wants all 10 characters to appear, you place 10 mountains (carets) in the positions where the actual account number should appear.
- You can use mountains only in specfiles that include a LETTER division.
- You should use mountains only to hold the place for data of known length. If the data is too long for the number of mountains you specify, the data appears in truncated form.
- You can use mountains only in the body of the letter, not in the LETTER division itself.
- The number of mountains corresponds to the fixed length of the data that replaces the mountains. Place the mountain symbols (^) in the exact columns where you want the data to appear. Mountains are commonly used to hold the position for character and date expressions, variables, and fields.

^+^ Bats Definition & Syntax

Bats (^+^) are symbols used as placeholders for variable-length data merged into a specfile. Each caret/plus sign/caret combination (^+^) represents one variable-length piece of data.

The syntax for bats is:

```
^+^
```

For example:

```
Dear ^+^,
You have $^+^ dollars in your share account.
```

- You can use bats only in specfiles that include a LETTER division.
- Place the first caret of the bat symbol in the column where you want the first character of the data to begin.
- You can use bats only in the body of the letter, not in the LETTER division itself.
- Use bats to hold the place for data of unknown length. Bats are commonly used to hold the place of character, money, rate, number, and code expressions, variables, and fields.

CAPITALIZE Definition & Syntax

The CAPITALIZE function takes a character item, capitalizes the first letter of each word, and lowercases every other letter. The function capitalizes the first letter of the item and each additional letter before which PowerOn finds a space.

The syntax for CAPITALIZE is:

```
CAPITALIZE(character expression)
```

For example, if you write this:

```
PRINT CAPITALIZE(NAME:STREET)
```

The system reformats `111 FIRST ST` to this:

```
111 First St
```

Be careful when using the CAPITALIZE function on the **Long Name** field. CAPITALIZE properly handles certain suffixes (such as II, III, etc.), but it does not properly capitalize other suffixes, such as MD or PhD (which leads to Dr. Mike Smith, Md or Dr. Delores Ali, Phd).

LETTER Division Limitations

There are several constraints in the use of the LETTER division:

- LETTER must be the first word of the LETTER division. You can use it only once in each specfile. Mark the end of the LETTER division with the word END.
- If you used a LETTER division, you cannot use PRINT as a division title in the same specfile.
- HEADER, HEADERS, or TRAILERS statements are not allowed in the LETTER division.
- There must be a one-to-one match between the expressions listed in the LETTER division and the placeholders that appear in the body of the letter.
- You cannot use a WHILE loop or a FOR EACH loop in the LETTER division.
- You must be sure to select all your records within the TARGET and SELECT divisions. If this is too constraining, use the PRINT division instead.

Activity: Using the LETTER Division

Specfile name: <initials>.LETTER

Your Loan department would like you to create a monthly delinquency notice. The notice should be generated for members who have a loan that is 1–30 days delinquent and should be a friendly reminder to the member that the loan is past due.

The resulting notice should look something like this (the merged data in the letter is underlined):

```
                                                        02/30/18
Dear Mr. West,

Your Account 0000003223 Loan 0032 is currently 16 days past due
with a past due amount of $212.60. Perhaps you have forgotten to
make your payment for this loan or mailed it late. Please review
your records and make the payment as soon as possible, or let us
know if there is some error on our part. You can call us toll-free
at 1-888-SYMITAR and speak to any Member Service Representative.
At Symitar Credit Union, we value your continued membership!


                                        MR. KEVIN WEST III
                                        6250 BERMUDA ISLAND DRIVE
                                        KEY WEST FL 32887-9852
```

Hints

- You can determine how many days a loan is past due by subtracting the date in the Loan record **Due Date** field from **SYSTEMDATE**.
- Don't forget to use `<!--NEWPAGE-->` to create a page break between the END of the LETTER division and the start of the body of the notice.
- The name and address should be in all uppercase letters with no punctuation.
- Be sure to create or identify appropriate test data before running your specfile in Batch Control!

Review Exercise 1

This exercise is a review of the following PowerOn statements and functions you have already learned.

- COL
- NEWLINE
- SUBTOTAL
- TOTAL
- SUPPRESS
- REPEATCHR
- CAPITALIZE

Activity: Review Exercise 1

Write a new specfile named <initials>.REVIEW1 which selects all open loans with a balance that is not $0.00.

- Sort by and provide subtotals by account number
- For each loan, print:
    - The description in uppercase and lowercase letters
    - The **Loan ID** and **Loan Type** field values
    - The **Open Date**, **Last Payment Date**, and **Due Date** field values
- At each subtotal break, and at the end of the report, print the total loan balance and past due amount. Don't print this information for each loan.

The resulting report should look something like this:

```
Symitar CU                                  Current Loan Status by Account

Description                  ID   Type  Open Date Last Pmt  Due Date
-------------------------------------------------------------------
New Auto 24 Months           0000 0000   10/27/08 02/01/18  03/01/18
Share Secured                0016 0005   04/16/06 02/16/18  03/16/18
Arm Type I 30 Years          0030 0030   06/18/02 02/01/18  03/01/18
===================================================================
Total for Account#:                                       0000123506
Count:                                                             3
Total Loan Balance:                                       759,043.07
Total Amount Past Due:                                          0.00
===================================================================
New Auto 24 Months           0000 0000   07/03/08 01/02/18  02/01/18
Share Secured                0016 0005   09/12/07 02/16/18  03/16/18
===================================================================
Total for Account#:                                       0000123508
Count:                                                             2
Total Loan Balance:                                        10,245.71
Total Amount Past Due:                                        242.43
===================================================================
Totals for Entire Report
Selected Record Count:                                            80
Total Loan Balance:                                     2,913,122.35
Total Amount Past Due:                                     65,916.88
```

Conditions

Sometimes you need to print different information on your report depending on the value of a field or variable. In this lesson, you'll learn how to write a conditional statement which performs a specified set of instructions if a condition is true.

Objectives

At the end of this lesson, you will be able to:

- Determine when to use a conditional statement in a specfile
- Use the IF…THEN statement in a specfile
- Use an ELSE clause to perform additional statements when necessary

Sample Specfile

Let's start by looking at a simple specfile which uses a conditional statement. This specfile lists all open credit card loans opened within the past 30 days. If the **Payment Method** field in the Loan record is set to **(2) Automatic Transfer**, the word `Transfer` is printed in the Payment Method column; otherwise, the words `Cash` or `Other` are printed.

```
TARGET=LOAN

SELECT
 LOAN:LOANCODE=3 AND
 LOAN:CLOSEDATE='--/--/--' AND
 LOAN:OPENDATE>=SYSTEMDATE-30
END

PRINT TITLE="Credit Cards Opened Within 30 Days"
 HEADER="Account#    ID  Payment Method"
 HEADER="-----------------------------"
 COL=001 ACCOUNT:NUMBER
 COL=013 LOAN:ID
IF LOAN:PAYMENTMETHOD=2 THEN
   COL=020 "Transfer"
 ELSE
   COL=020 "Cash or Other"
 NEWLINE
END
```

The resulting report looks like this:

```
Symitar CU             Credit Cards Opened Within 30 Days

Account#    ID      Payment Method
----------------------------------
0002392190  0010    Transfer
0002392190  0015    Cash or Other
0002495382  0001    Transfer
0002495382  0010    Transfer
0002495382  0030    Cash or Other
```

IF...THEN Definition

The IF … THEN statement and its variations are conditional statements which tell PowerOn to evaluate a Boolean expression, and then follow a specific course of action depending on the result.

For example, suppose you're writing a specfile that includes the member's address. You want to include any **Extra Address** information stored in the primary Name record, but you do not want a blank line to appear between the member's name and street address if there is no extra address information.

The IF … THEN statement lets you tell PowerOn to print the contents of the **Extra Address** field only if the field is not empty. If the field is empty, PowerOn moves onto the street address information without leaving a blank line.

It's worth noting at this point that you can print the mailing address for an account using the calculated **Payee Line 1** to **Payee Line 6** fields in the Account record. The advantage of using these fields is that Symitar takes into account issues like mail overrides and alternate addresses when calculating these fields. You can access these fields and print the same calculated address used by Symitar, as shown on the next page:

```
PRINT ACCOUNT:PAYEELINE:1
NEWLINE
PRINT ACCOUNT:PAYEELINE:2
NEWLINE
PRINT ACCOUNT:PAYEELINE:3
NEWLINE
IF ACCOUNT:PAYEELINE:4<>"" THEN
 DO
  PRINT ACCOUNT:PAYEELINE:4
  NEWLINE
 END
IF ACCOUNT:PAYEELINE:5<>"" THEN
 DO
  PRINT ACCOUNT:PAYEELINE:5
  NEWLINE
 END
IF ACCOUNT:PAYEELINE:6<>"" THEN
 DO
  PRINT ACCOUNT:PAYEELINE6
  NEWLINE
 END
```

IF…THEN Syntax

The following are examples of the format of IF…THEN syntax.

```
IF Boolean expression THEN
 statement

IF Boolean expression THEN
 DO
  statement
  statement
  statement
 END

IF Boolean expression THEN
 statement
ELSE
 statement

IF Boolean expression THEN
 DO
  statement
  statement
  statement
 END
ELSE
 statement

IF Boolean expression THEN
 statement
ELSE
 DO
  statement
  statement
  statement
 END

IF Boolean expression THEN
 DO
  statement
  statement
  statement
 END
ELSE
 DO
  statement
  statement
  statement
 END
```

The syntax you choose depends on how many statements you want to execute if the Boolean expression is true and how many statements (if any) you want it to execute if the Boolean expression is false. The examples on the next page should help you get a better sense of how each variation of IF … THEN is used.

IF...THEN Syntax Examples

The following are examples of IF...THEN syntax.

The following statement executes only one instruction when the condition is true. If the condition is false, Symitar goes on to the next instruction in the specfile.

```
IF LOAN:CHARGEOFFDATE<>'--/--/--' THEN
 COL=25 "ChargeOff"
```

The following statement executes multiple instructions when the condition is true. Symitar goes on to the next instruction in the specfile if the condition is false.

```
IF LOAN:CHARGEOFFDATE<>'--/--/--' THEN
 DO
  COL=25 "ChargeOff"
  COL=45 LOAN:CHARGEOFFDATE
 END
```

The following statement executes one instruction if the condition is true and another instruction if the condition is false:

```
IF LOAN:CHARGEOFFDATE<>'--/--/--' THEN
 COL=25 "ChargeOff"
ELSE
 COL=25 "Active"
```

The following statement executes multiple instructions if the condition is true and a single instruction if the condition is false:

```
IF LOAN:CHARGEOFFDATE<>'--/--/--' THEN
 DO
  COL=25 "ChargeOff"
  COL=45 LOAN:CHARGEOFFDATE
 END
ELSE
 COL=25 "Active"
```

The following statement executes multiple instructions if the condition is true and multiple instructions if the condition is false:

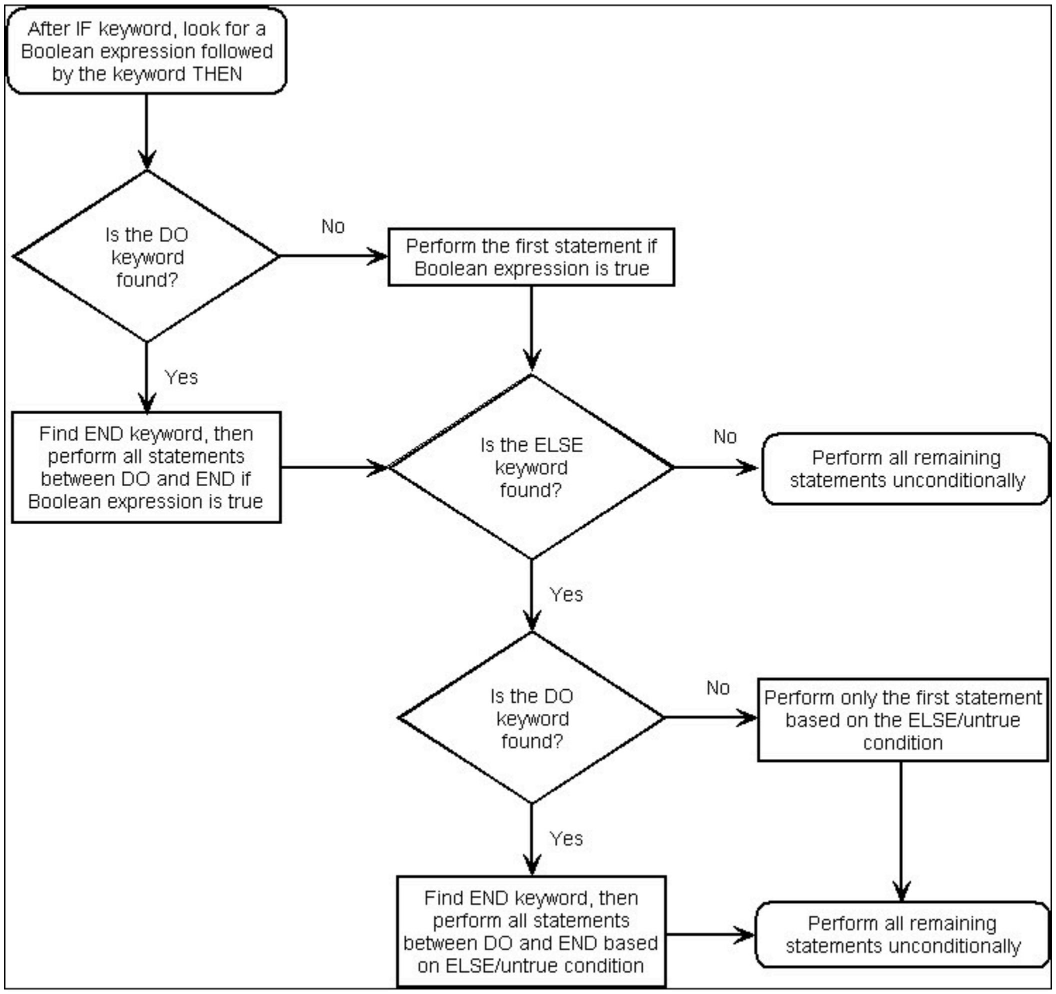
```
IF LOAN:CHARGEOFFDATE<>'--/--/--' THEN
 DO
  COL=25 "ChargeOff"
  COL=45 LOAN:CHARGEOFFDATE
 END
ELSE
 DO
  COL=25 "Active"
  COL=45 LOAN:LASTPAYMENTDATE
 END
```

The following example executes a single instruction if the condition is true and multiple instructions if the condition is false:

```
IF LOAN:CHARGEOFFDATE<>'--/--/--' THEN
 COL=25 "ChargeOff"
ELSE
 DO
  COL=25 "ACTIVE"
  COL=45 LOAN:LASTPAYMENTDATE
END
```

How PowerOn Evaluates IF…THEN Statements

The flowchart below shows how PowerOn evaluates the IF…THEN statement in a specfile.


After IF keyword, look for a Boolean expression followed by the keyword THEN
Is the DO keyword found?
No
Perform the first statement if Boolean expression is true
Yes
Find END keyword, then perform all statements between DO and END if Boolean expression is true
Is the ELSE keyword found?
No
Perform all remaining statements unconditionally
Yes
Is the DO keyword found?
No
Perform only the first statement based on the ELSE/untrue condition
Yes
Find END keyword, then perform all statements between DO and END based on ELSE/untrue condition
Perform all remaining statements unconditionally

Activity: Conditions

Specfile name: <initials>.LOAN.PORT1

We need a new specfile which produces a loan portfolio for the CFO:

- This report should list all open loans in Symitar.
- The CFO is interested in the status of the loan (current or delinquent). A loan should be considered delinquent if it's 10 or more days past due and there is an amount in the Past Due Amount field in the Loan record.
- The report should list the member's account number, name, the loan type, the loan's status (current or delinquent), the due date, the last payment date, and the balance of each loan.

The resulting report should look like this:

```
Symitar Credit Union                                        Loan Portfolio

Account#   Member's Name Type Status     Due Date Last Payment Balance
-----------------------------------------------------------------------
0000123444 SMITH,FRANK L 0000 Delinquent 04/14/18 01/11/18     9,900.00
0000123455 MARY,DODD     0002 Delinquent 08/01/18 07/01/18     5,263.68
0000123456 DOW,TOM       0002 Current    01/10/18 12/10/18     2,873.94
```

Hints

- You will need an IF … THEN statement to evaluate the loan's due date and past due amount to determine whether it is current or delinquent, and then print the appropriate status.
- Do not evaluate the due date or past due amount in the SELECT division. If you do, your report lists only delinquent loans, not all open loans.

Read Literals

Often, you need to gather information from the person running a specfile. You use a read literal to prompt a user for input in batch mode. You store the user's answer in a variable for use later in your specfile.

Objectives

At the end of this lesson, you will be able to do the following:

- Write a specfile that uses a SETUP division
- Define a variable
- Use the appropriate READ literal to prompt a user for input in batch mode

Sample Specfile

The following sample specfile selects open credit card loans that were opened within the past 30 days and that have a credit limit equal to or greater than an amount entered by the user.

```
TARGET=LOAN

DEFINE
 CREDLIMIT=MONEY
END

SETUP
 CREDLIMIT=MONEYREAD("Enter Lowest Credit Limit")
END

SELECT
 LOAN:LOANCODE=3 AND
 LOAN:CLOSEDATE='--/--/--' AND
 LOAN:OPENDATE>=SYSTEMDATE-30 AND
 LOAN:CREDITLIMIT>=CREDLIMIT
END

PRINT TITLE="Credit Cards Opened Within 30 Days"
 HEADER="Account#     ID     Payment Method"
 HEADER="----------------------------------"
 COL=001 ACCOUNT:NUMBER
 COL=013 LOAN:ID
 IF LOAN:PAYMENTMETHOD=2 THEN
   COL=020 "Transfer"
 ELSE
   COL=020 "Cash or Other"
 NEWLINE
END
```

The resulting report looks like this:

```
Symitar CU            Credit Card Within 30 Days

Account#    ID      Payment Method
----------------------------------
0002392190  0010    Transfer
0002392190  0015    Cash or Other
0002495382  0001    Transfer
0002495382  0010    Transfer
0002495382  0030    Cash or Other
```

Read Literal Syntax

To prompt for user input in a batch specfile, you use a READ literal. The READ literal you use depends on the data type of the information you are prompting the user for.

To prompt a user for data, you must define a variable to store the user's response in. You then use a READ literal in the SETUP division to place the user's response into the variable you defined.

- Use the CHARACTERREAD function to prompt for character data:

  ```
  CHARACTERREAD("prompt")
  ```

  For example:

  ```
  FIRSTNAME=CHARACTERREAD("First Name")
  ```
- Use the CODEREAD function to prompt for data that is stored in Symitar as code data. Although you must assign a data type of NUMBER to the variable that holds the data you read into your specfile, using the CODEREAD function limits the user's entry to a whole number 32767 or less.

  ```
  CODEREAD (“prompt”)
  ```

  For example:

  ```
  SELECTION=CODEREAD("Selection")
  ```
- Use the DATEREAD function to prompt for date data:

  ```
  DATEREAD("prompt")
  ```

  For example:

  ```
  STARTDATE=DATEREAD("Start Date")
  ```
- Use the MONEYREAD function to prompt for money data:

  ```
  MONEYREAD("prompt")
  ```

  For example:

  ```
  LOWBAL=MONEYREAD("Lowest Balance")
  ```
- Use the NUMBERREAD function to prompt for number date:

  ```
  NUMBERREAD ("prompt")
  ```

  For example:

  ```
  LOWTYPE=NUMBERREAD("Enter Lowest Share Type")
  ```
- Use the RATEREAD function to prompt for rate data:

  ```
  RATEREAD("prompt")
  ```

  For example:

  ```
  INTRATE=RATEREAD("Enter Interest Rate")
  ```

- Use the YESNOREAD function to prompt for a Y or N answer. This limits the user's input to Y, N, 1, or 0. If the user enters 1 or 0, Symitar changes the stored entry to Y or N respectively.

  The syntax is:

  ```
  YESNOREAD("prompt")
  ```

  For example:

  ```
  IF YESNOREAD("Okay?") THEN…
  ```

Additional Example

In the following example, the specfile selects loans for the report if the loan is open, has a value in the **Balance** field that is equal to or greater than the balance entered by the user, and has a value in the **Interest Rate** field that is equal to the rate entered by the user. Notice that the variables are defined first in the DEFINE division, and then the user's input is placed into the appropriate variable in the SETUP division.

```
TARGET=LOAN

DEFINE
 LOWBALANCE=MONEY
 INTRATE=RATE
END

SETUP
 LOWBALANCE=MONEYREAD("Lowest Loan Balance")
 INTRATE=RATEREAD("Interest Rate")
END

SELECT
 LOAN:CLOSEDATE='--/--/--' AND
 LOAN:BALANCE>=LOWBALANCE AND
 LOAN:INTERESTRATE=INTRATE
END
```

The previous example is not a complete specfile, because it does not include a PRINT division, but it shows how you can use PowerOn's ability to prompt users for input. As you can imagine, prompting users to enter data can save you from having to revise a specfile whenever minor criteria changes. You can also use this feature to write one specfile that serves multiple purposes.

Activity: Read Literals

Specfile name: <initials>.LOAN.PORT2

The CFO would like you to revise the loan portfolio specfile so that she can enter the date on or before which the loan should be considered past due. That way, she can run the report several times with different past due dates and compare the results.

You can read the ED.LOAN.PORT1 specfile into your new specfile or use your own specfile (called <initials>.LOAN.PORT1).

The resulting report should look like this:

```
Symitar Credit Union                          Loan Portfolio

Account#   Member's Name Type Status     Due Date Last Payment Balance
-----------------------------------------------------------------------
0000123444 SMITH,FRANK L 0000 Delinquent 01/15/18 12/11/17     9,900.00
0000123455 MARY,DODD     0002 Delinquent 02/01/18 01/01/18     5,263.68
0000123456 DOW,TOM       0002 Current    03/10/18 02/10/18     2,873.94
```

Hints

- Remember that dates in Symitar are evaluated to a number. Dates in the future are therefore greater than the system date, and dates in the past are less than the system date.
- Do not forget to define a variable in which to store the date entered by the user.
- Instead of comparing the loan’s due date to the system date minus 10, compare it to the value of the variable which holds the date entered by the user.

The FORMAT Function

When you prompt a user for information, you may want to include the information in the report's title. The title of your report must be made up entirely of character data. If you prompt the user to enter something other than character data, you need to use the FORMAT function to change it to character data.

Objectives

At the end of this lesson, you will be able to:

- Use the FORMAT function in a specfile
- Explain how to use each of the FORMAT function placeholders
- Explain why it is a good idea to format dates whenever you print them on a report or the screen

Sample Specfile

The following is an example of how the FORMAT function looks in a specfile. The title of the report now includes the credit limit entered by the user.

```
TARGET=LOAN

DEFINE
 CREDLIMIT=MONEY
END

SETUP
 CREDLIMIT=MONEYREAD("Enter Lowest Credit Limit")
END

SELECT
 LOAN:LOANCODE=3 AND
 LOAN:CLOSEDATE='--/--/--' AND
 LOAN:OPENDATE>=SYSTEMDATE-30 AND
 LOAN:CREDITLIMIT>=CREDLIMIT
END

PRINT TITLE="Credit Cards with Credit Limits>=$"+
            FORMAT("##,##9.99",CREDLIMIT)
 HEADER="Account#     ID     Payment Method"
 HEADER="-----------------------------------"
 COL=001 ACCOUNT:NUMBER
 COL=013 LOAN:ID
 IF LOAN:PAYMENTMETHOD=2 THEN
   COL=020 "Transfer"
 ELSE
   COL=020 "Cash or Other"
 NEWLINE
END
```

The resulting report looks like this:

```
Symitar CU         Credit Cards with Credit Limits>=$ 5,000.00

Account#     ID     Payment Method
----------------------------------
0002392190   0010   Transfer
0002392190   0015   Cash or Other
0002495382   0001   Transfer
0002495382   0010   Transfer
0002495382   0030   Cash or Other
```

FORMAT Definition and Syntax

The FORMAT function converts a numeric value into a predefined pattern of characters. This allows you to convert code, date, float, money, number, or rate data to character data so you can print it.

The syntax for the FORMAT function is as follows:

```
FORMAT("character pattern",numeric expression)
```

- The character pattern contains special contain characters that represent the “places” in the numeric expression.
- The numeric expression is either a specific non-character value or the name of a field, variable, or array that contains non-character data for conversion.
- Use the FORMAT function when you want to print or change non-character data to character data.

FORMAT Placeholders

The character pattern can contain any of the following special characters to represent the "places" in the numeric expression:

9	This is a placeholder for a digit. Symitar replaces each 9 with a digit from the numeric data. You should use this placeholder when you are sure that a number will always appear in this position in the numeric expression.
#	This is a placeholder for a digit that may or may not appear in each data item being converted. Symitar replaces each # with a digit from the numeric data, unless that position is blank or a leading zero.
+	This is a placeholder for a minus sign. If the value of the data item is positive, this position is blank when the value is converted to character data. If the value is negative, Symitar places a minus sign (-) in this position.
,	Symitar replaces commas with a blank space if the comma appears to the left of the first printed digit. Otherwise, the comma is printed in the location specified in the character expression.
/	Symitar copies slashes directly from the pattern to the result. For this reason, it is wise to format all four digits of the year, since January 1, 1902 and January 1, 2002 will both appear as 01/01/02 if you do not.

Any other characters you enter within the character expression's quotation marks are carried down to the printed data.

FORMAT Syntax Examples

The following are examples of FORMAT syntax.

The following FORMAT function converts a three-digit code field to character data, suppressing leading zeros:

```
FORMAT("##9",LOAN:APPROVALCODE)
```

The following FORMAT function converts a date to character data. Nothing is suppressed. Unless you are certain that the date you are formatting falls in the 1951-2050 range, you should format all four digits of the year, since FORMAT does not do this automatically.

```
FORMAT("99/99/9999",LOAN:APPROVALDATE)
```

The following FORMAT function converts money data to character data. Each digit to the left of the decimal point is printed only if a number exists in that position, and the commas are printed only if a number exists in the position to the left of the comma. A minus sign is printed to the right of the last digit if the amount is negative.

```
FORMAT("$#,###,##9.99+",SHARE:BALANCE)
```

When formatting money data, be sure to include sufficient # symbols. If the number of digits in the field exceeds the number of # symbols in the pattern, Symitar replaces all characters with an asterisk (for example: ***,***.**).

The following FORMAT function converts rate data to character data, suppressing leading zeros:

```
FORMAT("#9.999%",LOAN:INTERESTRATE)
```

You can also use the FORMAT function to convert float and number data. For more examples, see Symitar eDocs: **Programming** > **PowerOn** > **Lexicon** > **Function**.

When you convert numeric data to character data for use in a PRINT division title, you must use the following syntax:

```
PRINT TITLE="Share Types "+FORMAT("###9",STARTTYPE)+
            " to "+FORMAT("###9",ENDTYPE)
```

Activity: The FORMAT Function

Specfile name: <initials>.LOAN.PORT3

The CFO has realized that she cannot compare different reports unless she can tell which one was run with which date. Please change the title of the report to reflect the date entered by the user.

You can read the ED.LOAN.PORT2 specfile into your new specfile or use your own specfile (called <initials>.LOAN.PORT2) if you wrote one for the "Read Literals"lesson.

The resulting report should look like this:

```
Symitar Credit Union             Loan Portfolio (DQ as of 06/30/2018)

Account#   Member's Name Type Status     Due Date Last Payment Balance
-----------------------------------------------------------------------
0000123444 JAMES,EDWIN L 0000 Delinquent 02/15/18 01/11/18     9,900.00
0000123455 MARY,DODD     0002 Delinquent 01/01/18 12/01/17     5,263.68
0000123456 DOW,TOM       0002 Current    03/01/18 02/01/18     2,873.94
```

Boolean Rules of Precedence

When processing a specfile, PowerOn uses very rigid logic to determine the order in which it evaluates operations. This means that PowerOn does not always read from left to right, as you do. As a result, you must be careful when writing a specfile that you understand Boolean rules of precedence PowerOn users.

Objectives

At the end of this lesson, you will be able to:

- Use the following expressions and functions in a specfile, as needed:
  - ALL
  - AND
  - ANY
  - NOT or NOT ANY
  - OR
  - ANYSERVICE
  - ANYWARNING
- Explain the rules of precedence PowerOn uses to determine whether a condition is true or false.
- Use parentheses to override precedence when necessary.

Operators

An operator is a symbol or word that represents an operation performed following the rules of precedence. These operators can be used together or separately in an expression.

- You can use the following arithmetic operators:

  (+) Addition

  (-) Subtraction

  (*) Multiplication

  (/) Division
- You can use the following Boolean operators:
  - AND connects two expressions that must both be true for the entire expression to be true:

    ```
    LOAN:BALANCE>$0.00 AND LOAN:TYPE=10
    ```
  - NOT indicates that the Boolean condition is true if the operand or expression that follows it does not exist:

    ```
    NOT LOAN:TYPE=10
    NOT (LOAN:TYPE=10 OR LOAN:TYPE=20)
    ```
  - NOT ANY is a variation of NOT that lets you indicate that you do not want to select the target if a specific record or a record with a specific field value exists:

    ```
    NOT ANY LOAN TRANSFER
    NOT ANY CHECK WITH (CHECK:STATUS=2)
    ```
  - OR connects two expressions together, at least one of which must be true for the entire expression to be true:

    ```
    IF SHARE:TYPE=0 OR SHARE:TYPE=20
    ```
- You can use the following relational operators:

  = (equal to)

  > (greater than)

  < (less than)

  >= (greater than or equal to)

  <= (less than or equal to)

  <> (not equal to)

Other Expressions and Functions

The following are other Boolean expressions and functions.

- ALL is an expression that tells PowerOn to select every existing record of the target record type:

```
SELECT
 ALL
END
```

- ANY is a function that causes PowerOn to choose records of a specified type or records that have a field with a specified value:

```
ANY LOAN TRANSFER

ANY LOAN TRANSFER WITH
  (LOAN TRANSFER:EXPIRATIONDATE<>'--/--/--')

NOT ANY LOAN TRANSFER

NOT ANY LOAN:TRANSFER WITH
  (LOAN TRANSFER:AMOUNT<>$0.00)
```

- ANYSERVICE and ANYWARNING are functions that tell PowerOn to look through all the **Service Code** or **Warning Code** fields in the specified record and determine whether a specific value occurs in any of those fields. This saves you the trouble of writing eight ANY functions to look at each of the **Service Code** or **Warning Code** subfields.

```
SELECT
 ANYSERVICE(SHARE,10) OR
 ANYWARNING(LOAN,01)
END
```

- NONE is an expression that tells PowerOn not to select any records:

```
SELECT
 NONE
END
```

  The effect of NONE is that the PRINT division is executed only once. This is especially useful when you are processing a letter file or report file and do not want PowerOn to read through all the TARGET records before executing the PRINT division.

Rules of Precedence

A Boolean expression always evaluates to a true or false result. The order of precedence for determining whether the expression is true or false is as follows.

1. Apply unary minuses, if they exist (for example, -45).
2. Perform multiplication and division in the order they appear from left to right.
3. Perform addition and subtraction in the order they appear from left to right.
4. Evaluate the relational operators in the order they appear from left to right. (Relational operators are the symbols for less than [<], greater than [>], equals [=], does not equal [<>], etc.)
5. Evaluate the Boolean operators NOT, AND, and OR, in that order.

You can also use parentheses to control the evaluation order of the operations. (See *Using Parentheses to Control Precedence* for more information.)

If at any point PowerOn determines that the expression is false, it stops evaluating the expression and continues to the next instruction in the specfile.

Use Parentheses to Control Precedence

PowerOn evaluates expressions in parentheses in the order they are nested, beginning inside the innermost pair of parentheses. After that, PowerOn follows the standard rules of precedence to evaluate the remaining expressions.

For example, suppose you want a report that lists all shares that meet the following criteria:

- The share balance is greater than $0.00 or the available balance is greater than -$10.00.
- The Share **Type** field is set to 0-20, 50-60, or 70.
- The Share **Dividend Type** field is not set to 10.

Suppose that you write the following SELECT division:

```
SELECT
 SHARE:BALANCE>$0.00 OR
 SHARE:AVAILABLEBALANCE>-$10.00 AND
 SHARE:TYPE>=0 AND SHARE:TYPE<=20 OR
 SHARE:TYPE>=50 AND SHARE:TYPE<=60 OR
 SHARE:TYPE=70 AND
 NOT SHARE:DIVTYPE=10
END
```

According to the standard rules of precedence, PowerOn does the following:

1. Applies the -$10.00 first
2. Evaluates all relational operators in the order encountered
3. Applies the Boolean operator NOT to the SHARE:DIVTYPE=10 portion of the expression
4. Applies all AND operators in the order encountered
5. Applies all OR operators in the order encountered

If you carefully evaluate the expression according to the rules of precedence, you will discover that your SELECT criteria would try to select shares that meet one of the following criteria:

- A balance greater than $0.00
- An available balance greater than -$10.00, and the **Type** field is set to one of the following:
  - 0-20
  - 50-60
  - 70 with the **Dividend Type** field set to something other than 10

Now, by contrast, rewrite the SELECT division as follows:

```
SELECT
 (SHARE:BALANCE>$0.00 OR
  SHARE:AVAILABLEBALANCE>-$10.00) AND
 ((SHARE:TYPE>=0 AND SHARE:TYPE<=20) OR
  (SHARE:TYPE>=50 AND SHARE:TYPE<=60) OR
```

```
   SHARE:TYPE=70) AND
 NOT SHARE:DIVTYPE=10
END
```

PowerOn now evaluates the expression this way:

1. Checks the share type ranges from innermost to outermost parentheses, evaluating relational operators first, then evaluating AND operators, and finally evaluating the OR operators
2. Checks the share **Balance** and **Available Balance** fields, applying the -$10.00 first, then evaluating the relational operators, and then evaluating the OR operator
3. Applies the relational operator which compares the **Dividend Type** field to the literal that follows
4. Applies the NOT operator to the entire expression that follows it
5. Applies all remaining AND operators

If you carefully evaluate the expression following this order of precedence, you will find that it achieves the desired result.

Sample Specfile

The following specfile illustrates the use of Boolean expressions and parentheses to control the order of precedence. The resulting report list all accounts that meet the following criteria.

- Have either an open draft or certificate share
- Do not have any loans
- Have the share **Service Code** field set to 60 or 70
-
```
TARGET=ACCOUNT

SELECT
 ACCOUNT:CLOSEDATE='--/--/--' AND
 SHARE:CLOSEDATE='--/--/--' AND
(SHARE:SHARECODE=1 OR SHARE:SHARECODE=2) AND
 NOT ANY LOAN AND
 (ANYSERVICE(SHARE,60) OR ANYSERVICE(SHARE,70))
END

PRINT TITLE="Accounts with No Loan"
 HEADER="Account#   Member's Name     "
 HEADER="----------------------------"
 COL=001 ACCOUNT:NUMBER
 COL=012 NAME:SHORTNAME
 COL=030 NAME:HOMEPHONE
 NEWLINE
END
```

The resulting report lists the member’s account number, short name, and home phone number.

Activity: Boolean Rules of Precedence and Expressions

Specfile name: <initials>.BOOLEAN

You need to modify a loan portfolio specfile (called either ED.LOAN.PORT3 or <initials>.LOAN.PORT3). The CFO has decided that she only wants a loan to appear on the report if all the following criteria is true:

- The loan is open
- The current balance is greater than $0.00
- The original balance was greater than $500.00 or the credit limit is greater than $500.00
- The loan type is within a range entered by the user

In addition, add to your headers a line that indicates the loan type range selected by the user. The title already includes the due date to consider delinquent.

The prompting screen should look like this:

```
Lowest  Loan Type: 0
Highest Loan Type: 90
Date on which to consider Loan DQ: 03/01/2018
```

The resulting report should look like this:

```
Symitar Credit Union             Loan Portfolio (DQ as of 03/01/2018)

Loan Types Selected: 0000 to 0090

Account#   Member's Name Type Status     Due Date Last Payment Balance
----------------------------------------------------------------------
0000123440 CLARK,BETSY E 0002 Current    03/02/18 02/01/18     1,909.22
0000123444 SMITH,FRANK L 0000 Delinquent 02/05/18 01/11/18     9,900.00
0000123456 DOW,TOM       0000 Current    03/10/18 02/10/18     2,524.90
0000123456 DOW,TOM       0020 Current    03/15/18 02/15/18     1,000.00
```

Hints

- You need to add two variables in which to store the additional data you prompt for in the SETUP division.
- Do not delete the existing variable in the DEFINE division or the read literal from the SETUP division. You still need to prompt for a delinquency date.
- You can print the value of variables in your headers if you use HEADERS…END. You must FORMAT any non-character variables to print them out in headers if you use HEADER=.
- Be sure to set up appropriate test data to verify that your SELECT criteria is working properly. This is especially important when writing a specfile with complex selection criteria, because it is the only way to be certain that the expressions have been constructed correctly.

Debugging

Now that you have learned a fair amount of PowerOn syntax and have written a few fairly complex specfiles, it’s time to do some debugging.

Objective

At the end of this lesson, you will be able to debug a specfile which has common, simple errors in logic.

When and Why Do I Need to Debug?

As you already know, specfiles can pass the error checker but still not work as intended. This happens because there is a flaw in the logic or syntax which the error checker cannot detect.

Problems the error checker can't detect include:

- Incorrect use of parentheses to control precedence
- Improper use of relational operators (for example, <= instead of >= and vice versa)
- Failure to use DO…END clauses with the IF…THEN statement
- Targeting the wrong record

Debugging code with logic problems like these is an extremely useful skill and a great way to improve your own specfile writing skills.

Activity: The Logic is Right

Your instructor will divide you into teams. Each team will be assigned one of the specfiles on the following pages to debug. These specfiles will all pass the error checker, but do not operate as the writer intended. Your task is to correct all the flaws in the specfile's logic so that the specfile operates correctly, and then share your specfile (both before and after debugging) with the rest of the class.

Specfile #1

The following specfile is supposed to print the **Share ID**, **Share Type**, **Maturity Date**, and **Balance** field values of each open certificate share in the system. Instead, it lists only the first certificate in each account. In addition, it is supposed to print the last dividend paid and total dividend paid if the certificate is maturing in the next seven days and either of the following is true:

- The original balance was greater than $24,999.99.
- The total dividend paid exceeds $2,499.99.

This is not working properly. The output always shows the total dividend paid, regardless of the original balance or total dividend paid and maturity date. In addition, the last dividend paid amount is being printed for all shares with a total dividend exceeding $2,499.99, regardless of maturity date.

Correct the code shown below. The specfile is on Symitar and is called ED.DEBUG1. Copy it into a new specfile before making changes to it.

```
TARGET=ACCOUNT

SELECT
 SHARE:SHARECODE=2 AND
 SHARE:CLOSEDATE='--/--/--'
END

PRINT TITLE="Open Share Certificates"
 HEADERS
  COL=001 "Acct#"
  COL=015 "ID"
  COL=025 RIGHT "Type"
  COL=035 RIGHT "Maturity Date"
  COL=050 RIGHT "Balance"
  COL=065 RIGHT "Last Div Paid"
  COL=080 RIGHT "Tot Div Paid"
 END
 COL=001 ACCOUNT:NUMBER
 COL=015 SHARE:ID
 COL=025 SHARE:TYPE
 COL=035 SHARE:MATURITYDATE
 COL=050 SHARE:BALANCE
 IF SHARE:MATURITYDATE>=SYSTEMDATE+7 AND
    SHARE:ORIGINALBALANCE>$24,999.99 OR
    SHARE:DIVFROMOPEN>$2,499.99 THEN
  COL=065 SHARE:LASTDIVAMOUNT
  COL=080 SHARE:DIVFROMOPEN
 NEWLINE
END
```

Specfile #2

The following specfile is intended to find all auto loans (identified by **Loan Type**) that have been paid off before the end of the loan term during the past week. It's not correctly picking up the loans. In addition, the headers are all misaligned.

Correct the code (shown below). The specfile is on Symitar and is called ED.DEBUG2. Copy it into a new specfile before making changes to it.

```
TARGET=LOAN

SELECT
 LOAN:CLOSEDATE>SYSTEMDATE AND
 LOAN:CLOSEDATE<=SYSTEMDATE-7 AND
(LOAN:TYPE>=00 AND LOAN:TYPE<=02) OR
 LOAN:TYPE=04 OR
 LOAN:TYPE=10 OR
(LOAN:TYPE>=25 AND LOAN:TYPE<=27) OR
 LOAN:TYPE=57 AND
 LOAN:CHARGEOFFDATE='--/--/--' AND
 LOAN:CLOSEDATE<LOAN:MATURITYDATE
END

PRINT TITLE="Loans Paid Prior to End of Term"
 HEADERS
  COL=001 "Paid Date"
  COL=011 "Name"
  COL=030 "Account"
  COL=042 "ID"
  COL=055 "Mat Date"
  COL=001 REPEATCHR("-",54)
 END

 PRINT   LOAN:LASTPAYMENTDATE
 COL=011 NAME:SHORTNAME
 COL=030 ACCOUNT:NUMBER
 COL=042 LOAN:ID
 COL=055 LOAN:MATURITYDATE
 NEWLINE
END
```

Specfile #3

The following specfile is intended to find all issued cards (**Status** = **1**) with the specified card types that are expiring in the next 30 days and that must be manually reissued (**Reissue Code** = **0**). The report should list the account number, the primary name, the card number, the type of card (Visa®, ATM, or debit, depending on the **Card Type** field value), and the card's expiration date.

Currently, the specfile is not correctly identifying cards (it's picking up all **Type 25** and **Type 30** cards, regardless of expiration date, status, or reissue code). It's also not printing any card type at all for Visa cards.

Correct the code. The specfile is on Symitar and is called ED.DEBUG3. Copy it into a new specfile before making changes to it.

```
TARGET=CARD

SELECT
 CARD:STATUS=1 AND
 CARD:REISSUECODE=0 AND
 CARD:EXPIRATIONDATE<=SYSTEMDATE+45 AND
 CARD:EXPIRATIONDATE>=SYSTEMDATE AND
 CARD:TYPE>=10 AND CARD:TYPE<=12 OR
 CARD:TYPE=25 OR
 CARD:TYPE=30
END

PRINT TITLE="Cards with Manual Reissue Expiring this Month"
 HEADERS
  COL=001 "Account"
  COL=012 "Card Name"
  COL=030 "Card Number"
  COL=054 "Card Type"
  COL=080 RIGHT "Expires on"
  NEWLINE
  COL=001 REPEATCHR("-",80)
  NEWLINE
 END

 COL=001 ACCOUNT:NUMBER
 COL=012 NAME:SHORTNAME
 COL=030 CARD:NUMBER
 IF CARD:TYPE<=10 AND CARD:TYPE>=12 THEN
  COL=054 "Visa Card"
 ELSE IF CARD:TYPE=25 THEN
  COL=054 "ATM Card"
 ELSE IF CARD:TYPE=30 THEN
  COL=054 "Debit Card"
 COL=080 CARD:EXPIRATIONDATE
 NEWLINE
END
```

Demand Specfiles and Mathematical Expressions

As you have already learned, there are two different modes for running specfiles: batch mode and demand mode. In this lesson, you will learn how to write specfiles for demand use and how to write some simple mathematical expressions.

Objectives

At the end of this lesson, you will be able to do the following:

- Explain how demand specfiles differ from batch specfiles
- Install a specfile for demand use
- Run a demand specfile
- Write mathematical expressions and determine the data type of the result
- Write a demand specfile

Demand vs. Batch

A demand specfile differs from a batch specfile in the following ways.

- A demand specfile can target any record in any file except the ATM dialog file or the CDM queue file. The ATM dialog area does not have a **Specfile** field where you can type in or select a specfile, and there is no dedicated area for the CDM queue file. These files are not keyed or sequential and are inefficient for demand use.
- A demand specfile can use data from only one record path at a time. (Batch specfiles can use data from all selected records and all records in that record path in the database.)
- Specfiles must be installed for demand use.
- Demand specfiles are useful for designing custom inquiries and for generating certain types of forms and letters that use data from a single member’s account.
- You can also use demand specfiles to perform on-demand file maintenance (also called “FM scripting”) of Account file records. In addition, validation specfiles, the collection queue and collection screen specfiles, and specfiles that generate notices during batch processing are demand specfiles.

Installing a Specfile for Demand Use

After you write your demand specfile and ensure that it is error-free, follow these steps to make it available for demand use.

Steps

1. With the specfile open and in the active window, click **Install a Specfile for Demand Use** in the PowerOn Control work area.
2. Symitar tells you how many bytes are required to store the specfile in demand mode and displays the Install it? prompt. Answer Yes to continue.
3. Close the specfile.

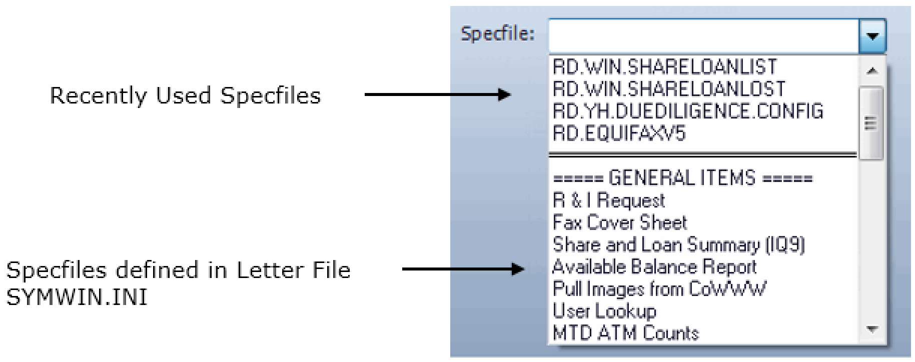
What to do next

Whenever you make changes to a specfile you have installed for demand use, you must follow this procedure again to reinstall the new version of the specfile. Otherwise, the demand version will not reflect your changes.

Run a Demand Specfile

The method you choose to run a demand specfile depends on which data file the specfile targets and what the specfile does.

- A validation specfile runs automatically when you create, revise, or delete the targeted record.
- You can run a demand specfile that targets the Account file in the following ways:
    - Go to the Account Manager work area, select a member account, and enter the name of the specfile in the **Specfile** field.
    - Go to the Teller Transactions work area, select a member account, and enter the name of the specfile in the **Specfile** field.
    - You can also run demand specfiles from the Application Processing work area and from the collection queue. In addition, you can set up demand specfiles to run automatically at certain times (for example, when someone enters the Teller Transactions work area), but that is a more advanced PowerOn function.



- You can run a demand specfile that targets the GL Account file from the GL Account Manager work area. To do this, you must select a GL account, then enter (or select) the specfile name in the **Specfile** field.
- You can run a demand specfile that targets the Inventory file from the Inventory Manager work area. Select an inventory item, then enter (or select) the specfile name in the **Specfile** field.
- You can run a demand specfile that targets the Accounts Payable file from the Accounts Payable Manager work area. Select a Vendor record, then enter (or select) the specfile name in the **Specfile** field.

Sample Specfile

The demand specfile you will be writing in this lesson will perform mathematical calculations. The sample specfile below shows you some examples. It calculates the total number of days elapsed in the current year and then calculates the percentage of the year represented by that number of days.

The first calculation results in a number because a date minus a date equals a number. The second calculation results in a rate because a number divided by a number equals a rate. We will also discuss how you can determine the data type of the result of your own mathematical calculations.

```
TARGET=ACCOUNT

DEFINE
 DAYS=NUMBER
 PERCENTOFYEAR=RATE
END

PRINT TITLE="Some Calculations"
 NEWLINE
 PRINT "Today's Date: "
 PRINT FORMAT("99/99/9999",SYSTEMDATE)
 NEWLINE
 NEWLINE
 PRINT "Number of days so far this year: "
 DAYS=SYSTEMDATE-'01/01/18'
 PRINT DAYS
 NEWLINE
 NEWLINE
 PRINT "Percent of the year passed to date: "
 PERCENTOFYEAR=DAYS/365.25
 PRINT PERCENTOFYEAR
 NEWLINE
END
```

The on-screen output looks like this:

```
Today’s Date: 06/30/2018
Number of days so far this year: 180
Percent of the year passed to date: 49.281%
```

Write Mathematical Expressions

You can perform mathematical calculations on any operand (field, variable, literal, or special literal) with a data type other than character data.

To write an expression that performs a mathematical calculation, use the following arithmetic operators:

- + to add two data items together:

  ```
  SHARE:BALANCE+AGGBAL
  ```
- – to subtract one data item from another:

  ```
  LOAN:DUEDATE-LOAN:LASTPAYMENTDATE
  ```
- * to multiply one data item by another:

  ```
  LOAN:INTERESTRATE*1.10
  ```
- / to divide one data item by another:

  ```
  SHARE:BALANCE/2
  ```

You can perform more complex calculations by enclosing portions of the expression in parentheses. For example, suppose you want to calculate how much principal the member would pay each month on the remaining balance of a loan if there were no interest charge. If you write the following expression:

```
LOAN:BALANCE/LOAN:PAYMENTCOUNT-LOAN:PAYMENTHISTORY:1
```

PowerOn divides the value in the **Balance** field by the value in the **Payment Count** field and then subtracts the value in the **Payment History 1** field (this field stores the number of payments received). For example, if the **Balance** field is set to $1,200, the **Payment Count** field is set to 12 months, and **Payment History 1** field is set to 2, the system divides $1,200 by 12, subtracts 2 and arrives at $98. This is not the result you want. Instead, you must write the following code to divide the balance by the result of the payment count minus the value in the **Payment History 1** field ($1,200/10 = $120):

```
LOAN:BALANCE/(LOAN:PAYMENTCOUNT-LOAN:PAYMENTHISTORY:1)
```

Results of Arithmetic Operations

When you perform an arithmetic operation, you must know what data type will be assigned to the result of the calculation. Without this information, you cannot know which data type to assign to the variable that stores the result of a calculation or how the result will appear on the screen or in a report.
The following tables illustrate which data type combinations are not acceptable (N/A) for each operation and the data type that results from each acceptable combination.

Addition

Operand	**MONEY**	**NUMBE**	**FLOAT**	**DATE**	**RATE**	**CODE**
MONEY	MONEY	MONEY	FLOAT	N/A	N/A	MONEY
NUMBER	MONEY	NUMBER	FLOAT	DATE	RATE	NUMBER
FLOAT	FLOAT	FLOAT	FLOAT	DATE	FLOAT	FLOAT
DATE	N/A	DATE	DATE	N/A	N/A	DATE
RATE	N/A	RATE	FLOAT	N/A	RATE	RATE
CODE	MONEY	NUMBER	FLOAT	DATE	RATE	NUMBER

Subtraction

Operand	**MONEY**	**NUMBE**	**FLOAT**	**DATE**	**RATE**	**CODE**
MONEY	MONEY	MONEY	FLOAT	N/A	N/A	MONEY
NUMBER	MONEY	NUMBER	FLOAT	N/A	RATE	NUMBER
FLOAT	FLOAT	FLOAT	FLOAT	N/A	FLOAT	FLOAT
DATE	N/A	DATE	DATE	NUMBER	N/A	DATE
RATE	N/A	RATE	FLOAT	N/A	RATE	RATE
CODE	MONEY	NUMBER	FLOAT	N/A	RATE	NUMBER

Multiplication

Operand	**MONEY**	**NUMBE**	**FLOAT**	**DATE**	**RATE**	**CODE**
MONEY	NUMBER	MONEY	FLOAT	N/A	MONEY	MONEY
NUMBER	MONEY	NUMBER	FLOAT	N/A	RATE	NUMBER
FLOAT	FLOAT	FLOAT	FLOAT	N/A	FLOAT	FLOAT

Operand	MONEY	NUMBE	FLOAT	DATE	RATE	CODE
DATE	N/A	N/A	N/A	N/A	N/A	N/A
RATE	MONEY	RATE	FLOAT	N/A	RATE	NUMBER
CODE	MONEY	NUMBER	FLOAT	N/A	NUMBER	NUMBER

Division

Operand	MONEY	NUMBE	FLOAT	DATE	RATE	CODE
MONEY	RATE	MONEY	FLOAT	N/A	MONEY	MONEY
NUMBER	RATE	RATE	FLOAT	N/A	RATE	NUMBER
FLOAT	FLOAT	FLOAT	FLOAT	N/A	FLOAT	FLOAT
DATE	N/A	N/A	N/A	N/A	N/A	N/A
RATE	N/A	RATE	FLOAT	N/A	RATE	NUMBER
CODE	MONEY	RATE	FLOAT	N/A	NUMBER	RATE

Activity: Demand Specfiles and Mathematical Expressions

Specfile name: <initials>.HOWOLD.DAYS

Your Member Services department would like a demand specfile that figures out the age, in days, of the primary member for an account. The output looks like this:

```
Today's Date: 12/10/2017

Member's Name: JOHN DOE JR

Member's Birthdate: 03/03/1964

Member's Age in Days: 19,640
```

Hints

- Remember that the PRINT statement places data in the next available column. This means that you must type additional spaces within your print statements to achieve proper spacing.
- To double-space the output as shown above, you need two NEWLINE statements between each line of text.
- Be sure to test the results in the Teller Transactions or Account Manager work area.

Arithmetic Functions

Many specfiles perform mathematical operations, but in some cases the data type resulting from a calculation isn't the data type you want. For example, suppose you want to determine how many months there are between two dates. You subtract one date from the other, and then divide by 30. The default data type for the result is a rate, so if 90 days have passed, the number of months is displayed not as 1.5, but as 150.000%. What now? The answer is an arithmetic function.

Objectives

At the end of this lesson, you will be able to use the following arithmetic functions in a specfile:

- MONEY
- NUMBER
- RATE
- FLOAT
- VALUE
- ABS

Sample Specfile

The specfile below shows you how to use arithmetic functions to change the data type of the result of a calculation. This is the same specfile we looked at earlier, but it now includes a calculation that determines the number of whole months that have elapsed.

```
TARGET=ACCOUNT
DEFINE
 DAYS=NUMBER
 PERCENTOFYEAR=RATE
 MONTHS=NUMBER
END

PRINT TITLE="Some Calculations"
 NEWLINE
 PRINT "Today’s Date: "
 PRINT FORMAT("99/99/9999",SYSTEMDATE)
 NEWLINE NEWLINE
 PRINT "Number of days so far this year: "
 DAYS=SYSTEMDATE-'01/01/18'
 PRINT DAYS
 NEWLINE NEWLINE
 PRINT "Percent of a year passed to date: "
 PERCENTOFYEAR=DAYS/365.25
 PRINT PERCENTOFYEAR
 NEWLINE NEWLINE
 PRINT "Number of whole months elapsed: "
 MONTHS=NUMBER(DAYS/30)
 PRINT MONTHS
 NEWLINE
END
```

The screen output looks like this:

```
Today’s Date: 06/30/2018

Number of days so far this year: 180

Percent of a year passed to date: 49.281%

Number of whole months elapsed: 6
```

Arithmetic Functions Descriptions

The specfile on the previous page shows an example of one arithmetic function: the NUMBER function. Most (but not all) of PowerOn's arithmetic functions are described in more detail below.

- The MONEY function converts numeric or rate data to monetary data. For example:

  ```
  MONEY(LOAN:BALANCE/30)
  ```
- The NUMBER function converts money or rate data to number data. For example:

  ```
  NUMBER((LOAN:OPENDATE-SYSTEMDATE)/365.25)
  ```
- The RATE function converts money or number data to rate data. For example:

  ```
  RATE(LOAN:INTERESTRATE*1.0325)
  ```
- The FLOAT function converts a money, number, rate, date, or code data item to a float data item. For example:

  ```
  FLOAT(LOAN:INTERESTRATE)
  ```
- The VALUE function extracts the numeric value of the digits in a character data item. This function ignores any characters other than digits and translates the digits into a number data item. For example:

  ```
  VALUE(SHARE:REFERENCE)
  ```

  This is useful when you want to perform a mathematical calculation on a field which includes numeric data, but has a data type of character. For example, if you use the Share record Reference field to store the member's Visa® credit limit, you might want to extract that value from the Reference field. Note that you can use the MONEY, RATE, or FLOAT function with the VALUE function to change the extracted numeric value to a data type other than number:

  ```
  MONEY(VALUE(SHARE:REFERENCE))
  ```
- The ABS function finds the absolute value of an arithmetic expression. In other words, it always returns a positive number, even if the calculation results in a negative number:

  ```
  ABS(STARTDATE-ENDDATE)
  ```

  This is especially useful when you are subtracting one date from another if the user enters both dates. You have no way of knowing whether the user will correctly enter the earlier date (the one further in the past) before entering the later date. If the user enters the later date first, the calculation would result in a negative number rather than a positive number.

  The ABS function returns the absolute value with the default data type of the original expression. For example, if the result of the calculation would be returned as money data, the ABS function returns a positive money amount.

Activity: Arithmetic Functions

Specfile name: <initials>.HOWOLD.YEARS

Your Member Services department would like you to modify a demand specfile that calculates the age of the member in days. Now they want to know the age of the member in full years, and they don't care about months and days.

The existing specfile is called ED.HOWOLD.DAYS, or if you wrote your own for the Demand Specfiles lesson, <initials>.HOWOLD. DAYS. Please read the existing specfile of your choice into a new specfile.

The output should look like this:

```
Today’s Date: 02/30/2009

Member’s Name: JOHN Q MEMBER

Member’s Birthdate: 02/16/1964

This member is 45 years old.
```

Hints

- You already know the age of the member in days. Try dividing that number by the number of days in a year.
- Remember that your arithmetic calculation may not result in a value that has the correct data type to be printed as a whole number (which is what your Member Services department wants).

TIP

This activity produces an estimate age to illustrate the use of the Number function. To calculate an accurate age, please use the following method.

```
AGE=YEAR(SYSACTUALDATE)-YEAR(NAME:BIRTHDATE)
   [If they have not had their birthday yet this year- subtract 1]
THISYEARSBIRTHDATE = DATE(MONTH(NAME:BIRTHDATE),DAY(NAME:BIRTHDATE),
YEAR(SYSACTUALDATE))
IF THISYEARSBIRTHDATE> SYSACTUALDATE THEN
    AGE=AGE-1
```

Enter Functions

Just as you can prompt a user for input in batch mode using read literals, you can prompt a user for input in demand mode using enter functions. In this chapter, you'll learn the enter functions and how to use them.

Objectives

At the end of this lesson, you will be able to:

- Use enter functions to prompt a user for input in demand mode
- Specify a default response for an enter function using any of the following:
    - Literal
    - Special literal
    - Database field
    - Variable

Sample Specfile

This specfile calculates the number of days elapsed between the system date and a date entered by the user, and then calculates the percentage of the year and the number of whole months elapsed between the two dates.

```
TARGET=ACCOUNT

DEFINE
 STARTDATE=DATE
 DAYS=NUMBER
 PERCENTOFYEAR=RATE
 MONTHS=NUMBER
END

SETUP
 STARTDATE=ENTERDATE("Enter start date",'--/--/--')
END

PRINT TITLE="Some Calculations"
 NEWLINE
 PRINT "Start Date: "
 PRINT FORMAT("99/99/9999",STARTDATE)
 NEWLINE
 PRINT "Today’s Date: "
 PRINT FORMAT("99/99/9999",SYSTEMDATE)
 NEWLINE
 PRINT "Number of days between the two dates: "
 DAYS=ABS(SYSTEMDATE-STARTDATE)
 PRINT DAYS
 NEWLINE
 PRINT "Percent of a year between the dates: "
 PERCENTOFYEAR=DAYS/365.25
 PRINT PERCENTOFYEAR
 NEWLINE
 PRINT "Number of whole months between the dates: "
 MONTHS=NUMBER(DAYS/30)
 PRINT MONTHS
 NEWLINE
END
```

The screen output looks like this:

```
Start Date: 05/15/2018
Today’s Date: 06/30/2018
Number of days between the two dates: 45
Percent of a year between the dates: 12.320%
Number of whole months between the dates: 1
```

Enter Function Syntax

The enter functions are very similar to the read literals you use to prompt for input in batch mode. The primary difference between enter functions and read literals is that all the enter functions require you to specify a default answer.

- Use ENTERCHARACTER to prompt for character data. This is a character function. The syntax is:

  ```
  ENTERCHARACTER("prompt",max length,default)
  ```

  For example:

  ```
  FIRSTNM=ENTERCHARACTER("First Name",40,"Joe")
  ```
- Use ENTERCODE to prompt for code data. This is an arithmetic function. The syntax is:

  ```
  ENTERCODE("prompt",max value,default)
  ```

  For example:

  ```
  SELECTION=ENTERCODE("Selection",3,0)
  ```
- Use ENTERDATE to prompt for date data. This is an arithmetic function. The syntax is:

  ```
  ENTERDATE("prompt",default)
  ```

  For example:

  ```
  STARTDATE=ENTERDATE("Start Date",SYSTEMDATE)
  ```
- Use ENTERMONEY to prompt for money data. This is an arithmetic function. The syntax is:

  ```
  ENTERMONEY("prompt",default)
  ```

  For example:

  ```
  LOWBAL=ENTERMONEY("Lowest Balance",$0.00)
  ```
- Use ENTERNUMBER to prompt for number data. This is an arithmetic function. The syntax is:

  ```
  ENTERNUMBER("prompt",default)
  ```

  For example:

  ```
  SELECTION=ENTERNUMBER ("Selection",0)
  ```
- Use ENTERRATE to prompt for rate data. This is an arithmetic function. The syntax is:

  ```
  ENTERRATE("prompt",default)
  ```

  For example:

  ```
  INTRATE=ENTERRATE("Interest Rate",10.000%)
  ```

- Use ENTERYESNO to prompt for a **Y** or **N** answer. This is a character function which limits the user's input to Y, N, 1, or 0. If the user enters **1** or **0**, Symitar changes the stored entry to **Y** or **N,** respectively. The syntax is:

  ```
  ENTERYESNO("prompt",default)
  ```

  For example:

  ```
  YN=ENTERYESNO("Okay?","Y")
  ```

For each function, the default can be a literal or special literal of the proper data type, or the default can be pulled from a field with the proper data type. For example:

```
ENTERMONEY("Enter starting balance",$0.00)
ENTERDATE("Enter starting date",SYSTEMDATE)
ENTERCHARACTER("Enter name",40,NAME:LONGNAME)
ENTERYESNO("Continue?","Y")
```

Activity: Enter Functions

Specfile name: <initials>.HOWOLD.YEARS2

Your Member Services department would again like you to modify the specfile which calculates the member's age. Now they want the user to be able to choose between calculating the primary member's age in years or the age of another person based on a date the user enters.

The existing specfile is called ED.HOWOLD.YEARS, or if you wrote your own for the Arithmetic Functions lesson, <initials>.HOWOLD.YEARS. Please copy the existing specfile of your choice into the new specfile.

The prompting screen should look like this (note that the **Date for Calculation** prompt should be displayed only if the user enters **N** at the **Calculate primary member's age?** prompt):

```
Calculate primary member’s age? [Y]: N

Date for calculation [--/--/--]: 06/06/64
```

The output should look like this if the user calculates the primary member's age:

```
Today’s Date: 06/30/2018

Member’s Name: JOHN Q MEMBER

Member’s Birthdate: 02/16/1958

This member is 60 years old.
```

The output should look like this if the user calculates another person's age:

```
Today’s Date: 06/30/2018

Birthdate Entered: 06/06/1964

This person is 54 years old.
```

Hints

- Remember that ENTERYESNO limits the user's input to Y, 1, N, or 0. This means that you don't have to check to be sure the user made a valid entry.
- The prompt that asks for the birth date to use for the calculation should appear only if the user chooses not to calculate the primary member's age. To make this happen, you need an IF…THEN statement in the SETUP division.
- You need another IF…THEN statement in the PRINT division to print different data, depending on whether the user calculates the primary member's age or another person's age.

Date Functions

There are several different functions you can use to help you calculate dates in the past or future. These functions let you find members who are over or under a certain age, find all certificates that mature on a certain date, and so on.

Objectives

At the end of this lesson, you will be able to use the following date functions in a specfile:

- DATEOFFSET
- DAY
- MONTH
- FULLYEAR

Sample Specfile

The following specfile shows you how you can use date functions to calculate a date in the future.

```
TARGET=ACCOUNT

DEFINE
 YN=CHARACTER(1)
 CERTTERM=NUMBER
 MATDATE=DATE
END

SETUP
 YN=ENTERYESNO("Is Term in Months?","Y")
 IF YN="Y" THEN
  CERTTERM=ENTERNUMBER("Enter term in months",12)
 ELSE
  CERTTERM=ENTERNUMBER("Enter term in days",90)
END

PRINT TITLE="Calculate Maturity Date"
 PRINT "Today’s date: "
 PRINT FORMAT("99/99/9999",SYSTEMDATE)
 NEWLINE NEWLINE
 IF YN="Y" THEN
  PRINT "Certificate Term in Months: "
 ELSE
  PRINT "Certificate Term in Days: "
 PRINT CERTTERM
 NEWLINE NEWLINE
 PRINT "This certificate will mature on "
  IF YN="Y" THEN
  MATDATE=DATEOFFSET(SYSTEMDATE,CERTTERM,0)
 ELSE
  MATDATE=SYSTEMDATE+CERTTERM
 PRINT FORMAT("99/99/9999",MATDATE)+"."
 NEWLINE
END
```

The input screen looks like this:

```
Is Term in Months [Y]:
Enter term in months [12]:
```

The output screen looks like this:

```
Today's date: 01/18/2018

Certificate Term in Months: 12

This certificate will mature on 01/18/2019.
```

DATEOFFSET Function

The DATEOFFSET function takes a starting date and adjusts it by a specified number of months and to a specified day of the month. This is useful for calculating dates like “the date six months from today” or “the last day of the previous month”.

The syntax is:

```
DATEOFFSET(Startdate,Monthcount,Day)
```

- **Startdate** is a date literal, variable, or expression. It is the starting date for the calculation.
- **Monthcount** is a numeric literal, variable, or expression. It is the number of months to be added or subtracted from the Startdate. It can be positive, negative, or zero, and it can have a value from -2200 to 2200.
- **Day** is a numeric literal, variable, or expression. It is a specific day of the month, or 0 if you want the day of the month to match the day value of Startdate. It can have a value from 0 to 31. The value 31 means the last day of the month.

DATEOFFSET returns a date value. Here are some examples:

- DATEOFFSET(SYSTEMDATE,-1,31) = last day of last month
- DATEOFFSET(SYSTEMDATE,0,31) = last day of this month
- DATEOFFSET(SYSTEMDATE,1,1) = first day of next month
- DATEOFFSET(SYSTEMDATE,12,0) = day exactly one year from today
- DATEOFFSET(SYSTEMDATE,-18*12,0) = day exactly 18 years ago

You can format the result of a DATEOFFSET function. For example:

```
FORMAT("99/99/9999",DATEOFFSET(SYSTEMDATE,-55*12,0))
```

DAY Function

The DAY function finds the value of the day portion of a date field or date variable.

The syntax is:

```
DAY(date expression)
```

For example, the following DAY function selects all loans due on the 15th of any month:

```
SELECT
 DAY(LOAN:DUEDATE)=15
END
```

MONTH Function

The MONTH function finds the value of the month portion of a date field or date variable.

The syntax is:

```
MONTH(date expression)
```

The following MONTH function will print the number of the month:

```
PRINT MONTH(SYSTEMDATE)
```

FULLYEAR Function

The FULLYEAR function finds the four-digit value of the year portion of a date field or date variable.

The syntax is:

```
FULLYEAR(date expression)
```

The following FULLYEAR function subtracts three years from the current system date and prints the result:

```
PRINT FORMAT("9999",FULLYEAR(SYSTEMDATE)-3)
```

Be sure to use the FORMAT function when printing out the value of FULLYEAR. If you don’t format the result, you’ll end up with a comma between the first and second digits of the year. In other words, the year 2000 will be printed as 2,000.

Activity: Date Functions

Specfile name: <initials>.DATES

Your Member Services department has another request. They would like a demand specfile that calculates the date on which a person will reach an age (in whole years) entered by the user. They would also like to be able to enter the birth date to use for the calculation. The default for this birth date should be the primary member's birth date.

The input screen that prompts the user to enter the age and birth date to be used for the calculation should look like this:

```
Enter age to calculate [0] :59
Enter birthdate [06/06/64] :021668
```

The output screen should look like this:

```
Birthdate: 02/16/1968

This person turns 59 on 02/16/2027.
```

Hints

- You'll need two variables: one to store the birth date, and one to store the age.
- Use a date function to calculate the date on which the member reaches the specified age.
- Remember that you can multiply the number of years by 12 to determine the number of months that will elapse during those years.

FOR EACH Loops

As you have learned in earlier lessons, if you target the Account record but want to print out information about shares, your specfile will only find and print information about the first share it finds that meets your criteria. Until now, the only way you have been able to get around this was to target the Share record instead.

The FOR EACH statement lets you target a parent record, and then sequentially loop through and print or gather information about each child record.

Objectives

At the end of this lesson, you will be able to:

- Define the term "loop"
- Use the FOR EACH statement in a specfile
- Explain when to use WITH criteria in a FOR EACH loop

Loop Definition

A loop in programming terms is a statement that is executed multiple times until a condition is met, at which time the loop ends.

For example, suppose you need to find all loans with a **Type** of **20** (these are your credit card loans) and then print the **Expiration Date** field value for each card issued for that account. You need to target the Account record because your report must include data from both the Loan and Card records. Without the looping statement, PowerOn looks at each Loan record only until it finds one that meets the criteria specified. As soon as a matching loan is found, PowerOn goes on to the next Account record in the system. To get PowerOn to look at each Loan record, you need a looping statement.

There are several statements in the PowerOn language that perform loops. In this lesson, you'll learn one of the simplest and most commonly used looping statements: FOR EACH.

Sample Specfile

The following specfile shows the use of the FOR EACH statement to determine the aggregate share and loan balance for each account.

```
TARGET=ACCOUNT

DEFINE
 AGGSHAREBAL=MONEY
 AGGLOANBAL=MONEY
END

SELECT
 ACCOUNT:CLOSEDATE='--/--/--' AND
(SHARE:CLOSEDATE='--/--/--' OR
 LOAN:CLOSEDATE='--/--/--')
END

PRINT TITLE="Aggregate Share & Loan Balances"
 HEADERS
  COL=001 "Account #"
  COL=013 "Member's Name"
  COL=045 RIGHT "Agg Share Bal"
  COL=065 RIGHT "Agg Loan Bal"
  NEWLINE
  COL=001 REPEATCHR("-",65)
 END
 COL=001 ACCOUNT:NUMBER
 COL=013 NAME:SHORTNAME
 AGGSHAREBAL=$0.00
 AGGLOANBAL=$0.00
 FOR EACH SHARE WITH (SHARE:CLOSEDATE='--/--/--')
  DO
   AGGSHAREBAL=AGGSHAREBAL+SHARE:BALANCE
  END
 FOR EACH LOAN WITH (LOAN:CLOSEDATE='--/--/--')
  DO
   AGGLOANBAL=AGGLOANBAL+LOAN:BALANCE
  END
 IF AGGSHAREBAL<>$0.00 THEN
  COL=045 AGGSHAREBAL
 ELSE
  COL=045 RIGHT "No Agg. Balance"
 IF AGGLOANBAL<>$0.00 THEN
  COL=065 AGGLOANBAL
 ELSE
  COL=065 RIGHT "No Agg. Balance"
 NEWLINE
END
```

The resulting report looks like this:

```
Symitar CU                          Aggregate Share and Loan Balances

Account #   Member's Name          Agg Share Bal          Agg Loan Bal
----------------------------------------------------------------------
0000123456  JAMES,TOM                  43,746.11              5,042.10
0000123457  DIXON,LINDA M        No Agg. Balance              6,412.81
0000123461  JENSEN,LAUREN C             4,865.50       No Agg. Balance
0000123462  ROGERS,DONNA M              4,828.33              4,197.60
```

FOR EACH Definition

The FOR EACH statement instructs PowerOn to examine each record of the specified record type and execute one or more additional statements for each record found. The record specified must be a subrecord of the record currently being processed.

You can use a FOR EACH loop only in the PRINT division, in the SETUP division (for demand specfiles only), or in a PROCEDURE called by the PRINT division. (We don't cover procedures in this course. You'll have to join us for the PowerOn 102 course if you want to learn how to use procedures.)

You should use a FOR EACH loop whenever you need to target one record, but you need PowerOn to examine information in subsidiary records of a specific type. For example, if you need to target the Account record, but the report should list all shares in the account, you can use a FOR EACH SHARE loop to instruct PowerOn to print each share. It is important to note that you will only have access to the subsidiary record fields while you are inside the loop. If there is any information from that record you will need later in your program, you must save it into a variable for later use.

You can nest FOR EACH loops within one another. You must do this if the targeted record is the grandparent (or great-grandparent) of the record you need to examine. For example, if you target the Account record and want to list all shares followed by all child Share Name records, you would place a FOR EACH SHARE NAME loop within the FOR EACH SHARE loop. The FOR EACH SHARE NAME loop is nested within the FOR EACH SHARE loop. If you do this, PowerOn executes the nested loop only if the conditions set in both loops are true.

FOR EACH Syntax

The following is the FOR EACH syntax.

```
FOR EACH record type
 DO
  statements
 END

FOR EACH record type
 DO
  statements
 END
 UNTIL Boolean expression

FOR EACH record type WITH Boolean expression
 DO
  statements
 END

FOR EACH record type WITH Boolean expression
 DO
  statements
 END
 UNTIL Boolean expression
```

- The record type is the PowerOn name for the record you want to access.
- A statement is an order for an action to be taken by PowerOn, such as PRINT or NEWLINE.
- A Boolean expression is an expression that evaluates to either true or false. In a FOR EACH loop, PowerOn continues to execute the loop as long as the Boolean expression is true. As soon as the Boolean expression is false, the loop ends.

FOR EACH Syntax Examples

This is an example of a simple FOR EACH loop without any Boolean expressions to evaluate.

```
FOR EACH SHARE
 DO
  COL=001 SHARE:ID
  COL=008 SHARE:DESCRIPTION
  NEWLINE
 END
```

This is an example of a nested FOR EACH loop:

```
FOR EACH LOAN
 DO
  COL=001 LOAN:ID
  FOR EACH LOAN NAME
   DO
    COL=008 LOAN NAME:SHORTNAME
    NEWLINE
   END
  NEWLINE
 END
```

This FOR EACH loop uses a Boolean expression:

```
FOR EACH CARD WITH (CARD:TYPE=20)
 DO
  COL=010 CARD:EXPIRATIONDATE
  NEWLINE
 END
```

WITH Criteria

When you use a FOR EACH loop, you must be especially conscious of how the TARGET and SELECT divisions in your specfile are working. If your specfile uses any SELECT criteria that point to information contained in the record you are examining with your FOR EACH loop, you almost certainly need to repeat those SELECT criteria in a WITH clause appended to your FOR EACH loop. Without the WITH clause, your specfile may not work as you intend.

For example, suppose you're writing a specfile to calculate the aggregate balance of all draft shares in each member's account. You target the Account record and use a FOR EACH loop to examine the Share records. Since you don't want to waste time by examining accounts that don't have any draft shares, you would probably write the following SELECT division:

```
SELECT
 ACCOUNT:CLOSEDATE='--/--/--' AND
 SHARE:SHARECODE=1 AND
 SHARE:CLOSEDATE='--/--/--'
END
```

Once you've done this, it's tempting to write the following FOR EACH loop in your PRINT division:

```
FOR EACH SHARE
 DO
  SHAREBAL=SHAREBAL+SHARE:BALANCE
 END
```

The problem is that your specfile won't add up the balances of draft shares; instead, it adds up the balances of all shares in the account. It does this because all you accomplish with your SELECT criteria is to ensure that the account you're processing has at least one draft share. You've done nothing to tell PowerOn to examine only those records that are draft shares.

For this specfile to work as you intend, you must repeat your SELECT criteria in a WITH clause as follows:

```
FOR EACH SHARE WITH (SHARE:SHARECODE=1 AND
                     SHARE:CLOSEDATE='--/--/--')
 DO
  SHAREBAL=SHAREBAL+SHARE:BALANCE
 END
```

Another way to think of a FOR EACH statement is to imagine it as a mini-specfile within your specfile. The words FOR EACH followed by the record type are essentially equivalent to TARGET=. The word WITH and the Boolean expression that follows it correspond to a SELECT division. Finally, the statements within the DO...END clause are similar to the statements you execute within your PRINT division because they are executed once for each record that meets your criteria.

Activity: FOR EACH Loops

Specfile name: <initials>.FOREACH

Your Marketing department wants a report that lists all open shares in each open account, followed by an aggregate share balance for that account. They would like the report sorted in alphabetical order by member's last name.

The resulting report should look like this:

```
Member's Name     Account#    Type    ID            Balance
-----------------------------------------------------------
BACALL,LAUREN C   0000123461  0000    0000           575.61
                              0020    0020         2,786.92
                              0050    0050           502.97
                                                 ----------
Aggregate Share Balance:                           3,865.50

BUTLER,RHETT      0077123402  0000    0000         6,142.31
                              0050    0051         1,517.00
                                                 ----------
Aggregate Share Balance:
```

Hints

- You must target the Account record to print an aggregate share balance for each account.
- You need a FOR EACH …WITH loop to repeat your select criteria to ensure that you print balances only for shares that meet those criteria.
- Define a variable to add up the balances of all shares in the account. Be sure to zero the value of this variable before entering your FOR EACH loop, and remember to print the final result in the PRINT division after you complete your loop.

TOTAL Division

You have learned how to use the TOTAL print statement option and the SUBTOTAL sort key option to create standard totals and subtotals for monetary and number fields. But there are times when these options don't provide enough flexibility, either because the layout doesn't suit your needs or because you need to perform additional calculations on the totals (for example, averages or percentages).

Objectives

At the end of this lesson, you will be able to:

- Explain how a TOTAL division works
- Write a specfile which uses a TOTAL division

Sample Specfile

The sample specfile shown below uses a TOTAL division to print the total balance of all loans, the balance of the Loan Loss GL as entered by the user, and the ratio of the two.

```
TARGET=LOAN

DEFINE
 LOSSBAL=MONEY
 LOANBAL=MONEY
END

SETUP
 LOSSBAL=MONEYREAD("Enter Loan Loss GL Balance")
END

SELECT
 LOAN:CHARGEOFFDATE<>'--/--/--'
END

PRINT TITLE="Allowance for Loan Loss Report"
 HEADER=""
 LOANBAL=LOANBAL+LOAN:BALANCE
END
TOTAL
 PRINT   "Total Charged Off Loan Balance:"
 COL=050 LOANBAL
 NEWLINE
 PRINT   "Current Loan Loss GL Balance:"
 COL=050 LOSSBAL
 NEWLINE
 PRINT   "Percent of Charged Off Balance"
 NEWLINE
 PRINT   "Covered by Loan Loss GL Balance:"
 COL=050 LOSSBAL/LOANBAL
 NEWLINE
END
```

The resulting report looks like this:

```
Symitar CU               Allowance for Loan Loss Report

Total Charged Off Loan Balance:          210,237.23
Current Loan Loss GL Balance:            278,934.33
Percent of Charged off Balance
Covered by Loan Loss GL Balance:           132.675%
```

How a TOTAL Division Works

PowerOn processes the TOTAL division after it looks at all selected records during the PRINT division.

Once PowerOn completes the PRINT division, it no longer has access to any records in the database. The only data you can retrieve in the TOTAL division is information stored in variables. You can gather this data through the SETUP division or accumulate it during the PRINT division. You can also use literals or special literals in a TOTAL division to perform calculations.

In some cases, you may write a specfile that does nothing in the PRINT division other than print out headers or accumulate values in variables, and then use the TOTAL division to produce the report.

Within a TOTAL division, you can use:

- All the print statements you've learned so far except SUPPRESS and TOTAL
- All relational and mathematical expressions
- The REPEATCHR, CAPITALIZE, and FORMAT functions
- The IF...THEN...ELSE statement

One side effect of using a TOTAL division is that it cancels out any TRAILERS subdivision you might have. Adding a TOTAL division suppresses printing of anything you've put in a TRAILERS subdivision. This means that if you want to print a breaking line at the end of your report after the detail and before the totals, you must print it as the first thing in your TOTAL division, as shown below:

```
TOTAL
 PRINT REPEATCHR("-",80)
 NEWLINE
 <etc.>
END
```

Activity: Using the TOTAL Division

Specfile name: <initials>.TOTAL.DIV

Modify the loan portfolio report (called either ED.BOOLEAN or <initials>.BOOLEAN) to add the following totals at the end of the report:

- The total balance of all loans
- The total balance of all delinquent loans
- The percentage of the total loan balance that is delinquent (divide the total delinquent balance by the total balance)

The resulting report should look like this:

```
Symitar Credit Union             Loan Portfolio (DQ as of 03/01/18)

Loan Types Selected: 00 to 90

Account#   Member's Name   Type Status     Due Date Last Payment Balance
-------------------------------------------------------------------------
0000123440 JAMES, EDWIN L  0002 Current    03/01/18 02/01/18     1,909.22
0000123444 ROGERS, BELINDA 0000 Delinquent 01/15/18 12/15/18     9,900.00
0000123456 DOW,TOM         0000 Current    03/10/18 02/10/18     2,524.90
0000123456 DOW,TOM         0020 Current    03/15/18 02/10/18     1,000.00
0000123456 DOW,TOM         0060 Delinquent 01/01/18 12/01/18     1,517.20
0000123460 WELBY,MARCUS G  0030 Current    03/01/18 02/01/18   149,227.48
-------------------------------------------------------------------------
Total Loan Balance:                                            743,502.24
Total Loan Balance, Delinquent Loans:                           86,936.60
DQ Loan Ratio:                                                    11.693%
```

Hints

- You need two variables to accumulate the total loan balance and total delinquent loan balance.
- Keep in mind that you need to accumulate the balances of all loans you encounter in one variable, regardless of whether the loan is current or delinquent. By contrast, you need to add to the other variable only if you determine that the loan is delinquent.

Review Exercise 2

This exercise is a review of the following concepts.

- Boolean rules of precedence
- Use of expressions
- Use of the TOTAL division
- The FOR EACH statement

Activity: Review Exercise 2

Specfile Name: <initials>.REVIEW2

We need a report that lists general holds and check holds (**Hold Type** = **(0) General** or **(1) Check**) placed on shares belonging to new accounts (**Account Warning Code** = **(5) New Account**). The report should not list Share Hold records that:

- Are expired
- Belong to a closed share
- Belong to members who are on the Board of Directors (**Account Type = 20**)

List the account number, share ID, share description (uppercase and lowercase letters, not all caps), the share hold type (**General** or **Check**, not **0** or **1**), the share hold amount, and the current share balance.

Accumulate the general hold amount, the check hold amount, and the total hold amount, and print the results in the TOTAL division.

The resulting report should look something like this:

```
Symitar CU                    Share Holds on New Member Accounts

Account#   ID   Description       Balance   Hold Type Hold Amount
-----------------------------------------------------------------
0000000904 0020 90-Day Certificat  3,162.66 General    50.00
0000123441 0000 Primary Share     15,627.71 Check     100.00
                                            Check     400.00
                                            Check     945.65
-----------------------------------------------------------------

Total General Hold Amount:                             50.00

Total Check Hold Amount:                            1,445.65

Total Holds:                                        1,495.65
```

Basic HTML

Hypertext Markup Language (HTML) is the language used to display information within an internet browser. You can use HTML to create formatted output for the Symitar Windows® interface.

Objectives

At the end of this lesson, you will be able to:

- Use HTML tags to
  - Format and display text on the user's screen
  - Change the appearance of backgrounds
  - Define colors
  - Insert graphic elements
  - Create links to URLs
- Explain where graphics should be stored for use in Symitar specfiles

HTML Tags

HTML tags are formatting commands embedded in a document that tell a browser how to display the text that follows or is enclosed within them. The following pages describe the tags you're most likely to need when writing specfiles for Symitar.

HTML tags are not case-sensitive. The browser interprets the uppercase and lowercase versions of each tag identically. For consistency's sake, tags are shown here in all lowercase letters. Many tags also have optional attributes, which are also shown in lowercase letters.

Document Structure

The following HTML tags define the structure of the document.

Tag	**Description**
`<html></html>`	Indicates that this is an HTML document. Optional. If used, these are the first and last tags in the document.
`<head></head>`	Used to mark the start and end of the header in a document. This usually includes descriptive information, such as the title. Optional.
`<title>text</title>`	Marks the start and end of the title for the document. This title appears in the title bar at the top of the browser window, not any title text displayed in the body portion of the browser page.
`<body></body>`	Marks the start and end of the document body. Within these tags, you can specify the document background (see *Graphic Elements* later in this section) and the default colors to be used for links and text, as well as the actual information displayed on the page.

Line Breaks, Paragraph Breaks, and Extra Spaces

The following tags specify line breaks, paragraph breaks, and extra blank spaces.

Tag	Description

`<br />`	Starts a new line with no space between lines.
	`Enter the bill payment amount.<br />`
	`Enter 0 or leave blank to stop the payment.`
	The screen display looks something like this:
	Enter the bill payment amount.
	Enter 0 or leave blank to stop the payment.

`<p></p>`	Starts a new paragraph, creating a new line with an extra space above it.
	`Enter the bill payment amount.<br />`
	`Enter 0 or leave blank to stop the payment`
	`<p>Bill Payment Amount:</p>`
	The screen display looks something like this:
	Enter the bill payment amount.
	Enter 0 or leave blank to stop the payment.
	
	Bill Payment Amount:

`&nbsp;`	Prints a blank space. Most browsers print only one blank space between words or at the beginning of a line, even if you type more.
	`Enter the bill payment amount.<br />`
	`Enter 0 or leave blank to stop the payment`
	`<p>&nbsp;&nbsp;&nbsp Bill Payment Amount:</p>`
	The screen display looks something like this:
	Enter the bill payment amount.
	Enter 0 or leave blank to stop the payment.
	
	Bill Payment Amount:

Justification

By default, all text and other elements on an HTML page are left justified. If you want to center or right-justify certain items (including graphics and tables), you can use the HTML tags described below.

Tag	Description

`<center></center>`	Centers all the elements placed between the `<center>` and `</center>` tags.
	`<center>Bill Payment Info</center>`
	The screen display looks something like this:
	`Bill Payment Info`

`<p align=right></p>`	Right-justifies the elements placed between the `<p align=right>` and `</p>` tags.
	`<p align=right>Payment Amount</p>`
	The screen display looks something like this:
	`Payment Amount`

Emphasis and Fonts

You can specify the color, size, and style (bold, underlined, and so on) of text displayed on the user's screen.

Physical Styles

Tag	Description
`<b>text</b>`	Start and end `bold` text.
`<i>text</i>`	Start and end italicized text.
`<pre>text</pre>`	Start and end preformatted text. This is another way to get a browser to print extra text, since whatever is enclosed in these tags is printed exactly as typed.
`<tt>text</tt>`	`Start and end typewriter style text.`

Tag	Description
`<u>text</u>`	Start and end underlined text.

Font Size, Face, and Color

Tag	Description

`<font size=size color=color face=font name> text</font>`	Changes the size and/or color of the font between the `<font>` and `</font>` tags.
	
	The `size` attribute specifies a font size between 1 and 7 (1 is the smallest, 3 is the default). You can specify a font size from 1 to 7 or add to or subtract from the default size.
	
	The `color` attribute specifies the font color. You can use the color name for one of the 16 standard colors (for example, red or blue), or RGB values to specify another color. See Defining Color Values for more information.
	
	The `face` attribute lets you specify that the browser should display the text using a specific font face (for example, Times New Roman or Arial). You should specify only TrueType fonts. Use this option with care, since it's possible that some people viewing your web page may not have the font you specify. As a rule, we advise against specifying font faces.
	
	You can use the `size`, `color`, and `face` attributes to change all values simultaneously:
	
	`<font color=RED size=5 face=ARIAL>`
	`Warning!</font>`
	`<font size=4>You might find you`
	`enjoy this.</font>`

`<body text=color>`	Use the `text` attribute with the body tag to specify a default color for standard text. This lets you override the font color throughout a document.

Tag	Description
	This is especially useful if you are using a dark background color or graphic. `<BODY TEXT=WHITE>`

Headers

HTML provides tags which let you create headers.

Tag	Description

`<h1>text</h1>`
`<h2>text</h2>`
`<h3>text</h3>`
`<h4>text</h4>`
`<h5>text</h5>`
`<h6>text</h6>`
	Defines text as a header. The header level determines the font size used to display the header. Levels range from 1 (the largest font size) to 6 (the smallest font size).
	`<h1>This is a level 1 header</h1>`
	`<h2>This is a level 2 header</h2>`
	`<h3>This is a level 3 header</h3>`
	The screen display looks like this:
	`This is a level 1 header`
	`This is a level 2 header`
	`This is a level 3 header`

Lists

Tag	Description

`<ul type=bullet>`
`<li>text</li>`
`<li>text</li>`
`<li>text</li>`
`</ul>`
	Builds an unordered (bulleted) list. The possible values for the `type` attribute are `disc` (for a solid bullet), `circle` (hollow bullet), or `square` (square solid bullet). Each item in the list must be prefaced with the `<li>` tag and followed by the `</li>` tag.
	`<ul type=circle>`
	`<li>New Auto</li>`
	`<li>Used Auto</li>`
	`<li>Other</li>`
	`</ul>`
	The screen display looks like this:
	■ New Auto
	■ Used Auto

Tag	Description
	■ Other

`<ol type=marker>`
 	
`<li>text</li>`
 	
`<li>text</li>`
 	
`<li>text</li>`
 	
`</ol>`
	Builds an ordered (numbered) list. The value of the `type` attribute indicates the type of markers to use for the list:
	
	type=1 for numbers (1, 2, 3)
	
	type=A for capital letters (A, B, C)
	
	type=a for lowercase letters (a, b, c)
	
	type=I for capital Roman numerals (I, II, III)
	
	type=i for lowercase Roman numerals (i, ii, iii)
	
	The <li> tag precedes each item in the list.
	
	`<ol type=A>`
	`<li>New Auto</li>`
	`<li>Used Auto</li>`
	`<li>Other</li>`
	`</ol>`
	
	The screen display looks like this:
	
	A New Auto
	
	B Used Auto
	
	C Other

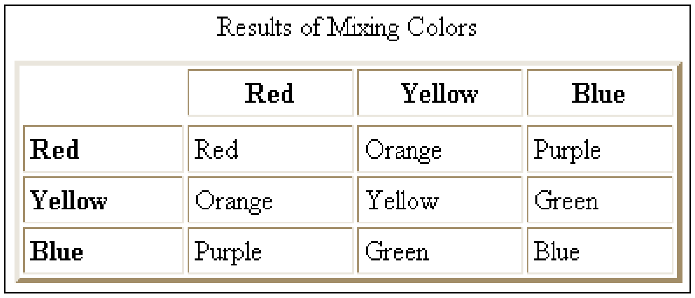
Tables

Tables are an excellent way to format browser displays, even if you're not putting out standard, tabular data. Because table tags can be hard to understand unless you see how they operate together to create a table, we'll start by looking at a sample table in HTML and then discuss the individual tags.

Sample Table in HTML

```
<table border=3 width=50% cellpadding=3>
<caption align=top>Results of Mixing Colors</caption>
<tr>
  <th></th>
  <th>Red</th>
  <th>Yellow</th>
  <th>Blue</th>
</tr>
<tr>
  <th align=left>Red</th>
  <td>Red</td>
  <td>Orange</td>
  <td>Purple</td>
</tr>
<tr>
  <th align=left>Yellow</th>
  <td>Orange</td>
  <td>Yellow</td>
  <td>Green</td>
</tr>
<tr>
  <th align=left>Blue</th>
  <td>Purple</td>
  <td>Green</td>
  <td>Blue</td>
</tr>
</table>
```

The resulting table appears in a browser window like this:



Results of Mixing Colors

	Red	Yellow	Blue
Red	Red	Orange	Purple
Yellow	Orange	Yellow	Green
Blue	Purple	Green	Blue

Tag	Description
`<table border=# bgcolor=color width=%/# cellpadding=#> table headers, rows, and data</table>`	Defines the start and end of a table. All other table tags must fall between the <table> and </table> tags. By default, there are no borders (or lines) around cells. To specify a border with a width of 1 pixel, add the border attribute without a width. To specify

Tag	Description
	a border with a width of 2 pixels, add border=2, or to specify a 3-pixel border, add border=3. You can specify a background color for a table cell using the bgcolor attribute. See Defining Colors later in this section for more information about specifying colors. When you define a table, the browser automatically makes the table wide enough to accommodate the data within the cells. If you want a table to span the entire viewable screen or a specific percentage of the screen, add the width attribute followed by a width in pixels or a percentage of the screen. For example, to make the table span the entire screen regardless of screen resolution, add width=100%. To create a table that is always 500 pixels wide, add width=500. The cellpadding attribute lets you specify the number of pixels between the borders of a cell and the text within. By default, this is 1 pixel, which is usually not pleasing to the eye, so you will usually want to increase this value. For example, to increase the cell padding to 3 pixels, add cellpadding=3.
`<caption align=alignment> text</caption>`	Optional. Specifies a caption for the table. If used, it must come immediately after the <table> tag. Possible values for the align attribute are top (default) and bottom.
`<tr>table data</tr>`	Specifies the table rows. The number of table rows is determined by how many <tr> tags appear in the table.

Tag	Description
	Within these tags, you must specify the headers and data. Each header cell must be preceded and followed by the <th> and </th> tags, and each data cell must be preceded and followed by the <td> and </td> tags (see next page).
`<th align=horalign valign=vertalign nowrap bgcolor=color colspan=# rowspan=#> text</th>`	Specifies the headers for the table. The text within the `<th>` and `</th>` tags is displayed in bold type. The `align` attribute indicates how you want the header text to be aligned horizontally: left, center (default for headers), or right. `valign` indicates how you want the header text to be aligned vertically: top, middle (the default), bottom, or baseline. `nowrap` indicates that you don't want the text within the cell to wrap. `bgcolor` specifies a background color for this cell in the table. `colspan` indicates the number of columns this cell should span horizontally (the default is 1). For example, to create a header cell that spans three data cells below, specify colspan=3. `rowspan` indicates the number of rows this cell should span vertically (the default is 1). For example, to create a row cell that spans three data cells down, specify rowspan=3.
`<td align=horalign valign=vertalign nowrap bgcolor=color colspan=# rowspan=#> text</td>`	Specifies the data within each table cell. These tags must fall between the <tr> and </tr>tags. Table data cells can have the same attributes as table header cells (see

Tag	Description
	above). By default, the align attribute is left. For all other attributes, the defaults are the same as described for table header cells.

Colors

With HTML you can specify the color of a variety of elements. To specify a color in an HTML tag, you use a six-digit hexadecimal value. The Red, Green, Blue (RGB) Color Values for 16 Standard Colors below define the hexadecimal values for the 16 standard colors that were originally supported by early color monitors. These days, most monitors support any "browser-safe" color.

RGB Color Values for 16 Standard Colors

Aqua = #00FFFF

Black = #000000

Blue = #0000FF

Fuchsia = #FF00FF

Gray = #808080

Green = #008000

Lime = #00FF00

Maroon = #800000

Navy = #000080

Olive = #808000

Purple = #800080

Red = #FF0000

Silver = #C0C0C0

Teal = #008080

White = #FFFFFF

Yellow = #FFFF00

Other Browser-Safe Colors

There are many websites that provide the RGB values for browser-safe colors for use in web design. You can find a particularly good one, which provides a large sample block for each color, at the following URL:

https://websafecolors.info/color-chart

http://www-cdr.stanford.edu/~petrie/216colrs.htm

https://www.w3schools.com/colors/colors_shades.asp

Another website you may find useful provides all possible color values (including "unsafe" colors) and their names is:

https://www.w3schools.com/colors/colors_picker.asp

Graphic Elements

The following tags let you work with graphic elements.

Tag	**Description**

`<img src=URL align=alignment border=# width=# height=#>`	Defines a graphic to be displayed
	
	The src (source) attribute specifies the name and location (the URL) of the graphic to be displayed. You can use an absolute or relative path to the file.
	
	An absolute path provides the full path from the root of the drive to the file:
	
	`src=/SYM/SYM000/WIN/HTML/ ourlogo.gif`
	
	A relative path directs the browser to look for the file in a location relative to that of the file being viewed. If the file is in the same directory as the current file, only the file name is needed.
	
	`src=ourlogo.gif`
	
	If the file is in a subdirectory of the directory containing the current file, specify the name of the directory followed by the name of the file:
	
	`src=icons/ourlogo.gif`
	
	If the path starts one level up from the current directory, specify the directory's name preceded by two periods and a slash (../). For example, the following finds a file located in `/SYM/SYM000/WIN/icons` when the current file is in /`SYM/SYM000/WIN/ HTML`:
	
	`src=../icons/ourlogo.gif`
	
	The align attribute determines how the image is aligned with surrounding text. Possible values are top, middle,

Tag	Description
	bottom, texttop, absmiddle, baseline, absbottom, right, and left. The border attribute provides a border around the graphic, if desired. width and height specify the width and height of the graphic in pixels. Although not required, we recommend using them because images with specified widths and heights load more quickly. To determine the width and height of a graphic in pixels, you can insert it into your web page without the specifications and view the page. Right-click the image and select Properties. The dialog box that appears provides the width and height in pixels.
`<img src=URL align=alignment border=# width=# height=#>`	A complete tag to display your logo might look something like this: `<img src=icons/ourlogo.gif align= texttop width=150 height=175>`
`<body background=URL>`	Defines a graphic image as the background display for every page. The URL attribute specifies the name and location (the URL) of the graphic to display. As with the <img> tag, you can use either an absolute or a relative path to the graphic. For example: `<body background=graymarble.gif>` A graphic image defined with this tag is repeated in a tiled format to fill the entire page. Be very careful to choose an image which can be tiled and which can produce a background without gaps or seams. If you are not an experienced HTML user, we encourage you to get expert help.

Tag	**Description**
`<body bgcolor=color>`	Defines a color to be displayed as the background color for the page. You can use the color name for one of the 16 standard colors (for example, red or blue) or RGB values to specify another color. See `Defining Color Values` for more information. `<BODY BGCOLOR=GRAY>`
`<hr align=alignment width=#/% size=pixels color=color />`	Inserts a horizontal line. `align` options are left (default), right, or center. `width` can be either a set number of pixels or a percentage of the page width. `size` is a set number of pixels that specifies how thick the line is. `color` sets the color of the line. `<hr align=center width=90% color=RED size=2 />` This example defines a red line, two pixels thick, centered, and spanning 90% of the page width.

Links

You may want to create links within Symitar specfiles to other sites within your credit union's intranet or on the internet itself.

Tag	**Description**

`<a href=URL target=new> text</a>`	Provides a clickable link which takes the user to the specified file or internet location. You can specify the URL attribute using either a relative path or an absolute path from the current document to the linked document.
	
	The target=new option specifies that the browser should open the document in a separate, new window. This is especially useful when you're directing a user to another website while keeping your website open in the browser.
	
	For example, this is a link to the NYStats.html file located in the AtlanticStates subdirectory:
	
	`<a href=AtlanticStates/NYStats.html>`
	`New York</a>`
	
	It's important to note that AIX® is a case-sensitive operating system where file names are concerned, while DOS and the Mac OS are not. Because your web server runs the AIX operating system, you must be careful to use uppercase and lowercase letters properly in your links.
	
	You can also place an image between the <a href="URL"> and </a> tags. This produces a clickable graphic which opens the specified document.

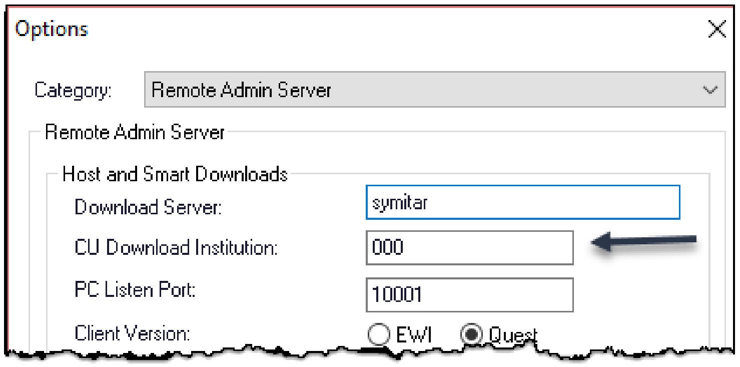
`<body link=color vlinl=color alink=color>`	You can use the link, vlink, and alink attributes to control the color of links, followed links (that is, a link a user has previously clicked), and active links. When you use background colors

Tag	Description
	other than white or very light colors, it's advisable to hard code these values so that the links are displayed clearly on the user's screen. `<body bgcolor=black link=white vlink=silver alink=#ffff00>`

File Locations

You can store HTML documents and graphics for display in Symitar on your host system.

You should put these files in `/SYM/SYMxxx/WIN/HTML`, where xxx is the three-digit number in the CU Download Institution field. You specify this value in the Options selection from the View menu, under the Remote Admin Server category:



Category: Remote Admin Server
Download Server: symitar
CU Download Institution: 000
PC Listen Port: 10001
Client Version: EWI [✓ Quest]

This is the institution from which people download the Symitar software, which is not necessarily the same institution where you are developing the specfile.

Whenever a user logs on to a SYM directory, Symitar looks to see if there's anything new to download in the download directory and does so if necessary. Note that this happens when the user logs in to any SYM directory, not just the download institution. The files end up in the <Symitar Install Location> \SFW \HTML folder on the PCs. This is the location at which you initially installed Symitar on the PC. The default location is `C:\Program Files\Symitar`.

To reference a file that has been copied to the download institution, simply use the file name. No additional path information is required.

For links to files or locations that don't reside on your host system:

- To display a page from a website or internal web server, use the full URL to the website, beginning with http://. For example, if your credit union's intranet resides on a computer with an IP address of 1.1.1.5, you could make a link to a page as shown below:

  ```
  <a HREF=http://1.1.1.5/policies.html>
  ```
- To display a file that resides on your credit union's file server, use the full path, beginning with file://. For example, to display a document in rich text format (RTF) from the S:\ drive on your web server, you could make a link to the file as shown below:

  ```
  <a HREF=file://S:\policies.rtf>
  ```

Using PC Transfer

You can use the PC Transfer work area to transfer files between an IBM®-compatible PC and the Symitar host without any additional software.

Steps

1. Click **PC Transfer**.
   The PC Transfer window opens, including a toolbar and *Local System* and *Host System* panes. The title of each pane indicates the path to the directory currently displayed. This path includes the drive and any directories above the one displayed. The title changes as you display different directories. The initial directories that display when you enter the PC Transfer work area are the following:

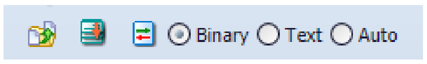
   Local System

   This is initially the directory from which you run Symitar. If you have the appropriate permissions and are on a network, you can move to other directories or other drives (which can include disk drives, network servers, other systems, etc.).

   Host System

   Initially this is the root of the SYM institution you are logged into. For security reasons, Symitar displays only those directories to or from which the user can transfer files. You can't go to another SYM directory.

   The PC Transfer toolbar appears at the top of the *PC Transfer* window, just below the Symitar toolbar:

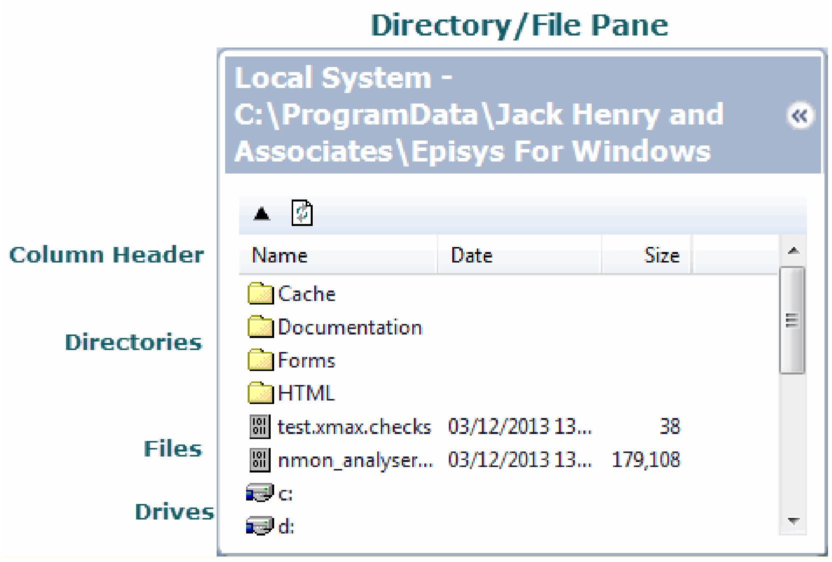
   

2. Use the radio buttons to select the transfer mode:

   - Select **Binary** if the file to be transferred is binary (usually a data or executable file). These are basically any files which are not text (ASCII) files.
   - Select **Text** if the file to be transferred is a text (ASCII) file.
   - Select **Auto** if you want Symitar to select the mode automatically. A list of file extensions in the system registry identifies whether

a file is a binary or text file. The icon in front of each file name indicates the file's type.

The System Pane directory and file list display below the pane toolbar.



Local System - C:\ProgramData\Jack Henry and Associates\Episys For Windows

Name	Date	Size
Cache		
Documentation		
Forms		
HTML		
test.xmax.checks	03/12/2013 13...	38
nmon_analyser...	03/12/2013 13...	179,108
c:		
d:		

You can identify a directory in the system pane directory or file list by the folder icon in front of the name. To display the contents of any directory, double-click anywhere on the line displaying the directory name.

Files displayed in the system pane include the following information on the same line:

Mode icon

This icon appears immediately in front of the file name, under the Name column header. This icon indicates the type of file (as well as Symitar can determine). A list of file extensions in the system registry and system defaults helps Symitar identify each file as one of the following:

- If a file is not an ASCII text file (usually data and executables), it is a binary file:
- If a file extension indicates an ASCII text file, or if the file is a report file, letter file, or specfile on the host system, it is a text file:
- A special mode icon represents the host system "edit" data file. This binary file is displayed whether or not it currently contains

any data, but the icon is dimmed if the file does not contain any data:

These icons represent the mode in which Symitar will attempt to transfer the file if you select a transfer mode of **Auto**. You can set the type of transfer manually, as well.

3. To transfer files from the local system to the host system or vice versa, select the file to be transferred in one pane and drag it to the appropriate location in the other pane.

Activity: Design a Web Page

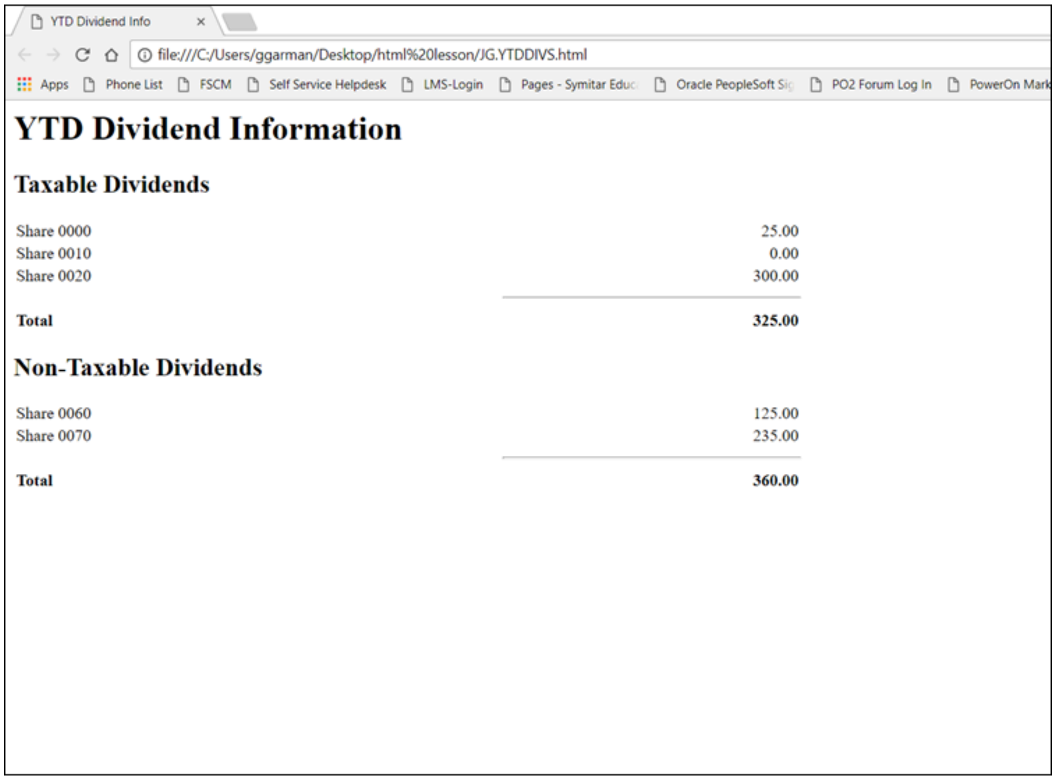
To practice using the HTML tags you have just learned, you will create a web page layout you can use in Symitar to display taxable and non-taxable dividend information for all shares in a member's account (see the example on the following page).

In this lesson, all you need to do is code the HTML shell. You can hard code mock data in the web page that you'll replace in the next lesson with information pulled from the member's account.

- Use Notepad to create a file and save it with an .html extension to the hard drive on your PC. After saving and closing the file, reopen it in Internet Explorer®, and then select **View** > **Source** from the menu to open the code in Notepad. When you make changes to the code in the Notepad file, be sure to save them and click the **Refresh** button in Internet Explorer to load the new version of the page.

TIP
Giving the file an .html extension causes Symitar to open it automatically in Internet Explorer when you double-click the icon.

- Be sure your web page uses a combination of the design elements discussed in this lesson. At a minimum, be sure to include:
  - A table with at least two rows
  - Font color changes
  - Bold, italic, or underlined text


YTD Dividend Info

file:///C:/Users/ggarman/Desktop/html%20lesson/JG.YTDDIVS.html

Apps | Phone List | FSCM | Self Service Helpdesk | LMS-Login | Pages - Symitar Educ | Oracle PeopleSoft Si | PO2 Forum Log In | PowerOn Mark

YTD Dividend Information

Taxable Dividends

Share 0000	25.00
Share 0010	0.00
Share 0020	300.00
Total	**325.00**

Non-Taxable Dividends

Share 0060	125.00
Share 0070	235.00
Total	**360.00**

Figure 4: Sample Web Page

HTML Displays

Three PowerOn statements let Symitar display HTML-formatted output from demand specfiles. This functionality lets you create attractive screen displays that are often suitable for print. In many cases, you may be able to convert demand specfiles that use laser commands to HTML displays, thus allowing the user to view the results on-screen before printing the form or letter.

At the end of this lesson, you will be able to write demand specfiles for Symitar that display HTML as output.

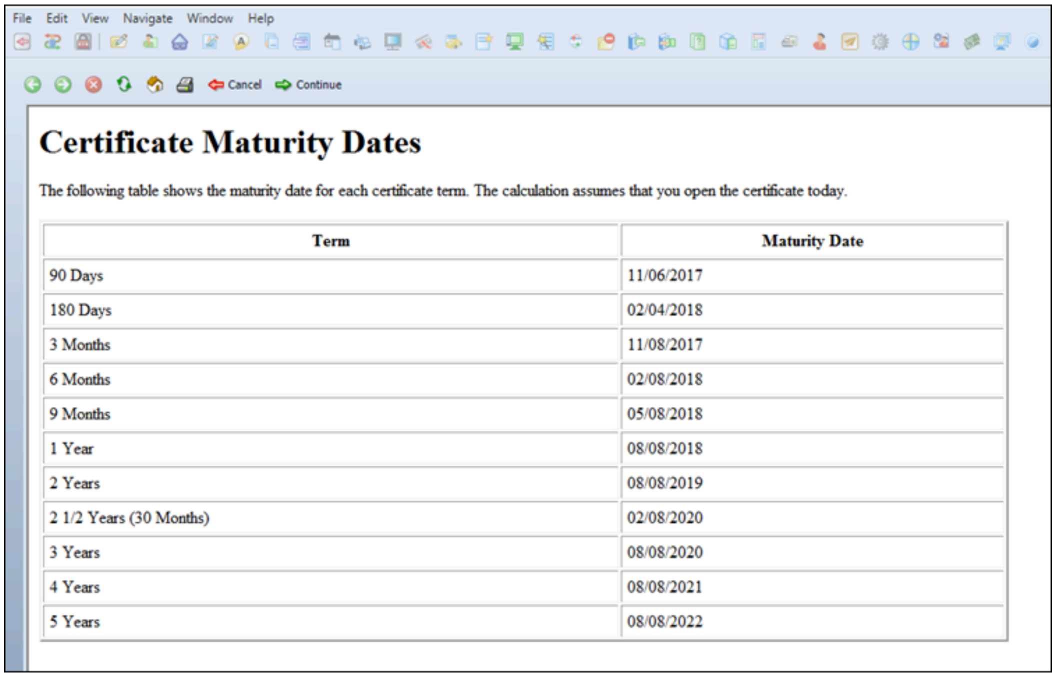
Sample Specfile

The following specfile lets a teller show a member when certificates of varying terms would mature if opened today. The use of HTML display statements creates an attractive layout for both screen viewing and print (see next page).

```
WINDOWS

TARGET=ACCOUNT

PRINT TITLE="Certificate Maturities"
 HTMLVIEWOPEN
 HTMLVIEWLINE("<head><title>Certificate Maturity Dates")
 HTMLVIEWLINE("</head></title><body>")
 HTMLVIEWLINE("<h1>Certificate Maturity Dates</h1>")
 HTMLVIEWLINE("<p>The following table shows the maturity ")
 HTMLVIEWLINE("date for each certificate term. The ")
 HTMLVIEWLINE("calculation assumes that you open the ")
 HTMLVIEWLINE("certificate today.</p>")
 HTMLVIEWLINE("<table cellpadding=5 border=2 width=50%>")
 HTMLVIEWLINE("<tr><th>Term</th><th>Maturity Date</th></tr>")
 HTMLVIEWLINE("<tr><td>90 Days</td><td>")
 HTMLVIEWLINE(FORMAT("99/99/9999",SYSTEMDATE+90))
 HTMLVIEWLINE("</td></tr>")
 HTMLVIEWLINE("<tr><td>180 Days</td><td>")
 HTMLVIEWLINE(FORMAT("99/99/9999",SYSTEMDATE+180))
 HTMLVIEWLINE("</td></tr>")
 HTMLVIEWLINE("<tr><td>3 Months</td><td>")
 HTMLVIEWLINE(FORMAT("99/99/9999",DATEOFFSET(SYSTEMDATE,3,0)))
 HTMLVIEWLINE("</td></tr>")
 HTMLVIEWLINE("<tr><td>6 Months</td><td>")
 HTMLVIEWLINE(FORMAT("99/99/9999",DATEOFFSET(SYSTEMDATE,6,0)))
 HTMLVIEWLINE("</td></tr>")
 HTMLVIEWLINE("<tr><td>9 Months</td><td>")
 HTMLVIEWLINE(FORMAT("99/99/9999",DATEOFFSET(SYSTEMDATE,9,0)))
 HTMLVIEWLINE("</td></tr>")
 HTMLVIEWLINE("<tr><td>1 Year</td><td>")
 HTMLVIEWLINE(FORMAT("99/99/9999",DATEOFFSET(SYSTEMDATE,12,0)))
 HTMLVIEWLINE("</td></tr>")
 HTMLVIEWLINE("<tr><td>2 Years</td><td>")
 HTMLVIEWLINE(FORMAT("99/99/9999",DATEOFFSET(SYSTEMDATE,24,0)))
 HTMLVIEWLINE("</td></tr>")
 HTMLVIEWLINE("<tr><td>2 1/2 Years (30 Months)</td><td>")
 HTMLVIEWLINE(FORMAT("99/99/9999",DATEOFFSET(SYSTEMDATE,30,0)))
 HTMLVIEWLINE("</td></tr>")
 HTMLVIEWLINE("<tr><td>3 Years</td><td>")
 HTMLVIEWLINE(FORMAT("99/99/9999",DATEOFFSET(SYSTEMDATE,36,0)))
 HTMLVIEWLINE("</td></tr>")
 HTMLVIEWLINE("<tr><td>4 Years</td><td>")
 HTMLVIEWLINE(FORMAT("99/99/9999",DATEOFFSET(SYSTEMDATE,48,0)))
 HTMLVIEWLINE("</td></tr>")
 HTMLVIEWLINE("<tr><td>5 Years</td><td>")
 HTMLVIEWLINE(FORMAT("99/99/9999",DATEOFFSET(SYSTEMDATE,60,0)))
 HTMLVIEWLINE("</td></tr></table></body>")
 HTMLVIEWDISPLAY
END
```



Certificate Maturity Dates

The following table shows the maturity date for each certificate term. The calculation assumes that you open the certificate today.

Term	Maturity Date
90 Days	11/06/2017
180 Days	02/04/2018
3 Months	11/08/2017
6 Months	02/08/2018
9 Months	05/08/2018
1 Year	08/08/2018
2 Years	08/08/2019
2 1/2 Years (30 Months)	02/08/2020
3 Years	08/08/2020
4 Years	08/08/2021
5 Years	08/08/2022

Figure 5: Screen Output

HTML Display Statements

You can use the following PowerOn statements to create HTML displays.

- HTMLVIEWOPEN
- HTMLVIEWLINE
- HTMLVIEWDISPLAY

HTMLVIEWOPEN Definition and Syntax

The HTMLVIEWOPEN statement initiates an HTML view window. You can use HTMLVIEWOPEN in the SETUP division or the PRINT division, or in a PROCEDURE called from those divisions.

The syntax is:

```
HTMLVIEWOPEN(0)
```

You should follow the HTMLVIEWOPEN statement with as many HTMLVIEWLINE statements as you need to output all the code for your HTML page.

HTMLVIEWLINE Definition and Syntax

The HTMLVIEWLINE statement defines a line of HTML tags and text to be displayed as output. You can use HTMLVIEWLINE in the SETUP division or the PRINT division or in a PROCEDURE called from these divisions.

The syntax is:

```
HTMLVIEWLINE(HTMLLine)
```

HTMLLine is a line of HTML tags and text, typically enclosed in quotation marks.

To print data from system fields or variables within an HTML line, use character addition. Everything within the parentheses must have a data type of character, so if you print data from non-character fields or variables, you must use the FORMAT function to change its data type.

For example:

```
HTMLVIEWLINE("<table><tr><td>"+
             FORMAT("99/99/99",SYSTEMDATE)+
             "</td></tr></table>")
```

HTMLVIEWDISPLAY Definition and Syntax

The HTMLVIEWDISPLAY statement displays the HTML page defined by the HTMLVIEWLINE statements that precede it. You can use HTMLVIEWDISPLAY in the SETUP division or the PRINT division, or in a PROCEDURE called from these divisions.

The syntax is:

```
HTMLVIEWDISPLAY
```

Other Issues and Requirements

When you write HTML display specfiles, keep the following points in mind.

- To use HTML view statements, your specfile must begin with the WINDOWS keyword.
- Once you invoke the WINDOWS keyword at the beginning of your specfile, you cannot use standard PRINT or COL statements anywhere in your specfile.
- When writing HTML displays, it is important to know that Symitar Quest has a setting that determines whether HTML displays launch in Internet Explorer® or Chromium™. Since Microsoft® is ending support for Internet Explorer soon, you should avoid using any HTML or JavaScript™ features that are Internet Explorer-specific, since these pages may not display correctly in Chromium. For more information, refer to the Chromium landing page.

Shortcuts for HTML Display

To make writing HTML display specfiles easier, we recommend that you write the HTML first. You can lay out the page by coding the HTML yourself in a text editor or by using a WYSIWYG HTML editor such as Dreamweaver® Microsoft® Expression Web.

Once you're satisfied with the HTML, edit the HTML file and break the lines down to fewer than 132 characters per line (and preferably to lines of approximately 80 characters each). It doesn't matter where you break the lines, but you should try to break them in a way that will make it easy for you to insert references to fields, variables, and other information.

When you've finished editing the file, send it via FTP to your letter files directory (for procedures, see *Using PC Transfer* in the *Basic HTML* lesson).

After you have the raw HTML file in your letter files directory, you can run the `RD.CONVERT.HTML` library specfile.

> **TIP**
> Be sure to install the specfile for demand use first!

The specfile prompts for the name of the HTML letter file to convert. The specfile creates a new specfile with the same name as the original HTML file, with the following changes:

- It adds the HTMLVIEWLINE statement to each line.
- It converts any quotation marks in your HTML to CTRLCHR(34).
- It converts any unprintable characters to HTML special characters.

Once you have converted the HTML to specfile code, you can edit it to include standard PowerOn statements and functions, information from fields and variables, and so on.

Activity: Writing an HTML Display Specfile

Specfile Name: <initials>.YTDDIVS

In the last lesson, you created a YTD dividends web page that contained hardcoded data. In this lesson, you'll take that web page and convert it to a specfile that pulls the data from a member's account.

To begin, transfer the file from your PC via FTP to the letter files directory in SYM 0. Run the RD.CONVERT.HTML specfile to convert the HTML code to specfile code. Once you've completed the conversion, you'll need to work through the specfile to replace the hardcoded data with member account data.

Appendix A

This appendix contains exercises that were previously part of the PowerOn 101 course, but were removed to allow sufficient time to cover HTML. We are including these exercises in this appendix to provide additional practice for students.

We suggest that you use the exercises in this appendix as self-study materials. You'll find suggested solutions for each activity at the end of Appendix A.

Practice 1

This exercise is a review of the following PowerOn statements and functions you have learned:

- IF...THEN
- Read literals
- FORMAT

Activity: Practice 1

Specfile name: <initials>.PRACTICE1

We need a report that lists each unexpired Share Transfer record in Symitar that is subsidiary to an open share with a type within a range entered by the user.

For each Share Transfer record, we need two lines of information:

- The first line should list the member's account number and the **Share ID**, **Description**, and **Balance** field values of the share.
- The second line should indicate the type of transfer and where the transfer is coming from or going to:
    - For overdraft Transfer records, print Overdraw Transfer from Account#.
    - For dividend Transfer records, print Dividend Transfer to Account#.
    - For maturity Transfer records, print Maturity Transfer to Account#.
    - For automatic Share Transfer records, print Auto Transfer to Account#.
- In all cases, follow the text described above with the share transfer account number, an **S** if the share transfer **ID Type** is **0** or an **L** if the **ID Type** is **1**, and the share transfer ID.
- Finally, if this is a maturity transfer, print the word "on" followed by the share **Maturity Date** field value.

Please double-space the report. See the next page for a sample report.

```
Symitar CU    Share Transfers for Types 00 to 99

Account#     ID    Description                          Balance
------------------------------------------------------------
0000123494   0000  PRIME SHARE                           190.17
    Overdraw Transfer from Account#  0000123494 S0010

0000123494   0020  PRIME SHARE                         2,920.85
    Dividend Transfer to Account#    0000123494 S0010

0000123494   0020  30 DAY CERTIFICATE                 56,293.04
    Maturity Transfer to Account#    0000123494 S0010 on 03/31/2018
```

Hints

- You'll need to consult **Programming** > **Files, Records, and Fields** > **Account File** in the *Symitar eDocs* to determine which Transfer record field indicates the type of transfer and what its valid values are.
- Remember that PowerOn can look up in the targeted record's record path. This means that if you target the Share Transfer record, you still have access to information in the Account and Share records.

- To select Share Transfer records that belong to shares with a type in a range specified by the user, you need to prompt the user for two values: the lowest type and the highest type. In your SELECT division, you must look for shares with a **Share Type** field value greater than or equal to the lowest value and less than or equal to the highest value.
- Be sure to indicate in the report’s title which share type range the user entered.

Practice 2

This exercise provides additional practice using:

- Mathematical expressions
- Arithmetic functions
- Enter functions
- Date functions

Activity: Practice 2

Specfile name: <initials>.PRACTICE2

Your tellers would like a demand specfile that allows them to print a stop payment order form and that calculates both the fee and the expiration date for the stop payment.

Prompt the tellers to enter the **ID** of the draft share and the number of the first and last check to stop. The prompt should indicate that a **0** entry at the second prompt means there is only one check to stop. The input screen should look something like this:

```
Share draft ID :0050
First check to stop [0] :671
Last check to stop (0 for none) [0] :676
```

Calculate the fee for stop payments as follows:

- If only one check is being stopped, the fee is a flat $8.50.
- If multiple checks are being stopped, the fee is $8.50 plus $0.50 per check stopped (including the first check). After calculating that amount, add an additional 10%.

Tellers will print the resulting output by entering **,PO** after the demand specfile name. The form must include the member's name and mailing address, the member's account number, the Share **ID** field value, and the check number or range of check numbers being stopped. Indicate the total fee amount for the stop payment and the date the stop expires (always six months from today). At the bottom of the form, create a line for the member's signature, a space for the member to write today's month and day, and then print the year of the system date.

See the next page for an example of the form.

```
LINDA MARIE DIXON
SYMITAR CU
1215 YORK ROAD
LUTHERVILLE, MD 21093


You have requested that Symitar Credit Union place a stop payment on the
check or range of checks listed below:

  Account# 0000123457 Share 0050 Check# 671-676

This stop payment order expires on 06/22/19. The fee for this stop payment
order is $12.65.



_________________________________________  ___________ ____, 2018
Member's Signature                         Month       Day
```

Practice 3

This exercise is a review of the following concepts:

- Date functions
- The FOR EACH statement

Activity: Practice 3

Specfile name: <initials>.PRACTICE3

Your tellers need a demand specfile to calculate the total dividend paid to all of a member's shares and the total interest paid on loans for the year to date. Calculate the amount paid to regular shares separately from the amount paid to tax-deferred shares (IRAs, Roth IRAs, Coverdell ESAs, etc.). You can calculate the interest paid to all loans as one lump sum.

- A share is normal if its **IRS Code** is **(0) Normal**; it is tax-deferred if its **IRS Code** is anything greater than **0**.
- Include closed shares and closed loans in the calculation, since the member may have received dividend payments for a share or paid interest to a loan that has since been closed.
- The Share record field that stores the year-to-date dividend payments has the mnemonic DIVYTD. The Loan record field that stores the year-to-date interest received has the mnemonic INTERESTYTD.

The output looks like this:

```
Total Taxable Dividend Paid in 2018:        933.37

Total Non-Taxable Dividend Paid in 2018:    398.45

Total Interest Paid in 2018:                453.22
```

Hint

Use the FULLYEAR function to find the year of the system date so that you don't have to edit this specfile on December 31 each year.

Solutions

Solutions to the practice exercises.

XXX.PRACTICE1

```
TARGET=SHARE TRANSFER

DEFINE
 LOWTYPE=NUMBER
HIGHTYPE=NUMBER
END

SETUP
 LOWTYPE=NUMBERREAD("Enter lowest Share Type")
 HIGHTYPE=NUMBERREAD("Enter highest Share Type")
END

SELECT
 SHARE:CLOSEDATE='--/--/--' AND
 (SHARE TRANSFER:EXPIRATIONDATE='--/--/--' OR
  SHARE TRANSFER:EXPIRATIONDATE>=SYSTEMDATE) AND
 (SHARE:TYPE>=LOWTYPE AND SHARE:TYPE<=HIGHTYPE)
END

PRINT TITLE="Share Transfers for Types "+FORMAT("99",LOWTYPE)+
            " to "+FORMAT("99",HIGHTYPE)+" - ED"
 HEADERS
  COL=001       "Account#"
  COL=014       "ID"
  COL=020       "Description"
  COL=060 RIGHT "Balance"
  NEWLINE
  COL=001 REPEATCHR("-",60)
  NEWLINE
 END

 COL=001 ACCOUNT:NUMBER
 COL=014 SHARE:ID
 COL=020 SHARE:DESCRIPTION
 COL=060 SHARE:BALANCE
 NEWLINE

 IF SHARE TRANSFER:TYPE=0 THEN
  DO
   COL=005 "Overdraw Transfer from "
  END
 ELSE IF SHARE TRANSFER:TYPE=1 THEN
  DO
   COL=005 "Dividend Transfer to "
  END
ELSE IF SHARE TRANSFER:TYPE=2 THEN
  DO
   COL=005 "Maturity Transfer to "
  END

 ELSE IF SHARE TRANSFER:TYPE=3 THEN
  DO
   COL=005 "Auto Transfer from "
  END

 PRINT "Account#"
 COL=038 SHARE TRANSFER:ACCOUNTNUMBER

 IF SHARE TRANSFER:IDTYPE=0 THEN
  DO
   PRINT " S"
  END
 ELSE
  DO
```

```
   PRINT " L"
  END

 PRINT SHARE TRANSFER:ID

 IF SHARE TRANSFER:TYPE=2 THEN
  DO
   PRINT " on "+FORMAT("99/99/9999",SHARE:MATURITYDATE)
  END

 NEWLINE
 NEWLINE
END
```

XXX.PRACTICE2

```
TARGET=ACCOUNT

DEFINE
 ID=CHARACTER(2)
 LOWCHECKNO=NUMBER
 HIGHCHECKNO=NUMBER
 CHECKSSTOPPED=NUMBER
 FEE=MONEY
END

SETUP
 ID=ENTERCHARACTER("Share draft ID",2,"")
 LOWCHECKNO=ENTERNUMBER("First check to stop",0)
 HIGHCHECKNO=ENTERNUMBER("Last check to stop (0 for none)",0)
END

PRINT TITLE="Stop Draft Order"
 NEWLINE NEWLINE
 PRINT NAME:LONGNAME
 NEWLINE

 IF NAME:EXTRAADDRESS<>"" THEN
  DO
   PRINT NAME:EXTRAADDRESS
   NEWLINE
  END

 PRINT NAME:STREET
 NEWLINE
 PRINT NAME:CITY+", "+NAME:STATE+" "+NAME:ZIPCODE
 NEWLINE NEWLINE NEWLINE
 PRINT "You have requested that Symitar Federal Credit Union place a "
 PRINT "stop payment on "
 NEWLINE
 PRINT "the check or range of checks listed below: "
 NEWLINE NEWLINE
 PRINT "  Account# "+ACCOUNT:NUMBER
 PRINT " Share "+ID+" Check# "
 PRINT LOWCHECKNO

 IF HIGHCHECKNO<>0 THEN
  DO
   PRINT "-"
   PRINT HIGHCHECKNO
  END

 NEWLINE NEWLINE
 PRINT "This stop payment order expires on "
 PRINT DATEOFFSET(SYSTEMDATE,6,0)
 PRINT ". The fee for this stop payment"
 NEWLINE


PRINT "order is $"

 IF HIGHCHECKNO<>0 THEN
```

```
  DO
   CHECKSSTOPPED=(HIGHCHECKNO-LOWCHECKNO)+1
   FEE=MONEY(($8.50+(CHECKSSTOPPED*$0.50))*1.1)
  END
 ELSE
  DO
   FEE=$8.50
  END

 PRINT FEE
 PRINT "."
 NEWLINE NEWLINE NEWLINE NEWLINE
 PRINT "_________________________________________  "
 PRINT "___________ ____, "
 PRINT FORMAT("9999",FULLYEAR(SYSTEMDATE))
 NEWLINE
 PRINT "Member's Signature"
 COL=044 "Month       Day"
 NEWLINE
END
```

XXX.PRACTICE3

```
TARGET=ACCOUNT

DEFINE
 YTDDIV=MONEY
 YTDINT=MONEY
END

PRINT TITLE="YTD Dividends"
 FOR EACH SHARE WITH (SHARE:IRSCODE=0)
  DO
   YTDDIV=YTDDIV+SHARE:DIVYTD
  END

 PRINT "Total Taxable Dividends Paid in "
 PRINT FORMAT("9999",FULLYEAR(SYSTEMDATE))+": "
 COL=050 YTDDIV
 NEWLINE
 NEWLINE
 YTDDIV=$0.00

 FOR EACH SHARE WITH (SHARE:IRSCODE>0)
  DO
   YTDDIV=YTDDIV+SHARE:DIVYTD
  END

 PRINT "Total Non-Taxable Dividends Paid in "
 PRINT FORMAT("9999",FULLYEAR(SYSTEMDATE))+": "
 COL=050 YTDDIV
 NEWLINE
 NEWLINE

 FOR EACH LOAN
  DO
   YTDINT=YTDINT+LOAN:INTERESTYTD
  END

 PRINT "Total Interest Paid in "
 PRINT FORMAT("9999",FULLYEAR(SYSTEMDATE))+": "
 COL=050 YTDINT
 NEWLINE
 NEWLINE
END
```

Appendix B

The following are charts of the Symitar Account Record Family and the Symitar Database Record Families.

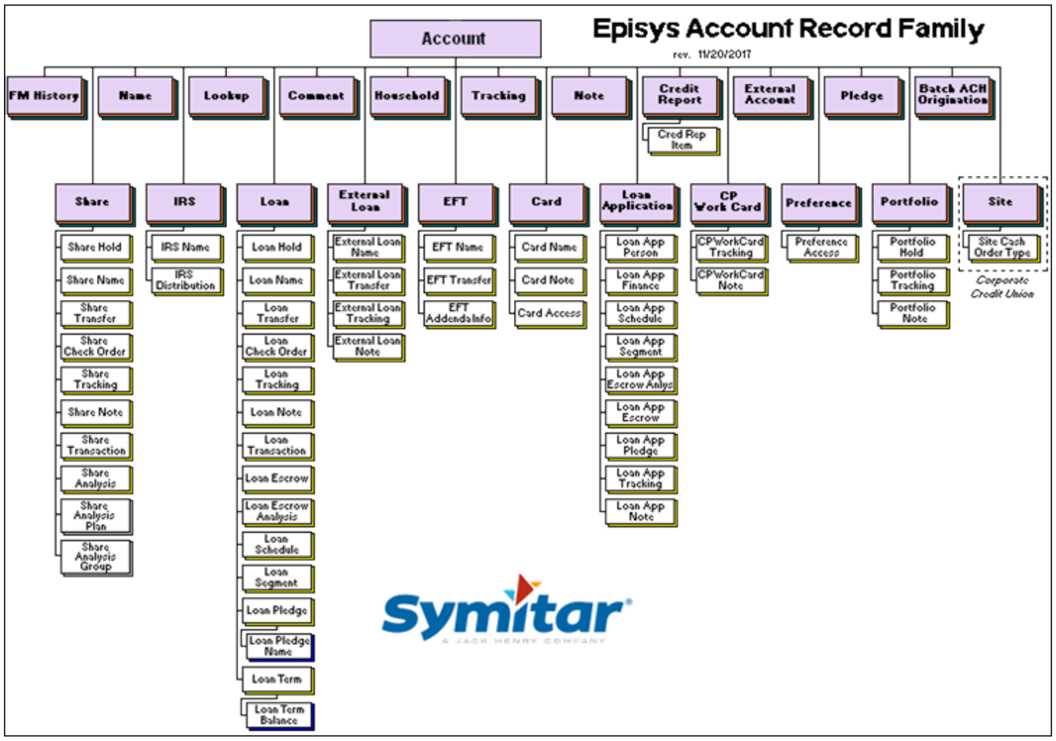


Figure 6: Symitar Account Record Family

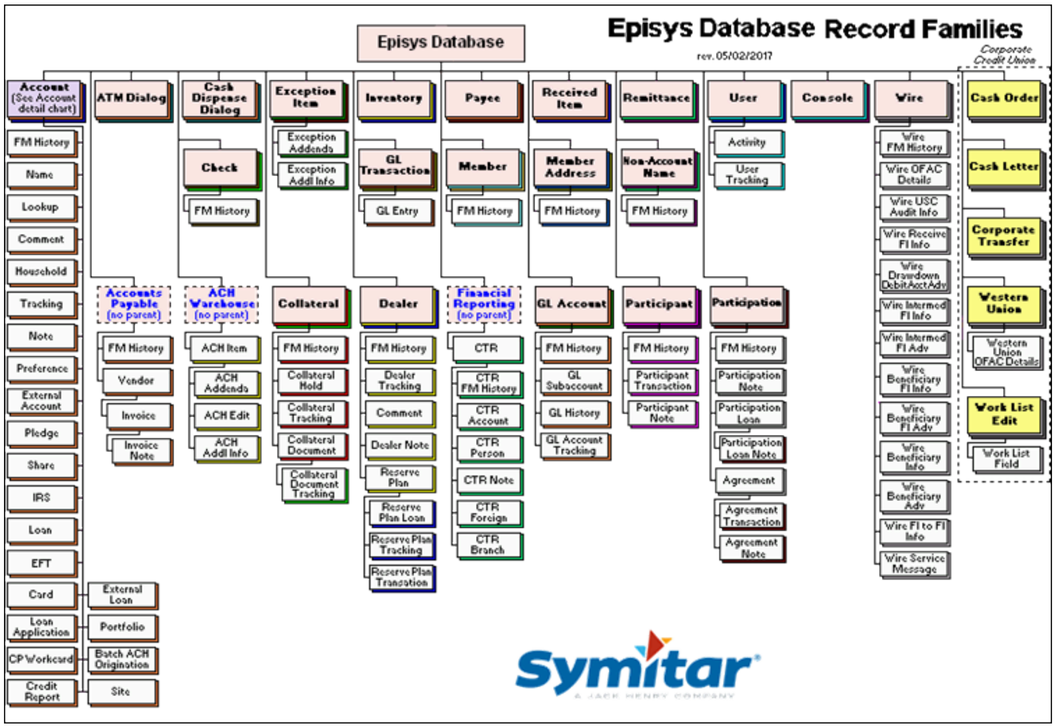


Figure 7: Symitar Database Record Families

Appendix C

The following are suggested solutions to the exercises.

Suggested Solution for <initials>.EZPRAC

```
[ -----------------------------------------------------------------
  ED.EZPRAC - PowerOn Easy Writer
  ----------------------------------------------------------------- ]

TARGET=ACCOUNT

DEFINE
 INPUTVAR001=DATE                      [ Accounts opened on or after  ]
END

SETUP
 INPUTVAR001=DATEREAD("Accounts opened on or after")
END

SELECT
 ACCOUNT:CLOSEDATE='--/--/--' AND
 ACCOUNT:OPENDATE>=INPUTVAR001 AND
 NAME:EMAIL<>""
END

SORT
 ACCOUNT:OPENDATE                      [ First Sort Key               ]
 ACCOUNT:TYPE                          [ Second Sort Key              ]
END

PRINT TITLE="Accounts with Email Addresses - ED"
 HEADER="Account Number  Short Name        Type  E-Mail Address
Open Date"
 HEADER="---------------------------------------------------------------"
 COL=001 ACCOUNT:NUMBER
 COL=017 NAME:SHORTNAME
 COL=038 ACCOUNT:TYPE
 COL=041 NAME:EMAIL
 COL=091 ACCOUNT:OPENDATE
 NEWLINE
END
```

Suggested Solution for <initials>.NAMES

```
TARGET=ACCOUNT

SELECT
 ACCOUNT:CLOSEDATE='--/--/--'
END

PRINT TITLE="Member Names - ED"
 PRINT NAME:LONGNAME
 PRINT ","
 PRINT NAME:STREET
 PRINT ","
 PRINT NAME:CITY
 PRINT ","
 PRINT NAME:STATE
 PRINT ","
 PRINT NAME:ZIPCODE
 NEWLINE
END
```

Suggested Solution for <initials>.COL

```
[List all open share certificates and their maturity date.
 No headers or totaling.]
```

```
TARGET=SHARE

SELECT
 SHARE:CLOSEDATE='--/--/--' AND
 SHARE:SHARECODE=2
END

PRINT TITLE="Share Certificates - ED"
 COL=001  ACCOUNT:NUMBER
 COL=013  NAME:SHORTNAME
 COL=032  SHARE:ID
 COL=045  SHARE:TYPE
 COL=060  SHARE:MATURITYDATE
 COL=080  SHARE:BALANCE
 NEWLINE
END
```

Suggested Solution for <initials>.HEADERS

```
[Lists all open share certificates and their maturity date with headers,
trailers, subtotals by share type, and standard totals.]

TARGET=SHARE

SELECT
 SHARE:CLOSEDATE='--/--/--' AND
 SHARE:SHARECODE=2
END

PRINT TITLE="Share Certificates - ED"
 HEADERS
  COL=001       "Account#"
  COL=013       "Member's Name"
  COL=032       "ID"
  COL=045       "Type"
  COL=060 RIGHT "Mat. Date"
  COL=080 RIGHT "Balance"
  NEWLINE
  COL=001 REPEATCHR("-",80)
 END
 TRAILERS
  COL=001 REPEATCHR("-",80)
 END
 COL=001      ACCOUNT:NUMBER
 COL=013      NAME:SHORTNAME
 COL=032      SHARE:ID
 COL=045      SHARE:TYPE
 COL=060      SHARE:MATURITYDATE
 COL=080      SHARE:BALANCE
 NEWLINE
END
```

Suggested Solution for <initials>.LETTER

```
TARGET=LOAN

SELECT
 LOAN:CLOSEDATE='--/--/--' AND
  LOAN:DUEDATE<SYSTEMDATE AND
 LOAN:DUEDATE>=SYSTEMDATE-30
END

LETTER TITLE="1 To 30 Day DQ Notices - ED"
 CAPITALIZE(NAME:TITLE)
 CAPITALIZE(NAME:LAST)
 SYSTEMDATE
 ACCOUNT:NUMBER
 LOAN:ID
 SYSTEMDATE-LOAN:DUEDATE
 LOAN:PASTDUEAMOUNT
 NAME:LONGNAME
 NAME:STREET
 NAME:CITY
 NAME:STATE
 NAME:ZIPCODE
END

<!--NEWPAGE-->


Dear ^+^ ^+^,                                                ^^^^^^^^

Your Account ^^^^^^^^^^ Loan ^^^^ is currently past due ^+^ days with a
past due amount of $^+^. Perhaps you have forgotten to make your payment
for this loan or mailed it late. Please review your records and make the
payment as soon as possible, or let us know if there is some error on
our part. You can call us toll-free at 1-888-SYMITAR and speak to any
Member Service Representative. At Symitar Credit Union, we value your
continued membership!





                                 ^+^
                                 ^+^
                                 ^+^ ^+^  ^+^
```

Suggested Solution for<initials>.STANDARD.TOTALS

```
[Lists all open share certificates and their maturity date with
 headers, trailers, subtotals by share type, and standard totals.]

TARGET=SHARE

SELECT
 SHARE:CLOSEDATE='--/--/--' AND
 SHARE:SHARECODE=2
END

SORT
 SHARE:TYPE SUBTOTAL="Totals for Share Type:"
END

PRINT TITLE="Share Certificates-Totals by Type - ED"
 HEADERS
  COL=001       "Account#"
  COL=013       "Member's Name"
  COL=032       "ID"
  COL=045       "Type"
  COL=060 RIGHT "Mat. Date"
  COL=080 RIGHT "Balance"
  NEWLINE
  COL=001 REPEATCHR("-",80)
 END
 TRAILERS
  COL=001 REPEATCHR("-",80)
 END
 COL=001      ACCOUNT:NUMBER
 COL=013      NAME:SHORTNAME
 COL=032      SHARE:ID
 COL=045      SHARE:TYPE
 COL=060      SHARE:MATURITYDATE
 COL=080      SHARE:BALANCE TOTAL="Total Share Balance:"
 NEWLINE
END
```

Suggested Solution for <initials>.SUPPRESS

```
[Same as ED.STD.TOTALS, except that all detail information is
 suppressed and the specfile prints only subtotals and totals.]

TARGET=SHARE
```

```
SELECT
 SHARE:CLOSEDATE='--/--/--' AND
 SHARE:SHARECODE=2
END

SORT
 SHARE:TYPE SUBTOTAL="Totals for Share Type:"
END

PRINT TITLE="Share Certificates-Summary Version - ED"
 HEADER=""
 SUPPRESS SHARE:BALANCE TOTAL="Total Share Balance:"
 SUPPRESS SHARE:DIVFROMOPEN TOTAL="Total Dividends Paid:"
END
```

Suggested Solution for <initials>.REVIEW1

```
TARGET=LOAN

SELECT
 LOAN:CLOSEDATE='--/--/--' AND
 LOAN:BALANCE>$0.00
END

SORT
 ACCOUNT:NUMBER SUBTOTAL="Total for Account#:"
END

PRINT TITLE="Current Loan Status by Account - ED"
 HEADERS
  COL=001       "Loan Description"
  COL=035       "ID"
  COL=045 RIGHT "Type"
  COL=060 RIGHT "Open Date"
  COL=075 RIGHT "Last Pmt"
  COL=090 RIGHT "Due Date"
  NEWLINE
  COL=001 REPEATCHR("-",90)
 END
 TRAILERS
  COL=001 REPEATCHR("=",90)
 END
 COL=001 CAPITALIZE(LOAN:DESCRIPTION)
 COL=035 LOAN:ID
 COL=045 LOAN:TYPE
 COL=060 LOAN:OPENDATE
 COL=075 LOAN:LASTPAYMENTDATE
 COL=090 LOAN:DUEDATE
 SUPPRESS LOAN:BALANCE TOTAL="Total Loan Balance:"
 SUPPRESS LOAN:PASTDUEAMOUNT TOTAL="Total Amount Past Due:"
 NEWLINE
END
```

Suggested Solution for<initials>.LOAN.PORT1

```
[This specfile lists all open loans and indicates the status of the loan:
Current or Delinquent. It also prints the current due date, the last
payment date, and the loan balance.]

TARGET=LOAN

SELECT
 LOAN:CLOSEDATE='--/--/--'
END

PRINT TITLE="Loan Portfolio - ED"
 HEADERS
  COL=001       "Account#"
  COL=013       "Member's Name"
  COL=035 RIGHT "Type"
  COL=038       "Status"
  COL=050       "Due Date"
  COL=060       "Last Payment"
  COL=085 RIGHT "Balance"
  NEWLINE
  COL=001 REPEATCHR("-",85)
 END
 COL=001 ACCOUNT:NUMBER
 COL=013 NAME:SHORTNAME
 COL=035 LOAN:TYPE
 IF LOAN:DUEDATE<=(SYSTEMDATE-10) AND LOAN:PASTDUEAMOUNT>$0.00 THEN
  COL=038 "Delinquent"
 ELSE
  COL=038 "Current"
 COL=050 LEFT LOAN:DUEDATE
 COL=062 LEFT LOAN:LASTPAYMENTDATE
 COL=085 LOAN:BALANCE
END
```

Suggested Solution for <initials>.LOAN.PORT2

```
[This specfile lists all open loans and indicates the status of the loan:
Current or Delinquent. It also prints the current due date, the last
payment date, and the loan balance. A loan is considered delinquent if the
due date is on or before the date entered by the user and it has a Past Due
Amount greater than $0.00.]

TARGET=LOAN

DEFINE
 STARTDATE=DATE
END

SETUP
 STARTDATE=DATEREAD("Date on which loans become DQ")
END

SELECT
 LOAN:CLOSEDATE='--/--/--'
END

PRINT TITLE="Loan Portfolio using Date - ED"
 HEADERS
  COL=001       "Account#"
  COL=013       "Member's Name"
  COL=035 RIGHT      "Type"
  COL=038       "Status"
  COL=050       "Due Date"
  COL=060       "Last Payment"
  COL=085 RIGHT "Balance"
  NEWLINE
  COL=001 REPEATCHR("-",85)
 END
 COL=001 ACCOUNT:NUMBER
 COL=013 NAME:SHORTNAME
 COL=035 LOAN:TYPE
 IF LOAN:DUEDATE<=STARTDATE AND LOAN:PASTDUEAMOUNT>$0.00 THEN
  COL=038 "Delinquent"
 ELSE
  COL=038 "Current"
 COL=050 LEFT LOAN:DUEDATE
 COL=062 LEFT LOAN:LASTPAYMENTDATE
 COL=085 LOAN:BALANCE
END
```

Suggested Solution for <initials>.LOAN.PORT3

```
[This specfile lists all open loans and indicates the status of the loan:
Current or Delinquent. It also prints the current due date, the last
payment date and the loan balance. A loan is considered delinquent if the due
date is on or before the date entered by the user and it has a Past Due
Amount greater than $0.00. The amount entered by the user is formatted into
the report title.]

TARGET=LOAN

DEFINE
 STARTDATE=DATE
END

SETUP
 STARTDATE=DATEREAD("Date on which loans become DQ")
END

SELECT
 LOAN:CLOSEDATE='--/--/--'
END

PRINT TITLE="Loan Portfolio (DQ as of " +
  FORMAT("99/99/9999",STARTDATE) + ")-ED"
 HEADERS
  COL=001       "Account#"
  COL=013       "Member's Name"
  COL=035 RIGHT "Type"
  COL=038       "Status"
  COL=050       "Due Date"
  COL=060       "Last Payment"
  COL=085 RIGHT "Balance"
  NEWLINE
  COL=001 REPEATCHR("-"850)
 END
 COL=001 ACCOUNT:NUMBER
 COL=013 NAME:SHORTNAME
 COL=035 LOAN:TYPE
 IF LOAN:DUEDATE<=STARTDATE AND LOAN:PASTDUEAMOUNT>$0.00 THEN
  COL=038 "Delinquent"
 ELSE
  COL=038 "Current"
 COL=050 LEFT LOAN:DUEDATE
 COL=062 LEFT LOAN:LASTPAYMENTDATE
 COL=085 LOAN:BALANCE
END
```

Suggested Solution for <initials>.BOOLEAN

```
[Select open loans with a balance greater than zero, an original amount
greater than $500.00 or credit limit greater than $500.00, a Loan Type
within a range entered by the user.]

TARGET=LOAN

DEFINE
 STARTDATE=DATE
 STARTRANGE=NUMBER
 ENDRANGE=NUMBER
END

SETUP
 STARTRANGE=CODEREAD("Lowest  Loan Type")
 ENDRANGE=CODEREAD("Highest Loan Type")
 STARTDATE=DATEREAD("Date on which to consider Loan DQ")
END

SELECT
 LOAN:CLOSEDATE='--/--/--' AND
 LOAN:BALANCE>$0.00 AND
 (LOAN:ORIGINALBALANCE>$500.00 OR
  LOAN:CREDITLIMIT>$500.00) AND
 (LOAN:TYPE>=STARTRANGE AND LOAN:TYPE<=ENDRANGE)
END

PRINT TITLE="Loan Portfolio (DQ as of "+
            FORMAT("99/99/9999",STARTDATE)+")-ED"
 HEADERS
  COL=001 "Loan Types Selected: "
  PRINT   FORMAT(“9999”,STARTRANGE)
  PRINT   " to "
  PRINT   FORMAT(“9999”,ENDRANGE)
  NEWLINE NEWLINE
  COL=001       "Account#"
  COL=013       "Member's Name"
  COL=035 RIGHT "Type"
  COL=038       "Status"
  COL=051       "Due Date"
  COL=063       "Last Payment"
  COL=088 RIGHT "Balance"
  NEWLINE
  COL=001 REPEATCHR("-",88)
 END
 COL=001 ACCOUNT:NUMBER
 COL=013 NAME:SHORTNAME
 COL=035 LOAN:TYPE

IF LOAN:DUEDATE<=STARTDATE AND LOAN:PASTDUEAMOUNT>$0.00 THEN
  COL=038 "Delinquent"
 ELSE
  COL=038 "Current"
 COL=051 LEFT FORMAT("99/99/9999",LOAN:DUEDATE)
 COL=064 LEFT FORMAT("99/99/9999",LOAN:LASTPAYMENTDATE)
 COL=088 LOAN:BALANCE
END
```

Suggested Solution for <initials>.HOWOLD.DAYS

```
[Determine the age in days of the primary member.]

TARGET=ACCOUNT

DEFINE
 AGEINDAYS=NUMBER
END

PRINT TITLE="How Old"
 NEWLINE
 NEWLINE
 PRINT "Today's Date: "
 PRINT FORMAT("99/99/9999",SYSTEMDATE)
 NEWLINE
 NEWLINE
 PRINT "Member's Name: "
 PRINT NAME:LONGNAME
 NEWLINE
 NEWLINE
 PRINT "Member's Birthdate: "+FORMAT("99/99/9999",NAME:BIRTHDATE)
 NEWLINE
 NEWLINE
 AGEINDAYS=SYSTEMDATE-NAME:BIRTHDATE
 PRINT "Member's Age in Days: "
 PRINT AGEINDAYS
END
```

Suggested Solution for <initials>.HOWOLD.YEARS

```
[Determine the age in whole years of the primary member.]

TARGET=ACCOUNT

DEFINE
 AGEINDAYS=NUMBER
 AGEINYEARS=NUMBER
END

PRINT TITLE="How Old"
 PRINT "Today's Date: "
 PRINT FORMAT("99/99/9999",SYSTEMDATE)
 NEWLINE
 NEWLINE
 PRINT "Member's Name: "
 PRINT NAME:LONGNAME
 NEWLINE
 NEWLINE
 PRINT "Member's Birthdate: "+FORMAT("99/99/9999",NAME:BIRTHDATE)
 NEWLINE
 NEWLINE
 AGEINDAYS=SYSTEMDATE-NAME:BIRTHDATE
 AGEINYEARS=NUMBER(AGEINDAYS/365.25)
 PRINT "This member is "
 PRINT AGEINYEARS
 PRINT " years old"
END
```

Suggested Solution for <initials>.HOWOLD.YEARS2

```
[Determine the age in whole years of the primary member or of
 another person based on the birthdate entered.]

TARGET=ACCOUNT

DEFINE
 AGEINDAYS=NUMBER
 AGEINYEARS=NUMBER
 DOB=DATE
 YN=CHARACTER(1)
END

SETUP
 YN=ENTERYESNO("Calculate primary member's age?","Y")
 IF YN="N" THEN
  DOB=ENTERDATE("Date for calculation",'--/--/--')
END

PRINT TITLE="How Old"
 PRINT "Today's Date: "
 PRINT FORMAT("99/99/9999",SYSTEMDATE)
 NEWLINE
 NEWLINE
 IF YN="Y" THEN
  DO
   PRINT "Member's Name: "
   PRINT NAME:LONGNAME
   NEWLINE
   NEWLINE
   PRINT "Member's Birthdate: "+FORMAT("99/99/9999",NAME:BIRTHDATE)
   NEWLINE
   NEWLINE
   AGEINDAYS=SYSTEMDATE-NAME:BIRTHDATE
   PRINT "This member is "
  END
 ELSE
  DO
   PRINT "Birthdate Entered: "+FORMAT("99/99/9999",DOB)
   NEWLINE
   NEWLINE
   AGEINDAYS=SYSTEMDATE-DOB
   PRINT "This person is "
  END
 AGEINYEARS=NUMBER(AGEINDAYS/365.25)
 PRINT AGEINYEARS
 PRINT " years old."
END
```

Suggested Solution for <initials>.DATES

```
[Determine the date on which an individual will reach an age.]
```

```
TARGET=ACCOUNT

DEFINE
 AGE=NUMBER
 DOB=DATE
END

SETUP
 AGE=ENTERNUMBER("Enter age to calculate",000)
 DOB=ENTERDATE("Enter birthdate",NAME:BIRTHDATE)
END

PRINT TITLE="When Will I Be Old?"
 NEWLINE
 PRINT "Birthdate: "
 PRINT FORMAT("99/99/9999",DOB)
 NEWLINE
 NEWLINE
 PRINT "This person turns "
 PRINT AGE
 PRINT " on "
 PRINT FORMAT("99/99/9999",DATEOFFSET(DOB,AGE*12,0))
 PRINT "."
END
```

Suggested Solution for <initials>.FOREACH

```
[Lists all open shares with a balance greater than zero, followed by an
aggregate share balance for each member account.]

TARGET=ACCOUNT

DEFINE
 AGGSHBAL=MONEY
END

SELECT
 SHARE:CLOSEDATE='--/--/--' AND
 SHARE:BALANCE>$0.00
END

SORT
 NAME:SHORTNAME
 SHARE:TYPE
END

PRINT TITLE="Share Balance by Share Type - ED"
 HEADERS
  COL=001   "Name"
  COL=020   "Account#"
  COL=040 RIGHT "Type"
  COL=045  "ID"
  COL=070 RIGHT "Balance"
  NEWLINE
  COL=001  REPEATCHR("-",70)
 END

AGGSHBAL=$0.00

 COL=001 NAME:SHORTNAME
 COL=020 ACCOUNT:NUMBER
 FOR EACH SHARE WITH (SHARE:CLOSEDATE='--/--/--' AND
                      SHARE:BALANCE>$0.00
  DO
   COL=040 SHARE:TYPE
   COL=045 SHARE:ID
   COL=070 SHARE:BALANCE
   NEWLINE
   AGGSHBAL=AGGSHBAL+SHARE:BALANCE
  END
 COL=047 REPEATCHR("-",10)
 NEWLINE
 COL=001 "Aggregate Share Balance"
 COL=056 AGGSHBAL
 NEWLINE
 NEWLINE
END
```

Suggested Solution for <initials>.TOTAL.DIV

```
[Same as ED.BOOLEAN, except that we now calculate a total balance of DQ
loans and a total balance of all loans in the PRINT division. In the
TOTAL division, we print out the totals, then divide the DQ loan balance
by the total loan balance to determine the percentage of the loan balance
that's DQ.]

TARGET=LOAN

DEFINE
 STARTDATE=DATE
 STARTRANGE=NUMBER
 ENDRANGE=NUMBER
 TOTALLOANBAL=MONEY
 DQLOANBAL=MONEY
END

SETUP
 STARTRANGE=NUMBERREAD("Lowest  Loan Type")
 ENDRANGE=NUMBERREAD("Highest Loan Type")
 STARTDATE=DATEREAD("Date on which to consider Loan DQ")
END

SELECT
 LOAN:CLOSEDATE='--/--/--' AND
 LOAN:BALANCE>$0.00 AND
 (LOAN:ORIGINALBALANCE>$500.00 OR
  LOAN:CREDITLIMIT>$500.00) AND
 (LOAN:TYPE>=STARTRANGE AND LOAN:TYPE<=ENDRANGE)
END

PRINT TITLE="Loan Portfolio (DQ as of "+
            FORMAT("99/99/9999",STARTDATE)+")-ED"
 HEADERS
  COL=001 "Loan Types Selected: "
  PRINT   STARTRANGE
  PRINT   " to "
  PRINT   ENDRANGE
  NEWLINE NEWLINE
  COL=001       "Account#"
  COL=013       "Member's Name"
  COL=040 RIGHT "Type"
  COL=045       "Status"
  COL=060 RIGHT "Due Date"
  COL=075 RIGHT "Last Payment"
  COL=100 RIGHT "Balance"
  NEWLINE
  COL=001 REPEATCHR("-",100)
 END

COL=001 ACCOUNT:NUMBER
 COL=013 NAME:SHORTNAME
 COL=040 LOAN:TYPE
 IF LOAN:DUEDATE<=STARTDATE AND LOAN:PASTDUEAMOUNT>$0.00 THEN
  DO
   COL=045 "Delinquent"
   DQLOANBAL=DQLOANBAL+LOAN:BALANCE
  END
 ELSE
   COL=045 "Current"
 COL=060 LEFT FORMAT("99/99/9999",LOAN:DUEDATE)
 COL=075 LEFT FORMAT("99/99/9999",LOAN:LASTPAYMENTDATE)
 COL=100 LOAN:BALANCE
 TOTALLOANBAL=TOTALLOANBAL+LOAN:BALANCE
```

```
END

TOTAL
 NEWLINE
 COL=001 REPEATCHR("-",100)
 NEWLINE
 NEWLINE
 COL=001 "Total Loan Balance:"
 COL=100 TOTALLOANBAL
 NEWLINE
 COL=001 "Total Loan Balance, Delinquent Loans:"
 COL=100 DQLOANBAL
 NEWLINE
 COL=001 "DQ Loan Ratio:"
 COL=100 DQLOANBAL/TOTALLOANBAL
END
```

Suggested Solution for <initials>.REVIEW2

```
TARGET=SHARE

DEFINE
 GENHOLDAMT=MONEY
 CHECKHOLDAMT=MONEY
END

SELECT
 NOT ACCOUNT:TYPE=20 AND
[ANYWARNING(ACCOUNT,05) AND]
 SHARE:CLOSEDATE='--/--/--' AND
 (SHARE HOLD:EXPIRATIONDATE>=SYSTEMDATE OR
  SHARE HOLD:EXPIRATIONDATE='--/--/--') AND
 (SHARE HOLD:TYPE=0 OR SHARE HOLD:TYPE=1)
END

PRINT TITLE="Share Holds on New Member Accounts - ED"
 HEADERS
  COL=001       "Account#"
  COL=015       "ID"
  COL=025       "Description"
  COL=070 RIGHT "Balance"
  COL=075       "Hold Type"
  COL=100 RIGHT "Hold Amount"
  NEWLINE
  COL=001 REPEATCHR("-",90)
  NEWLINE
 END
 COL=001 ACCOUNT:NUMBER
 COL=015 SHARE:ID
 COL=025 CAPITALIZE(SHARE:DESCRIPTION)
 COL=070 SHARE:BALANCE
 FOR EACH SHARE HOLD WITH (SHARE HOLD:EXPIRATIONDATE>=SYSTEMDATE OR
                           SHARE HOLD:EXPIRATIONDATE='--/--/--') AND
                          (SHARE HOLD:TYPE=0 OR SHARE HOLD:TYPE=1)
  DO
   IF SHARE HOLD:TYPE=0 THEN
    DO
     COL=075 "General"
     GENHOLDAMT=GENHOLDAMT+SHARE HOLD:AMOUNT
    END
   ELSE
    DO
     COL=075 "Check"
     CHECKHOLDAMT=CHECKHOLDAMT+SHARE HOLD:AMOUNT
    END


(continued)

COL=100 SHARE HOLD:AMOUNT
   NEWLINE
  END
```

```
END
TOTAL
 PRINT REPEATCHR("-",100)
 NEWLINE NEWLINE
 PRINT "Total General Hold Amount:"
 COL=100 GENHOLDAMT
 NEWLINE NEWLINE
 PRINT "Total Check Hold Amount:"
 COL=100 CHECKHOLDAMT
 NEWLINE NEWLINE
 PRINT "Total Hold Amount:"
 COL=100 GENHOLDAMT+CHECKHOLDAMT
 NEWLINE
END
```

Suggested Solution for <initials>.YTDDIVS

```
WINDOWS

TARGET=ACCOUNT

DEFINE
 QUOTE=CHARACTER(1)
 DIVTOT=MONEY
 TRUE=1
 FALSE=0
END

SETUP
 QUOTE=CTRLCHR(34)
END

PRINT TITLE="WCM.YTDDIV.HTML"
 HTMLVIEWOPEN

 HTMLVIEWLINE("<html>")
 HTMLVIEWLINE("    <head>")
 HTMLVIEWLINE("    <title>")
 HTMLVIEWLINE("        YTD Dividend Information")
 HTMLVIEWLINE("    </title>")
 HTMLVIEWLINE("    </head>")
 HTMLVIEWLINE("    <body bgcolor=00ccff text=003366>")
 HTMLVIEWLINE("        <center>")
 HTMLVIEWLINE("            <h1>")
 HTMLVIEWLINE("                YTD Dividend Information")
 HTMLVIEWLINE("            </h1>")

 HTMLVIEWLINE("            <table width=60%>")
 HTMLVIEWLINE("                <tr>")
 HTMLVIEWLINE("                    <td colspan=2>")
 HTMLVIEWLINE("                        <h3>")
 HTMLVIEWLINE("                            Taxable Dividends")
 HTMLVIEWLINE("                        </h3>")
 HTMLVIEWLINE("                    </td>")
 HTMLVIEWLINE("                </tr>")

 FOR EACH SHARE WITH SHARE:IRSCODE=0 AND
                    (SHARE:CLOSEDATE='--/--/--' OR
                     SHARE:CLOSEDATE>=DATE(1,1,FULLYEAR(SYSTEMDATE)))
  DO
   HTMLVIEWLINE("              <tr>")
   HTMLVIEWLINE("                  <td>")
   HTMLVIEWLINE("                      Share "+SHARE:ID)
   HTMLVIEWLINE("                  </td>")
   HTMLVIEWLINE("                  <td align=right>")
   HTMLVIEWLINE(                   FORMAT("$###,###,##9.99",SHARE:DIVYTD))
   HTMLVIEWLINE("                  </td>")
   HTMLVIEWLINE("              </tr>")
   DIVTOT=DIVTOT+SHARE:DIVYTD
  END

 HTMLVIEWLINE("                <tr>")
 HTMLVIEWLINE("                    <td>")
 HTMLVIEWLINE("                        &nbsp;")
 HTMLVIEWLINE("                    </td>")
 HTMLVIEWLINE("                    <td align=right>")
 HTMLVIEWLINE("                        <hr width=100% color=003366>")
 HTMLVIEWLINE("                    </td>")
 HTMLVIEWLINE("                </tr>")
 HTMLVIEWLINE("                <tr>")
 HTMLVIEWLINE("                    <td>")
 HTMLVIEWLINE("                        <b>")
 HTMLVIEWLINE("                            Total")
 HTMLVIEWLINE("                        </b>")
 HTMLVIEWLINE("                    </td>")
 HTMLVIEWLINE("                    <td align=right>")
 HTMLVIEWLINE("                        <b>")
 HTMLVIEWLINE(                          FORMAT("$###,###,##9.99",DIVTOT))
 HTMLVIEWLINE("                        </b>")
 HTMLVIEWLINE("                    </td>")
 HTMLVIEWLINE("                </tr>")
 HTMLVIEWLINE("            </table>")

 HTMLVIEWLINE("            <br />")
 HTMLVIEWLINE("            <br />")
 HTMLVIEWLINE("            <br />")

 DIVTOT=$0.00

 HTMLVIEWLINE("            <table width=60%>")
 HTMLVIEWLINE("                <tr>")
 HTMLVIEWLINE("                    <td colspan=2>")
 HTMLVIEWLINE("                        <h3>")
 HTMLVIEWLINE("                            Non-Taxable Dividends")
 HTMLVIEWLINE("                        </h3>")
 HTMLVIEWLINE("                    </td>")
 HTMLVIEWLINE("                </tr>")

 FOR EACH SHARE WITH SHARE:IRSCODE>0 AND
                     (SHARE:CLOSEDATE='--/--/--' OR
                      SHARE:CLOSEDATE>=DATE(1,1,FULLYEAR(SYSTEMDATE)))
  DO
   HTMLVIEWLINE("              <tr>")
   HTMLVIEWLINE("                  <td>")
   HTMLVIEWLINE("                      Share "+SHARE:ID)
   HTMLVIEWLINE("                  </td>")
   HTMLVIEWLINE("                  <td align=right>")
   HTMLVIEWLINE(                   FORMAT("$###,###,##9.99",SHARE:DIVYTD))
   HTMLVIEWLINE("                  </td>")
   HTMLVIEWLINE("              </tr>")
   DIVTOT=DIVTOT+SHARE:DIVYTD
  END
 HTMLVIEWLINE("                <tr>")
 HTMLVIEWLINE("                    <td>")
 HTMLVIEWLINE("                        &nbsp;")
 HTMLVIEWLINE("                    </td>")
 HTMLVIEWLINE("                    <td align=right>")
 HTMLVIEWLINE("                        <hr width=100% color=003366>")
 HTMLVIEWLINE("                    </td>")
 HTMLVIEWLINE("                </tr>")
 HTMLVIEWLINE("                <tr>")
 HTMLVIEWLINE("                    <td>")
 HTMLVIEWLINE("                        <b>")
 HTMLVIEWLINE("                            Total")
 HTMLVIEWLINE("                        </b>")
 HTMLVIEWLINE("                    </td>")
 HTMLVIEWLINE("                    <td align=right>")
 HTMLVIEWLINE("                        <b>")
 HTMLVIEWLINE(                          FORMAT("$###,###,##9.99",DIVTOT))
 HTMLVIEWLINE("                        </b>")
 HTMLVIEWLINE("                    </td>")
 HTMLVIEWLINE("                </tr>")
 HTMLVIEWLINE("            </table>")
```

```
 HTMLVIEWLINE("    </body>")
 HTMLVIEWLINE("</html>")

 HTMLVIEWDISPLAY
END
```