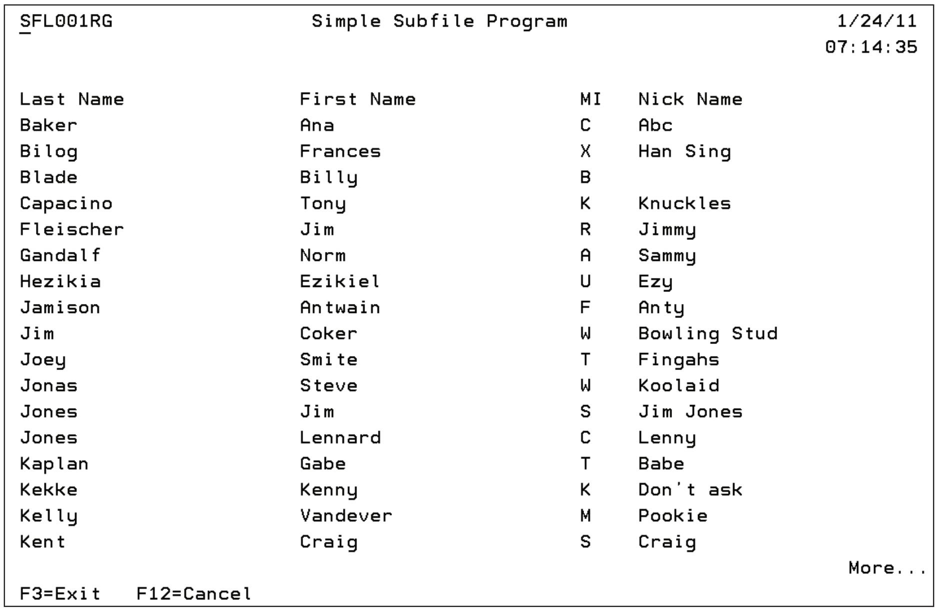
CHAPTER 1

The Specifics on Subfiles

According to IBM, a subfile is a group of records that have the same record format and are read from and written to a display station in one operation. As this definition suggests, a subfile is not a file; rather, it is a temporary place to store data of the same format to be used by a display file program.

Why Use Subfiles?

Figure 1.1 shows the typical use of a subfile. As you can see, subfiles are useful when you want to list multiple, like records on a display. A major benefit of subfiles is that you can define them so the number of displayed records exceeds the number of lines available on the screen, allowing the user to scroll, or page, through the data. This is usually the reason you would decide to use subfiles in your display program. Because of their ease of use, however, I deploy subfiles even when I think I will never display more than one screen, because they are easy to change if the data someday requires more than one screen.



```
SFL001RG               Simple Subfile Program                       1/24/11
                                                                   07:14:35

Last Name               First Name               MI   Nick Name
Baker                   Ana                      C    Abc
Bilog                   Frances                  X    Han Sing
Blade                   Billy                    B
Capacino                Tony                     K    Knuckles
Fleischer               Jim                      R    Jimmy
Gandalf                 Norm                     A    Sammy
Hezikia                 Ezikiel                  U    Ezy
Jamison                 Antwain                  F    Anty
Jim                     Coker                    W    Bowling Stud
Joey                    Smite                    T    Fingahs
Jonas                   Steve                    W    Koolaid
Jones                   Jim                      S    Jim Jones
Jones                   Lennard                  C    Lenny
Kaplan                  Gabe                     T    Babe
Kekke                   Kenny                    K    Don't ask
Kelly                   Vandever                 M    Pookie
Kent                    Craig                    S    Craig
                                                                    More...
F3=Exit   F12=Cancel
```

Figure 1.1: A typical subfile application lists records from a file and allows the user to page through them.

In addition to simply displaying multiple lines of data, you can use subfiles and their multiple-line capabilities to add, change, and delete records very effectively. You can also use subfiles in a non-display manner, to create self-extending data structures and arrays in your RPG program. They work like multiple-occurrence data structures and arrays, except that instead of having to hard-code a number of elements large enough to hold the maximum number of entries, you can use subfiles. You start with a small number of entries and allow the number to expand dynamically, as your data expands.

Why Should You Care?

Haven't programmers been able to display data to a screen for years, without subfiles? Yes, but subfiles allow you to display lists of like data that can extend beyond one screen. They make it easier for you to create display applications, which, in turn, makes you more productive. Subfile programs are easy to write and maintain because much of the work is done for you in the Data Definition Specifications (DDS). Most of the time, you can change the characteristics of a subfile program without having to modify the RPG code driving it.

If that isn't enough, keep in mind that the ability to code subfiles is often the measure of an RPG programmer's worth. As an added benefit, when you're talking with a group of programmers at a users' group or technical conference, you can hold your head high, knowing that you're right up there with the rest of the subfile-savvy programmers.

Two DDS Formats Are Better Than One

As I stated earlier, most of the work in a subfile application is accomplished in the DDS. For every subfile you describe in your DDS, you're required to use two format types: a subfile record format (SFL) and a subfile control record format (SFLCTL). The subfile record format is used much as a record format is for a physical file. It defines the fields contained in a row of a subfile, describes the definition and attributes of those fields, and holds the actual data.

Unlike a physical file, however, a subfile record format is used only in memory, and only for the duration of the job. Once the program using the subfile ends, the data in that subfile is gone. Individual subfile records are read from, written to, and updated to the subfile by accessing the subfile record format in your program. The SFL keyword is required in your DDS to define a subfile record, just as the RECORD keyword is used for a typical display record format.

The subfile control record format describes the heading information for the subfile and controls how the subfile is going to perform. It's here that you define the size of the subfile, as well as the conditions under which the subfile will be cleared, initialized, displayed, and ended. The subfile control record format is unique because it controls aspects of a subfile in a way that other files aren't controlled. For example, you control the size of a physical file during compilation or by using the Change Physical File (CHGPF) command. You do not determine it in the DDS. Your program will operate the subfile control record format directly when it performs functions to the whole subfile, not individual records. Actions such as initializing, clearing, and displaying are accomplished in this fashion.

Four keywords are required in the subfile control record format:

1. The subfile control (SFLCTL) keyword identifies the subfile control record format, much as the record (RECORD) keyword identifies a typical display record format. The SFLCTL keyword also identifies the subfile record format that must immediately precede it.
2. The subfile size (SFLSIZ) keyword specifies the initial number of records the subfile may contain.
3. The subfile page (SFLPAG) keyword specifies the number of records that one screen of data may contain.
4. The subfile display (SFLDSP) keyword specifies when the subfile is displayed.

Note that the first requirement states that the SFLCTL keyword identifies the subfile record format that precedes it. Therefore, the SFL record format must immediately precede the SFLCTL format. The individual keywords available in each format can be placed in any order, but they must exist at the top of the format, before any constant or field definitions.

DDS and RPG Working as One

Once you code and compile the DDS for a subfile, you are ready to use that subfile in your RPG program. Your first three actions will be initializing, loading, and displaying your subfile. There are other things you can, and will, do to a subfile, but let's start with that foundation, and build up from there.

Before loading a subfile, you might want to clear or initialize it for use. Clearing or initializing takes effect when you write to the subfile control record format with the appropriate conditioning indicator set on, using the RPG WRITE operation. This isn't a requirement for the introductory load in your program, but as you create more complex subfile applications, you'll need to know the difference between clearing and initializing a subfile.

Let's return to the DDS for a moment. To clear a subfile, use the subfile clear (SFLCLR) keyword. This removes all records from the subfile. However, it does not remove them from the display until the next time the subfile is written to the screen by your program. This is different from the subfile initialize (SFLINZ) keyword, which will set all the records in your subfile to their default value. If you have no default value set in your DDS, numeric fields will default to zeros and character fields will default to blanks. Suppose your subfile size keyword is set to 100. If you clear it, instead of having an empty subfile, you will have 100 records, each initialized with its default value. I will talk more about SFLINZ later on, so don't worry if you don't quite get it yet. In short, the decision about whether to use SFLCLR or SFLINZ is determined by the role of your program.

You can load your subfile in a few different ways. The most common way is to retrieve data from one or more data files and write that data to the subfile record format, using the RPG WRITE operation. You might also choose to write only enough records to fit on one page (as determined by SFLPAG), and write more only if the user requests more; write all the desired records to the subfile before displaying the screen to the user; or use a combination of both. Still another way to load your subfile is to initialize the subfile first using the SFLINZ keyword, display the subfile with its default values to the screen, and allow the user to type data into the subfile from the screen.

As we go along in this book, I will show you examples of all these techniques because they are coded quite differently. I will also explain when to use which method.

Once your subfile is loaded or initialized, you're ready to display it. Depending on how you created your DDS, your subfile can take on many looks. No matter which form it takes, however, you only need one line of RPG code to display it.

Let's look at the code for a basic subfile program that displays a list of names. These could be the names of customers, salespeople, or employees; for our purpose, we'll just use generic names. Take notice of how much work is done with very little RPG code. Be careful, though. Once you understand this code, there's no turning back—you're on your way to becoming a subfile programmer.

Oh Goody! Some Code

The complete source for this example at is available at *http://www.mc-store.com/5104.html*, including compile information for a physical file (SFL001PF), a logical file (SFL001LF), a display file (SFL001DF), and the RPG (SFL001RG). To make this example work, you need to compile the three source members in order: first SFL001PF, then SFL001DF, and finally SFL001RG. In real life, it might not always matter whether you compile the data file before the display file, since the data file will probably already exist. In this example, however, I reference the physical file field information in my display file. As a result, the physical file needs to be created first. Figure 1.2 shows the code for the physical file.

```
A                                       UNIQUE
A          R PFR
A            DBIDNM         7  0
A            DBFNAM        20
A            DBLNAM        20
A            DBMINI         1
A            DBNNAM        20
A            DBADD1        30
A            DBADD2        30
A            DBADD3        30
A          K DBIDNM
```

Figure 1.2: The DDS for the physical file SFL001PF.

After creating the physical file, you'll need some sort of tool to get data into it. One way is to create a subfile program to enter the data, but that's what you'll learn later in this book. For now, you can use the IBM i operating system's (OS's) native Data File Utility (DFU) or any third-party package your shop already owns. To start DFU, run the command UPDDTA FILE(SFL001PF), or take Option 18 next to SFL001PF if you're using the Work with Objects Using Programming Development Manager (PDM) view.

The DDS for the physical file is pretty self-explanatory, so I will leave it alone. I'm going to start with the DDS for the display file. This is really where most of the work related to subfiles is accomplished. As I mentioned in the introduction, I'm assuming you have some basic knowledge of DDS. With that, let's jump to the subfile record format, signified by a record name of SFL1 and the record-level keyword of SFL. SFL1 is where you define the fields contained in each individual subfile record. It is also the format that will be written to in your RPG program. This is the format that will actually contain the subfile data.

The example in Figure 1.3 describes four fields: DBLNAM, DBFNAM, DBMINI, and DBNNAM. These fields are referenced from the physical file SFL001PF. It's important to notice here that these fields are defined as output only (indicated by the "O" in position 38), and they are set to start on row 5. That means the list of data to be displayed on the screen will start on row 5.

Each field has its own starting column number. DBLNAM starts in column 2, DBFNAM starts in column 26, and so on. The column settings let you place the fields across the row in whatever fashion you want, just as you would with a printed report. That's about it for the subfile record format.

```
A                                      DSPSIZ(24 80 *DS3)
A                                      PRINT
A                                      ERRSFL
A                                      CA03
A                                      CA12
A*
A          R SFL1                      SFL
A*
A            DBLNAM    R        O  5  2REFFLD(PFR/DBLNAM*LIBL/SFL001PF)
A            DBFNAM    R        O  5 26REFFLD(PFR/DBFNAM *LIBL/SFL001PF)
A            DBMINI    R        O  5 50REFFLD(PFR/DBMINI *LIBL/SFL001PF)
A            DBNNAM    R        O  5 55REFFLD(PFR/DBNNAM *LIBL/SFL001PF)
```

Figure 1.3: The DDS for the subfile record format SFL1 in SFL001DF.

You have defined your field's size and usage, and you have designed the layout of each field. The next record format is the subfile control record format, signified by the record name SF1CTL and the keyword SFLCTL. The subfile control format must come directly after the subfile record format. It's always distinguished by the SFLCTL keyword. Even though it has to come directly after the SFL format, you still must place the name of the subfile record format, in this case SFL1, within parentheses next to the SFLCTL keyword.

The DDS for SF1CTL is shown in Figure 1.4. The subfile record format is where you define everything about the individual record, while the subfile control record format is where you define everything about the subfile as a whole. You have two steps to take in the subfile control record format:

1. Define the subfile characteristics with a series of keywords.
2. Define any fields that will exist in the heading of the subfile.

The fields usually include column headings for the subfile records, as well as a title, program name, and date and time. However, they can also include input-capable fields, discussed in the next chapter. It's important to note that fields defined in the subfile control format are not part of the scrollable, subfile records.

```
A          R SF1CTL                   SFLCTL(SFL1)
A*
A                                     SFLSIZ(0500)
A                                     SFLPAG(0017)
A                                     OVERLAY
A N32                                 SFLDSP
A N31                                 SFLDSPCTL
A  31                                 SFLCLR
A  90                                 SFLEND(*MORE)
A            RRN1          4S 0H      SFLRCDNBR
A                                 4  2'Last Name'
A                                     DSPATR(HI)
A                                 4 26'First Name'
A                                     DSPATR(HI)
A                                 4 50'MI'
A                                     DSPATR(HI)
A                                 4 55'Nick Name'
A                                     DSPATR(HI)
A                                 1  2'SFL001RG'
A                                 1 27'Simple Subfile Program'
A                                     DSPATR(HI)
A                                 1 71DATE
A                                     EDTCDE(Y)
A                                 2 71TIME
```

Figure 1.4: The DDS for the subfile control record format SF1CTL in SFL001DF.

There is no requirement for the content of the fields in the subfile control record format, but there is a limit as to where they can be placed. Remember that the SFL record format started on row 5. That means that fields defined in the SFLCTL record format must not be placed anywhere below row 4. Now, looking at the example, let's examine the keywords used. They can be placed in any order within the SFLCTL record format, but they must also be entered before any fields or constants are defined.

The first thing I do in Figure 1.4 is define the subfile size (SFLSIZ) and subfile page size (SFLPAG). The SFLSIZ is set to 500. That means the initial number of records my subfile can contain is 500. I can extend the number contained in the SFLSIZ keyword up to a maximum of 9,999 records, but I will cover that in the next chapter.

The SFLPAG keyword determines how many records can fit on one page. I set mine to 17, meaning that each screen of data will contain a maximum of 17 records. You're limited on this parameter only by the number of records that can be physically displayed on a screen.

The OVERLAY keyword tells the display file to display the screen on top of what is already displayed—overlaying anything that is already there, but not erasing it. The subfile display (SFLDSP) keyword displays the subfile on the screen. It's conditioned, in this case, by N32, which means it will display only when indicator 32 is set off. You're not required to condition the SFLDSP keyword. I condition it in this example (and in most of my subfile programs, for that matter) because I want to stop the subfile from displaying if there are no records in it. You'll get a runtime error if you try to display a subfile with no records in it. There are a number of ways to handle the no-record situation, but I choose to set on indicator 32. This stops the subfile, and subsequent errors, from being displayed.

The next keyword is the subfile display control (SFLDSPCTL). This keyword allows you to display the control record format. The conditioning indicator on the SFLDSPCTL keyword, N31, conditions when the SFLCTL record format is to be displayed. In this case, the SFLCTL record format will be displayed only when it's told to do so by the RPG program, and when indicator 31 is off. This keyword isn't required if there are no fields to display or function keys to control in the format. Many keywords change the characteristics of the subfile, but these changes aren't seen until either the SFLDSP or the SFLDSPCTL keyword is activated.

The next keyword is the subfile clear (SFLCLR), which clears the subfile of its entries. It provides you with an empty subfile just waiting to have records written to it. Notice that I used the opposite conditioning indicator for the SFLDSPCTL keyword. I did this because the same RPG operation (WRITE) is used to display the control record format and clear the subfile. I probably don't want to clear the subfile at the same time I want to display it, so I use the same indicator. When indicator 31 is off, I want to display the subfile. When it's on, I want to clear it.

The subfile end (SFLEND) keyword tells the user there are more records in the subfile. A conditioning indicator is required when you're using the SFLEND keyword. In this case, indicator 90 conditions the SFLEND keyword. The *MORE parameter is just one of the valid parameters used with the SFLEND keyword. (The others will be discussed later.) It will cause the screen to show the word "More . . ." at the bottom-right corner of the last subfile record on the page. When the subfile is on its last page, the word "Bottom" replaces "More . . ." to indicate that this is the end of the subfile. SFLEND specified by itself, with no parameter, will cause the screen to display a plus sign ("+") in the lower-right corner when there are further pages to be displayed. When the subfile is on its last page, the plus sign is replaced by a blank.

Now that all the keywords have been defined, you can define any fields you need for your subfile control record format. In Figure 1.4, I describe some basic column headings: a title, the date, and the time. Because this is the same technique you would use to define headings on any display record format, it does not warrant discussion here.

Notice one particular field defined in the SFLCTL record format: RRN1, which will be used as my subfile relative record number. I define it as a 4-digit signed numeric, with no decimal places. (Remember that there is a maximum of 9,999 records in a subfile.) The "H" in position 38 indicates that this is a hidden field. I'm not going to display this field anywhere on the screen, but it's extremely important because it will keep track of which subfile record I'm working with.

The last record format defined in this DDS is the function key line. On line 23 of the display, I will list the function keys available to the user. This is the reason I use the `OVERLAY` keyword in the `SFLCTL` record format. I will first write the `FKEY1` format, which will display on line 23, and then `OVERLAY` the `FKEY1` format, but not erase it.

I use a separate record format to display the function keys because I want my function keys to appear at the bottom of the screen, as is standard with most IBM i display screens. I cannot place the function key constants in the subfile record format because I don't want them repeated with each subfile record. I'm also restricted from placing the function key constants in the subfile control format because doing so would violate the rule of the subfile control format not overlapping the subfile record format. The only way I could get away with placing the function key constants in the subfile control format is if I wanted them to display at the top of the screen. The `FKEY1` format is shown in Figure 1.5.

```
A          R FKEY1

A*

A                                 23  2'F3=Exit'

A                                      COLOR(BLU)

A                                 23 12'F12=Cancel'

A                                      COLOR(BLU)
```

Figure 1.5: The DDS for function key record format FKEY1 in SFL001DF.

There, you have just learned how to build a basic subfile display file. This is more than a watered-down version appropriate only for learning. It is an example of a typical display file that you would actually use to display a list of data to a user. You've done the hard part, believe it or not. Now let's look at the RPG code necessary to load and display this subfile.

Figure 1.6 shows the file specifications (F specs). Notice that there are two files described. SFL001DF is the display file you just learned about. I define it as a combined (C in column 17), full procedural (F in column 18), externally described (E in column 22) workstation (WORKSTN) file. I further define SFL001DF on the next line of code by using the SFILE keyword with two parameters separated by a colon. The SFILE keyword defines a subfile contained in the display file. If you want to use that subfile in your program, you would code a SFILE line for every subfile record format contained in your DDS. The first parameter of the SFILE keyword is the name of the subfile record format, and the second is the field that will contain the subfile relative record number.

```
FSfl001df  cf   e             Workstn

F                                     Sfile(Sfl1:Rrn1)

FSfl001lf  if   e           k Disk
```

Figure 1.6: RPG F specs for program SFL001RG.

Remember that I defined the relative record number field (RRN1) as a hidden field in the SFLCTL record format. The important thing to note is that the hidden field defined in the subfile record format is the one placed in the second parameter of the SFILE keyword. I will use SFL001LF, the logical file built over SFL001PF, to load the subfile.

The body of the program consists only of a main routine and one subroutine. The main routine, shown in Figure 1.7, is pretty simple. I first execute the subfile build routine (Build_Subfile) and then code the DOU loop that will process the screen until either the F3 or F12 key has been pressed. I set indicator 90 on before displaying the screen, so that the end of the subfile will be indicated properly. Indicator 90 is also the indicator I conditioned the SFLEND keyword on. Therefore, when the screen that contains the last record in the subfile is displayed, the word “Bottom” will appear in the lower-right corner.

```
/Free


 // Main Routine



 Exsr Build_Subfile;   // Execute the subfile build routine.



 Dou  *Inkc or *Inkl;  // Process until F3 or F12 is pressed.

   *In90 = *On;        // Set Subfile end indicator.

   Write Fkey1;        // Display the function key line.

   Exfmt Sf1ctl;       // Display the subfile.

 Enddo;



 *Inlr = *On;
```

Figure 1.7: The mainline code for RPG program SFL001RG.

Let's examine the Build_Subfile routine first, since that code will be executed before the DOU loop. The subroutine Build_Subfile is shown in Figure 1.8.

```
 // *****************************************************************

 // Build_Subfile - Build the List

 // *****************************************************************



Begsr Build_Subfile;



 // Clear the Subfile



 Rrn1 = *Zero;       // Set the record counter to 0.

 *In31 = *On;        // Turn on the subfile clear indicator.

 Write Sf1ctl;       // Clear the subfile.

 *In31 = *Off;       // Turn off the subfile clear indicator.



 // Load data to subfile



 Setll (*Loval) Sfl001lf;

 Read Sfl001lf;
```

Continued

```
 Dow (Not %eof) And (Rrn1 <= 500); // Read while there are records in

                                   // the file, up to the 500 limit.

   Rrn1 = Rrn1 + 1;  // Increment subfile record number.

   Write Sfl1;       // Write the record to the subfile.

   Read Sfl001lf;    // Read next data record.

 Enddo;



 // If no records were loaded, do not display the subfile



 If Rrn1 = *Zero;    // If no records were loaded,

   *In32 = *On;      // set the indicator to NOT display the subfile.

 Else;               // Otherwise,

   Rrn1 = 1;         // set the rec to 1 so the first page is displayed.

 Endif;



 Endsr;



/End-Free
```

Figure 1.8: The subroutine used to build the subfile in SFL001RG.

The first block of code clears the subfile (SFLCLR in the DDS). Because it was conditioned on indicator 31, I set on indicator 31 and write to the SFLCTL record format, which performs the clear. (The SFLDSPCTL isn't activated in this case because it's conditioned on indicator 31 being off.) After I write the SFLCTL record format, I set indicator 31 off, so the next write to the format will cause it to be displayed.

Now it's time to load data to the subfile. I do this by setting the file pointer to the beginning of the file and reading the first record of the data file. The three lines contained within the DOW loop are sufficient to load my subfile. I'm going to process the DOW loop until there are no more records in the file, or until I hit the 500-record limit set on the SFLSIZ keyword. Each time I read a record, I increment the subfile relative record number (RRN1) and write the record to the subfile record format. Because my subfile field names are the same as the data file field names, no MOVE, MOVEL, EVAL, or Z-ADD statements are necessary.

The IF statement below the loop keeps an error from ending the program. If no records were read from the file (RRN1 = 0), I set indicator 32 on. Remember that the subfile display (SFLDSP) keyword is conditioned on 32 being off. If it's on, the subfile won't be displayed. Displaying a subfile with zero records will cause an error and end the program.

Back in the main routine (Figure 1.7), now that the subfile is loaded, I will display it. Within the DOU loop, I write the function key format (FKEY1). This will display the function keys available to the user on line 23 of the screen. I then display the subfile with the execute format (EXFMT) keyword.

When displaying a subfile, the EXFMT is always done on the SFLCTL record format. Because the OS knows which subfile record format is associated with which subfile control record format, there's no need to explicitly perform an EXFMT to the SFL record format. I set indicator 31 off after the subfile was cleared, and with this write to the SFLCTL record format, the subfile control record will be displayed. If there are records in your subfile, indicator 32 will be off. As a result, the subfile should also be displayed.

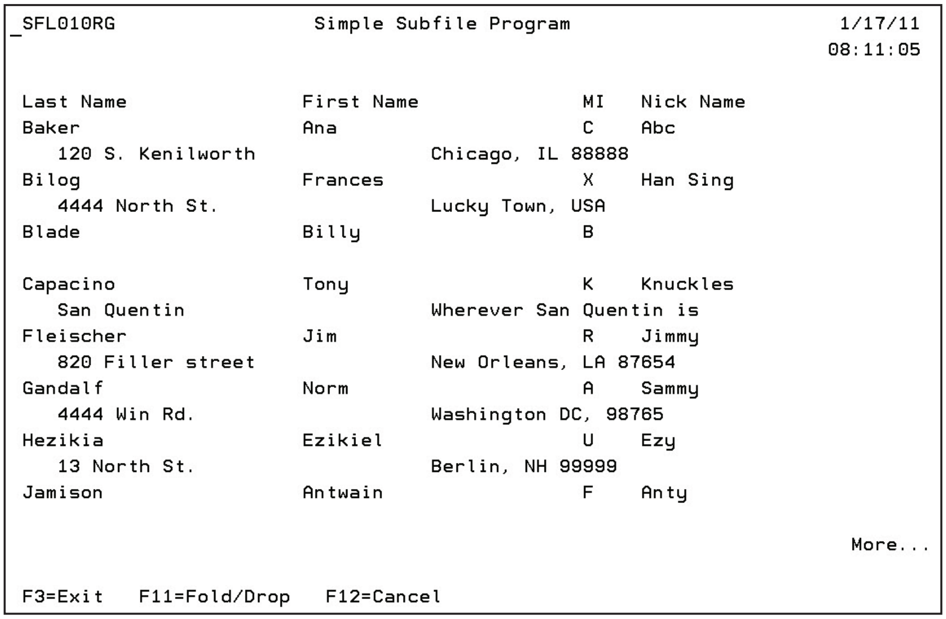
Depending on how many records you added to your data file, you would see either "More . . ." or "Bottom" in the lower-right corner of the screen. If you saw "More . . . ," you'd be able to press the Page Down key to display the next page of data. You could do this until you saw "Bottom." If there were more than one page of data, you'd be able to use the Page Up key to scroll back to the top.

What's nice about this is that I didn't have to code a single line of RPG to handle the scrolling. IBM i took care of it for me. The program will sit on the EXFMT operation until a key is pressed that returns control back to the RPG program. In this case, the Page Up and Page Down keys are handled by the system and won't return control back to the program.

This method of subfile programming is called the *load-all* method. Using this method, you load the subfile one time, and display it to the screen. IBM i handles the paging for you. As you can see, it doesn't take a lot of code to get a load-all subfile up and running. In the next chapter, you'll learn more about the load-all method as well as two other methods, and when to use which one.

Hip Hop Subfiles Can Wrap

There might be a time when one line of data doesn't cut it. The subfile in Figure 1.1 displays first name, last name, middle initial, and nickname neatly on one line. But what if you want to show part of the address on the screen? There isn't enough room on the subfile line to squeeze in the address. What do you do? Do you abandon the subfile approach and try something else? The answer, thankfully, is no. With just a little more code in the DDS and, optionally, a couple more lines in the RPG, your subfile can display multiple lines of data per subfile record. Figure 1.9 shows the output from previous subfile programs, with the addition of the address, city, and state on a second line.



```
_SFL010RG                Simple Subfile Program                          1/17/11
                                                                        08:11:05

 Last Name                First First Name            MI   Nick Name
```

```
_SFL010RG                Simple Subfile Program                          1/17/11
                                                                        08:11:05

 Last Name                First Name             MI   Nick Name
 Baker                    Ana                    C    Abc
    120 S. Kenilworth                Chicago, IL 88888
 Bilog                    Frances                X    Han Sing
    4444 North St.                   Lucky Town, USA
 Blade                    Billy                  B

 Capacino                 Tony                   K    Knuckles
    San Quentin                      Wherever San Quentin is
 Fleischer                Jim                    R    Jimmy
    820 Filler street                New Orleans, LA 87654
 Gandalf                  Norm                   A    Sammy
    4444 Win Rd.                     Washington DC, 98765
 Hezikia                  Ezikiel                U    Ezy
    13 North St.                     Berlin, NH 99999
 Jamison                  Antwain                F    Anty

                                                                          More...

 F3=Exit   F11=Fold/Drop   F12=Cancel
```

Figure 1.9: An example of a subfile with lines that wrap.

Just Add the Fields

Most of the work needed to create the multiple-line subfile record is accomplished in the DDS. In fact, the only reason you might decide to modify the RPG is to tighten up the technique a little. You could easily leave the RPG alone and still provide multiple-line subfile records.

Figure 1.10 shows a new version of the subfile record format that places two address fields on a second line. All I've done is add two new fields on a second line of the subfile. The subfile starts on line 5 with the name fields, and now extends to a second line (line 6) with the addition of DBADD1 and DBADD2. This will cause your subfile to display two lines per record. The system figures all this out for you, once you define your subfile record format and determine how many records you want to display on a page (SFLPAG) in your record control format. You can see the complete DDS (SFL010DF) at *http://www.mc-store.com/5104.html*.

```
A            DBLNAM    R        O  5  2REFFLD(PFR/DBLNAM *LIBL/SFL001PF)
A            DBFNAM    R        O  5 26REFFLD(PFR/DBFNAM *LIBL/SFL001PF)
A            DBMINI    R        O  5 50REFFLD(PFR/DBMINI *LIBL/SFL001PF)
A            DBNNAM    R        O  5 55REFFLD(PFR/DBNNAM *LIBL/SFL001PF)
A            DBADD1    R        O  6  5REFFLD(PFR/DBADD1 *LIBL/SFL001PF)
A            DBADD2    R        O  6 37REFFLD(PFR/DBADD2 *LIBL/SFL001PF)
```

Figure 1.10: The subfile record format for a multiple-line subfile record.

This brings me to my next point. There is one additional step you'll have to take to properly fit this new subfile on the screen. You need to change the SFLPAG keyword in the control record format. In the previous example, SFLPAG was set to 17. However, trying to fit 17 double-lined subfile records on a page will create some problems when you try to compile. To counter this, you'll have to modify the SFLPAG number to some appropriate number. Because I had 17 before and am now displaying two lines per record, I changed the SFLPAG keyword to 8. That means the subfile will display eight two-line subfile records per page, which (and I know you can do the math) equates to 16 actual lines.

Sflfold and Sfldrop

By simply adding the two fields to the subfile record format, changing the SFLPAG keyword to 8, and recompiling both the display file and the RPG program, your subfile will now look something like Figure 1.9. Cool, right? Well, what if you want to toggle back and forth between one- and two-line subfile records? Maybe you don't always want to see the address information. Maybe you'd like to display the subfile as in Figure 1.1 and press a function key to show the address lines (Figure 1.9 shows that hint).

Is this asking too much? Of course not. With the addition of a few subfile keywords, you can actually toggle between single-line and multiple-line subfile records.