4th Quarter Shell

by Lincoln Stoller, Ph.D.

Braided Matrix, Inc.
May 2000

For information concerning 4th Quarter products contact Braided Matrix, Inc.
Voice (914) 340-4206
Internet: info@4thquarter.com

CHAPTER 1

4Q™ Shell

4Q Shell provides the conventions and utilities needed to build a multi-user database that can later be enhanced by the addition of a full accounting engine.

Introduction

4Q Shell is a complex program that provides the tools for you to easily write complex applications in 4th Dimension. 4Q Shell provides adaptable templates, built from reusable 4D code. Applications you create using 4Q Shell are:

- ready to be integrated with 4th Quarter® Accounting Solution, Braided Matrix's high-end accounting engine.
- robust, multi-user applications with sophisticated features.

The framework laid out in 4Q Shell provides the tools and conventions that are the basis of 4th Quarter Accounting. These tools and conventions facilitate the programming of a basic database. They have little to do with accounting.

The only elements in the 4Q Shell that are specific to accounting are certain fields in the system default table. These fields are not used in the shell and are there only to facilitate later incorporation of 4Q Accounting into your application. You can ignore these fields in your use of 4Q Shell.

What's Required for 4Q

4Q Shell provides both tools and examples. The examples consist in a set of forms that support a simple flat-file address database. The tools consist in the nearly 200 system methods, many of which are demonstrated in the example.

Some shell methods are not used in the address example. Many of those that are do not need to be included in your application in order for it to be integrated with 4th Quarter Accounting.

Required Conventions

Your applicaiton must follow several 4Q Shell conventions in order for it to be merged with the 4th Quarter Accounting engine. Many other useful but non-required conventions are provided simply because they're useful. If you are building an application that you later want to integrate with the accounting engine you should understand the difference between what is required and what is not.

Required conventions pertain to areas of the application that must be shared, or those areas where there must be some cooperation. This includes the following areas:

Designer Password

After launching the shell application log on as Designer using the password "Designer".

Startup

There is only one startup sequence in an integrated application. In order to ensure that 4th Quarter Accounting is initialized correctly, you must follow 4Q Shell conventions for launching the application and for starting new processes.

This means that you must follow the conventions laid out in Chapter 2, "Starting the Application" on page 11.

Control Screen

While the use of a control screen is not strictly required, you must support some means of providing access to accounting functions in an integrated application. 4Q Shell provides you with a basic control screen that opens when the application is first launched.

If you use this control screen concept, when you integrate your application with 4th Quarter Accounting you will only have to provide a means of accessing your application and the 4th Quarter Accounting application on separate pages of the control screen.

If you use 4Q Shell's control screen, which has the same structure as 4th Quarter Accounting's control screen, then merging the two applications will be a simple matter of adding pages and navigation buttons to the single control screen form.

4Q Shell's control screen is described in the section entitled "Control Process" on page 15 of Chapter 2.

Messaging

4th Quarter has a complete and comprehensive interprocess messaging system written in native 4D code. The details of what this system does and how it operates are available in 4Q Developer Note #24, available from Braided Matrix.

The only aspect of the messaging system that must be included in an application is the checking for messages. This requires that you add the following 2 lines of code to every form you create that is used for on-screen display; this code is not needed in forms used for printing, importing or exporting:

```
: ((Form event=On Activate) | (Form event=On Outside Call))
    $Err:=_SYLayoutPhase ($CallerName;"";<Form Type>;
                          <Table Pointer>)
```

Where <Form Type> is the string value "input", "output" or "dialog" and <Table Pointer> is a pointer to the current table. This is discussed in Chapter 3' section "Message Passing" on page 26.

System Tables

4th Quarter Accounting uses about 60 tables. Of these the following 9 are central to the systems operation:

- Bug_Report — not used in 4Q Shell
- Help_Record — not used in 4Q Shell
- ID_Number — stores ID counters for all of the system's tables
- List_Item — provides multi-user access to lists
- Log_History — records database maintenance events
- Picture — not used in 4Q Shell
- Procedure_Lock — supports a table-based version of a semaphore
- System_Default — stores default database settings
- Special_Value — not used in 4Q Shell

With the exception of the ID_Number table you should not need to use any of these tables in your own code. The code that manages these tables is part of 4th Quarter Accounting and does not appear in 4Q Shell.

While it is not mandatory that you use the ID number system provided by 4Q Shell, it would be a good idea to do so for two reasons:

- ID numbers are crucial to the successful operation of any database and 4Q Shell's ID system is bulletproof.

- The 4Q Shell ID Number system is easily extended and it is wasteful to have two systems within an application that perform the same function.

Using the 4Q Shell ID system is described in the section entitled “The ID System” on page 67 of Chapter 6.

Menu Bars

4Q Shell contains only a few menu bars, with menu bar#1 being the main one. The only restriction on the use of menu bars is that 4th Quarter Accounting has control of menu bars #1 and #51-130. You are free to use the other menu bars.

Quitting

Quitting the application must be done by a call to _SYMBQuit. This starts an independent Quit process that sends messages to all currently active processes telling them to terminate. After all processes have ended, the Quit process issues the Quit4D command to return to the desktop.

The only thing you must do to use this command is to associate it with the Quit item on the File menu of any menu bars that you create. It is already installed on the File menu of menu bar #1.

This mechanism will work as long as you remember to include a call to _SYLayoutPhase in the On Outside Call or On Activate phases of all of your form methods. If you neglect to include this call in a particular form method, and that form happens to be displayed when the user selects Quit from the Fie menu, then that form will not receive the message to terminate and the Quit process will not issue the Quit4D command.

The _SYLayoutPhase method is described further in Chapter 3, “_SYLayoutPhase Method” on page 27.

Useful Conventions

The following examples, templates, and functions provided in 4Q Shell that are not listed above are not required. You do not need to use them in order for your final application to be compatible with 4th Quarter Accounting.

The List Screen

This includes the template for handling the list screen with all its functions and many system level methods. These perform tasks from message passing to string manipulation.

How you choose to interact with your users after they leave the control screen is of no concern to 4th Quarter Accounting. The processes and the

screens that are opened in your part of the application can follow any interface conventions you wish.

In general, the look and feel of 4th Quarter Accounting will be different from your application. Part of this will be due to different graphics, and part of it will be due to different options presented to the user.

This should not be considered a problem because the accounting system is not used by the same people who use the rest of the system. Also, there are restrictions on how accounting data can be handled that do not apply to other data. That is to say, accounting data is different by its nature. It makes sense for the accounting portion of the application to have its own look and feel.

On the other hand, if you are interested in supporting the same look and feel across your whole application, then you should follow the model for list screens in 4Q Shell. This means that you should reuse the code that appears in the Address table list forms. This code is discussed in Chapter 5.

System Methods

4Q Shell contains about 170 system methods. These are methods whose names start with the characters "_SY". These methods are grouped by function and then listed alphabetically, along with a brief description of each, in the "System Procedures" document.

20 of these methods are used for special purposes and do not appear in 4Q Shell. Others are used as part of black box systems and should not be called by you directly. This includes methods that function as part of the messaging system, the ID system, and similar systems.

A short list of simple system procedures that you may find useful includes the following:

Arrays

_SYBulletAraLin	Places or removes a bullet character
_SYFndInAray	Finds an element that best matches a specified value
_SYHdlSeltPop	Handles the action of clicking on a popup-type object
_SYSelFromArray	Creates current selection from array of record ID's
_SYSELinTextAra	Displays a text array from which the user chooses and item

Customization

_SYCustomDefltLayouts	Part of the startup routine
_SYCustomReadOnlyAll	Part of the startup routine
_SYCustomUnload	Part of the startup routine

DB Structure & Administration

_SYIncrmntSeqNm	Used to obtain sequence numbers
_SYMBQuit	Handles quitting from the application

Disk Files

_SYMakeDiskFile	Creates a text file on disk
_SYReadFileToAr	Reads the rows of a disk file to elements of text array
_SYTestFilePath	Tests for the existence of a specified file

Event & Error Handling

_SYOnErrCall	Handles nesting of calls to install On Err Call methods
_SYSetCanclKey	Sets and removes trapping for “Command+.” or “Control+.” key equivalents

Menus

_SYDisableMBs	Disables all menu bar items
_SYEnablMenu	Enables all menu bar items
_SYMrkAssocMenu	Marks and unmarks specified menu items

Processes & Transactions

_SYHdlProcess	Handles the creation of new processes
_SYTransaction	Manages the nesting of 4D transactions

Records

_SYFndDupsOfOne	Checks for records with duplicate field information
_SYModRcrd	Opens a record modification window
_SYPopAllOffStk	Pops all records off the stack

Selections

_SYClearCurrSel	Clears the current record and current selection
_SYDistinctValu	Returns array of distinct values from current selection
_SYTotAmtInSEL	Returns total in current section of indicated field

Sets

_SYCekSelUsrSet	Checks the contents of the UserSet
_SYLoadSet	Presents user with set management options
_SYTestEqualSet	Tests two sets for equality

String

_SYApndWildCard	Adds a wild card character if one is absent
_SYCaps	Capitalizes words
_SYCenterText	Centers text by adding left and right padding
_SYRemovSpacs	Removes spaces from text
_SYWrapText	Wraps text by inserting returns between words

User Interface

_SYAlert	Displays an alert box
_SYAsk3Choices	Message box with three choices
_SYAsk4Choices	Message box with four choices
_SYCustomReqst	Requests a text string from the user
_SYCustomCnfrm	Displays a configurable “confirm” dialog
_SYMessage	Displays and hides the message window

Utilities

_SYNil	Checks whether a pointer is a nil value
_SYShowUserWind	Displays 4D’s Splash Screen in the User Environment
_SYUserInGroup	Checks membership of user in multiple groups
_SYWait	Pauses process for a specified number of ticks

Windows

_SYCancelBox	Cancel command to be used with a close box
_SYClosWindow	Closes windows
_SYEnterBox	Accept command to be used with a close box
_SYOpnWindow	Opens windows
_SYSetWindSize	Sets window size

These system methods are described in more detail in the document from the 4th Quarter Training Manual entitled “System Module Methods”, available from Braided Matrix. They also conatin in-line documentation so that you can open them in the 4D method editor and learn more about them directly.

The Address Example

The 4Q Shell contains an example database for handling address records. This simple, flat-file database is provided to show you how to use many of the 4Q Shell system calls. The address example also demonstrates our approach to creating a user interface.

The user interface is largely independent from the system-level calls and conventions followed in the shell. You can use the address table methods and forms as templates for your own forms, or you can replace them entirely and use your elements of your own design. The only required elements that must be included in your code are those elements having to do with the startup and quit methods, and the passing of messages. These are described in detail in the rest of this manual.

The address example consists of a set of forms and supporting methods. These are described in detail in Chapter 7.

Adding Accounting

4Q Shell does not contain any accounting functionality. Nothing in this document pertains to the functional requirements of accounting. 4Q Shell's purpose is to provide the coding discipline that is required for the easy integration of your custom application with the 4th Quarter Accounting engine.

4th Quarter Accounting

When you decide that you want to add accounting to your application you will need to purchase a license to the source code of the 4th Quarter Accounting Solution application. This will give you from 12 to 16 megabytes of additional code and approximately 60 additional tables.

In most cases the 4th Quarter Accounting application will be larger than the application that you've created with 4Q Shell. In this case it will make more sense to move your code into 4th Quarter Accounting, than importing all of 4th Quarter Accounting into your application.

Merging Startup and Control

Once the two code bases are integrated into a single application you still do not have an integrated system. In fact, until you merge the two startup routines, you will only have access in the Custom/Runtime environment to one of the applications. The first thing you will need to do is merge the startup and the control screens.

Merging the startup methods means that all the calls needed to initialize both 4th Quarter and your application are called when the application is launched. This will be easy if you have constructed your application using the conventions provided by 4Q Shell.

Merging the control screens means either creating a single form that contains all controls for both 4th Quarter Accounting and your application, or creating two forms with some means to navigate from one to the other.

Integrated Accounting

Once the startup and the control screens have been integrated, you have two separate "programs" that exist within a single 4D application. You can then begin the tasks of sending accounting information from your application to 4th Quarter, and retrieving accounting information from 4th Quarter and displaying it in your application.

This final step has little to do with 4th Dimension, and is completely determined by the 4th Quarter Accounting application. 4th Quarter Accounting is

designed to make this job easier. It is at this point that the unique, patented design of 4th Quarter provides you with unequaled control and opportunity.

To support you in your task of sending and receiving accounting information 4th Quarter Accounting provides a series of tools to both the developer and the administrator. For example, there are methods that create accounting transactions and methods that create accounts.

The table structure of 4th Quarter Accounting is such that you will not need to make any changes to its core structure. The core methods, forms and menus are all completely modular. All you will need to learn are the calls you'll use in your system. 4th Quarter Accounting will then operate as an integrated accounting module alongside your application.

For more details on how this is done, please contact Braided Matrix directly. We have many articles and developer notes addressing various issues of interest. These are listed in the Resources chapter beginning on page 150.

CHAPTER 2

Starting the Application

In most cases it is an easy task to initialize variables when an application starts up. However, in 4Q Shell the assumption is that you will ultimately integrate your application with the 4th Quarter Accounting application. Initializing two applications to work together takes some coordination. 4Q Shell provides a simple framework in which to accomplish this.

4Q Shell is a multi-process application. Every list and entry screen opens in its own process. Since processes have separate record access and their own space of global variables, it is important that you initialize variables and handle record locking correctly.

4Q Shell includes a simple, automated system that initializes variables and keeps records unlocked. You only need to add code that handles your files and variables. 4Q Shell will call these methods at the required times as part of its normal operation.

This chapter explains what these methods do and when they are called. It explains how 4Q Shell handles the initialization of the application and the initialization of new processes.

Once this is accomplished we can go on, in the next chapter, to consider 4Q Shell's control screen and how to use it.

Launching the Application

On Startup

When 4Q Shell is first launched it runs the On Startup database method. This method executes in the User process.

4D automatically opens the User environment window when it begins executing the On Startup method. The first thing 4Q Shell does is to hide this window.

The On Startup method then calls each of the following methods. The string value “startup” is passed as the second parameter of each method. The last thing On Startup does is spawn a new process for the control screen.

After On Startup executes its last command it does not disappear, as do processes that you start yourself. This is because On Startup runs in the User Process, and the User Process always exists as long as you’re in the application.

When On Startup executes its last command it returns to a Waiting Event state. It will begin executing again when you drop into the User Environment.

Because the User process persists while your application is running, you must be sure to do three things before the end of the On Startup method:

- unload all records and place tables in a Read Only state
- assign a menu bar using the Menu Bar command.

  This will be the menu bar that will be displayed both when you drop into the User environment, and when you first exit the User and go back to the Custom Menus environment.
- assign values to all interprocess variables

The process management methods described below help you accomplish these tasks.

the Design & User Environments

To get to the User environment from the Custom Menus environment you must first open the splash screen. To do this select “Show User Window” from the Designer menu. This opens the splash screen in the Custom Menus environment.

Press Alt+F4 on Windows, or Optn-f on the Mac. This drops you into the user environment. You have now existed from the Custom Menu environment. Press Ctrl+Y on Windows, or Command+y on the Mac to reach the Design environment.

Process Management Methods

The following four methods are reserved exclusively for your use. These methods are empty in the version of the shell supplied to you, but they are called as part of normal operations. Modify these methods to include all the code that is specific to your application.This code will be executed automatically.

Each of these methods is automatically called when the application first starts and whenever a new process is spawned. In each case 4Q Shell passes these methods a string value in the second parameter that takes the value “startup” when the application is first launched. A blank string is passed when subsequent processes are started. Each method returns a longint error code.

The 4Q Shell method named “__SYCustomInit” calls the following __SY_AddressInit method. __SYCustomInit is the method in which you place calls to your own initialization methods. Initialization is performed when the applicaiton is first launched and again any time a new process is spawned.

All initialization methods should returnan integer-valued error code. __SYCustomInit will pass this code back to the method that called it. The calling method monitors this error code and takes different actions depending on its value.

Error codes passed back by __SYCustomInit are interpreted as follows:

- equal to or greater than zero: everything executed successfully.
- negative value: a fatal error occured. 4Q Shell will jump to the end of the initialization routine and exit the process.

__SYCustomInit

Place two kinds of code in this method: code that initializes interprocess variables and code that initializes process variables.

Process initialization code should be placed in the __SYCustomInit method supplied with the 4th Quarter shell. I suggest that you create a series of initialization methods, one for each of your tables (or modules, depending on how you organize your application).

Call these methods from within the __SYCustomInit method. Pass the second parameter to each of your procedures. Inside your initialization methods this parameter is referred to as $2. This parameter will have the value “startup” when the application is first launched. It will be a blank string in subsequent calls.

By monitoring the value of $2, your initialization methods will know whether or not to initialize both interprocess and process variables or just process variables.

__SY_AddressInit

__SY_AddressInit is an example of an initialization method. This method initializes variables and records needed for handling the Address table. The code is listed below.

```
$CallerName:="GP-__SH_AddressInit"
   `$1=caller name;
   `$2=action code;
   `$3=error code;
C_BOOLEAN($4QU981020)
C_LONGINT($0;$Err)
C_TEXT($1;$2;$Action)
$Err:=0
$Action:=$2

If ($Action="STARTUP")
  ARRAY TEXT(<>varARSchNam;0)
  <>vSeqAddr:=134    `address records sequence number
  <>vARSet:="GeneralAddressSet"
  <>vARSrt_A:=->[Address]Last_Name
  <>vARSrt_B:=<>vSYNilPtr
  <>vARSrt_C:=<>vSYNilPtr
  <>ARSrt1UP:=True
  <>vARSrt2UP:=True
  <>vARSrt3UP:=True
  LIST TO ARRAY("AddressPopup";<>vyGOControl1)
  <>vyGOControl1{0}:="Address"
  <>vyGOControl1:=0
  LIST TO ARRAY("PhoneTypes";<>vyPhoneType)
  LIST TO ARRAY("Salutation";<>vySalutation)
$Err:=_SY_IDRecord ($CallerName;"CREATE";<>vSeqAddr;1
            ;Table(->[Address]);Field(->[Address]Address_ID))
End if
COPY ARRAY(<>vyPhoneType;vyPhoneType1)
COPY ARRAY(<>vyPhoneType;vyPhoneType2)
COPY ARRAY(<>vySalutation;vySalutation)
$0:=$Err
```

The code in this method is discussed in different places in this manual according to where these variables are used. The interprocess variables are largely related to the list form and are discussed in Chapter 5, “Global Variables for Output” on page 36.

The assignment of <>vSeqAddr and the call to _SY_IDRecord are part of the ID number system discussed in Chapter 6, “ID Numbers” on page 77.

Interprocess Variable Initialization

This is code that only needs to be run when the application is first started. Here you assign values to interprocess variables and perform data initializa-

tion, data verification, and routine maintenance all of which is executed whenever the user logs on. This code should be run whenever the value “startup” is passed in the second parameter.

Process Variable Initialization

This includes the assignment of values to process variables. Process variables are undefined in an interpreted application whenever a new process begins. This code should be run whenever the method is called, regardless of the value provided in the second parameter.

_SYCustomDefIt-Layouts

Place in this method the default input form and output form assignments for each of the files that you add to the system. This ensures that 4th Dimension will know what forms to use when you issue a Modify Record, or Display Selection command. It is good practice to redeclare these assignments before issuing a command that makes use of them. It is also good practice to set the default forms to their most likely values at the beginning of each process.

_SYCustomRead-OnlyAll

Place in this method commands that set each of the files that you add to the system to Read Only mode. Calling this method will ensure that whatever record management actions you perform will not leave any records loaded and locked.

You should call this method whenever you exit from a form, process, or event loop that may have left some of your files in Read Write mode. This is used in conjunction with _SYCustomUnload, which is described next.

_SYCustomUnload

Modify this method to contain commands that unload the current record from all of your tables. This ensures that records are available to others on the network before you begin operations in a new process.

This method is called in conjunction with _SYCustomReadOnlyAll. It is used frequently to make sure that stray records are not loaded in Read Write mode, as this would cause them to be locked to other users.

Control Process

After the initialization methods are called from the On Startup method, 4Q Shell starts a new process to handle the control screen. This process calls the above-mentioned methods again because the control process, like any process, has its own space of global variables that need to be initialized.

However, when the initialization methods are called for the control screen, they are passed a blank string in their second parameters. This is because the startup phase has already been handled when On Startup was called in the User process.

Control Screen

All actions begin at the control screen. The control screen is the only screen that is visible when the application first opens. To display a list of records the user must choose an option on the control screen, or select an item from the menu bar. In either case a new process will be spawned.

When the user selects a separate menu item, or an item from a popup on the control panel, the application will call a method that you write to handle further processing.

An example of this is provided in the method _SYHMBAddrList. This method simply initializes an array and passes a pointer to this array, along with some other information, to the 4Q Shell method named _SYOpenProcess.

The action of this method is to start a new process. In version 6 of 4th Dimension you can now pass parameters to processes. The passing of parameters is handled for you automatically through the use of the aforementioned array. A pointer to this array is passed as a parameter in the _SYOpenProcess command that is shown below.

In this case the new process is named “SampleList” and it is being controlled by a method named “SampleUserArea”.

```
ARRAY TEXT(var1SYText;1)
var1SYText{1}:=<>vSYquo+"LIST"+<>vSYquo
$Err:=_SYOpenProcess ($CallerName; "NEW"; "SampleUserArea";
<>vSYDfltPrcs;"SampleList";False;"";0;->var1SYText)
```

This call will start a new process under the method “SampleUserArea”. SampleUserArea is the name of the process handling method that you create.

Control Process Method

The control screen is a dialog that runs in its own process. This process is handled by the method named _SYCtrlScreen. Here is a pseudo-code outline of what this method does:

```
Initialize the control screen process (set global variables, etc.)
Retrieve any pending messages sent from other processes
Set variables that determine the size of the control screen window
Open the control screen window
Enter a While loop
      Unload all tables, set all tables to Read Only
      Display the [ID_Number];“ControlScreen_d” screen dialog
```

Exit While loop if the user wants to quit the process or application
Close the control screen window
Exit the control screen process

The actual code in the _SYCtrlScreen method contains some additional tests and conditions. Refer to the code in this method if you are interested in the details. You do not need to know the details of the method to use it.

The only change you may want to make to this method is the width and height of your control screen. These values are assigned to the two local variables named $ControlWidth and $ControlHeight. The fewer alterations you make to existing 4Q Shell procedures the better.

Control Screen Dialog

The control screen dialog is stored with the [ID_Number] table. It is important that the dialog be stored with this table in order for the window to open to the indicated size. To facilitate correct window sizing the control dialog must also be set to either Automatic Size, or its size must be based on an object that appears on the form.

These options are accessed through the Sizing page of the Form Properties dialog in the forms editor. To reach this screen, open the dialog and select “Form Properties...” from the Form menu. Then click on the “Sizing Options” tab.

The sample control screen has the following two Menu/Drop-down list objects.

User Areas popup

This popup is assigned to the variable <>vySYUsPrcNa. This popup displays the names of the currently running processes. As the user opens processes, these processes are added to this array. The user can select from this popup to bring the indicated process to the foreground.

Processes are added and removed from this array automatically. You don’t need to add any coding of your own for this popup menu to work with whatever processes you add.

Address popup

This popup has two elements, List and Add. Selecting List opens a new process that displays addresses in a List form. Selecting Add opens a new process that presents a blank new address record.

The address popup works by calling the method _SYCtrlPanelPop. As you add or modify the objects on the control screen, you may choose to have your objects also call the _SYCtrlPanelPop method. If you do, then you'll need to modify _SYCtrlPanelPop to handle your specific requests.

_SYCtrlPanelPop

This method handles requests for access to the Address table. It either spawns a new process to open an address list, or a new process to open an address entry screen.

The method takes three parameters, as listed below. The third is a pointer to the object on the control screen that called the method.

```
$CallerName:="GP-_SYCtrlPanelPop"
   `$1=$CallerName;
   `$2=action code
   `$3=»control screen variable;
C_BOOLEAN($4QU981001)
C_STRING(80;$1)
C_TEXT($2)
C_POINTER($3;$Control_)
C_LONGINT($Err)

$Err:=0
$Control_:=$3
Case of
 : ($Control_=(-><>vyGOControl1))
  Case of
   : (<>vyGOControl1=1)    `display address list
    ARRAY TEXT(var1SYText;1)
    var1SYText{1}:=<>vSYquo+"LIST"+<>vSYquo
     $Err:=_SYOpenProcess ($CallerName; "NEW";
          "_SHAdressUserArea"; <>vSYDfltPrcs; "Address List";
          False;"";0;->var1SYText)

   : (<>vyGOControl1=2)    `display address entry screen
    ARRAY TEXT(var1SYText;1)
    var1SYText{1}:=<>vSYquo+"ENTRY"+<>vSYquo
     $Err:=_SYOpenProcess ($CallerName; "NEW";
          "_SHAdressUserArea"; <>vSYDfltPrcs; "Address
          Entry"; True;"";0;->var1SYText)
  End case

 Else
```

```
    _SYAlert ($CallerName;"Programming error: unrecognized control
            screen instruction passed to "+$CallerName+" by "+$1)
    $Err:=-1
End case
```

By monitoring what object $3 is pointing to, you can modify the _SYCtrlPanelPop method to respond differently when called by different objects.

For example, if you added a Menu/Drop-down list that used the array <>arSample, then you could call _SYCtrlPanelPop from the object method of <>arSample, passing a pointer to <>arSample:

```
_SYCtrlPanelPop ("[Id_Number] Control_Screen; <>arSample"; "";
Self)
```

In the body of _SYCtrlPanelPop you would then add the Case Of statement

```
:($Control_=(-><>arSample))
    Case of
    :(<>arSample=1)
    'The first element was selected, perform the indicated action.
    :(<>arSample=2)
    'The second element was selected, perform the indicated action.
    end case
```

The objects you add to the control screen can be managed by adding additional Case Of tests in the _SYCtrlPanelPop method.

After Startup

Once the initialization methods have run and the control screen is displayed, the user can begin to navigate through the database. The control screen is the central user interface for 4Q Shell application. The character of the application is largely determined by what you place on this form.

The 4Q Shell concept is that each action performed from the control screen opens a window in a separate process. The control screen itself does little more than open processes that provide functionality and access to your data.

In the next chapter I consider what it takes to open processes. I assume that you will be following the 4Q Shell interface conventions. This can be done using 4Q Shell's preexisting methods.

CHAPTER 3

Processes

4Q Shell is a multi-process application. Multiple processes require extra work to initialize the variables and selections that are specific to those processes. 4Q Shell provides mechanisms to make this task easier.

4Q Shell is designed with the idea that each area of your application opens in its own process. This raises the issue of determining what each area does.

Should you support a separate area for every table? Should some areas handle multiple tables together? These are questions of database design. They are questions that you must answer for yourself.

The data management functions in 4Q Shell provide a basic suite of services that include searching, sorting, and reporting. These are operations that can apply to any kind of information. They should not be seen as applying individually to every table in your database.

This chapter explains what these functions do and the code that is used to call them. It also explains what variables and conventions 4Q Shell requires in order to support these functions. You will have to follow these conventions if you want to adopt 4Q Shell structure and reuse the maximum amount of 4Q Shell code.

Designing Software

If you are designing your application in parts that correspond to general functions, which you should do, then you should create separate methods that group together the handling of related tasks.

Designs that follow this prescription are referred to as “architecture driven”. This is distinguished from an application whose design is broken into parts that correspond to the different tables the application contains. Such a design is called “structure driven”.

In an architecture driven design your application is divided into modules that correspond to areas of function, such as Sales, Inventory, and Operations. In a structure driven design your application is divided into modules that correspond to tables in the database such as Invoices, Line Items, and Contacts.

In a crude sense you can equate the different tables in an application with the different functions that application performs. You might think that with enough work this correspondence can be perfected. This is untrue. Structure and function are intrinsically different in all but the simplest applications.

Structure driven designs are easier to program, but they are less functional. They are also harder to modify and to maintain in real-world applications. That's because the evolution of an application used in an actual production environment is determined by is function, not by its internal structure.

The distinction between structure and application driven designs becomes greater and more important in more complex applications. However, for the purposes of describing the 4Q Shell these distinctions will be largely overlooked. This is because a discussion of design principles is beyond the scope of this manual. The most I can do is to remind you to break your code into functional modules, each of which may manage one or more tables in the database.

Learning Design

Good design is ultimately more important than well-written code. It is a process that actually precedes the use of 4Q Shell. 4Q Shell can only provide you with a means of producing well-written code.

Application design is more of an art than a science. Learning it requires more theory than learning to program or learning to use a programming shell. If you are interested in learning more about application design I can provide you with references to various sources. Those that I have written can be provide to you by Braided Matrix. These articles are listed in the section entitled "Database Design" on page 150 of chapter 10.

Control Process

As I have mentioned, after the initialization methods are complete, 4Q Shell starts a new process to handle the control screen. This process calls the above mentioned methods again to initialize the control process variables.

When the initialization methods are called for the control screen they are passed a blank string in their second parameters. This indicates that the appli-

cation startup phase has already been handled when On Startup was first called in the User process.

Your Process Handling Methods

You must write methods to handle your processes. In 4Q Shell I suggest that these methods support two different actions: the creation of new records, and the display of existing records. These two actions can be distinguished by the value of a parameter passed to the methods when the processes are started.

The 4Q Shell provides an example of this in the method named _SHAddressUserArea. The code has three main parts:

- Initialization:
  this is a call to __SY_ProcesInit, which handles all initialization.
- List Display:
  this is handled in the :($1=”LIST”) case, and involves a call to Display Selection.
- Entry Display:
  this is handled in the :($1=”ENTRY”) case, and involves a Repeat/Until loop that makes a call to the Add Record command.

In the following adaptation of this method I have replaced references to the Address table with references to the Sample table. This method is shown below.

SampleUserArea Method

```
$CallerName := “SampleUserArea”
$Err:=__SY_ProcesInit ($CallerName;"Sample")
Case of
  : ($Err<0)
    _SYAlert ($CallerName;"Unable to initialize the Sample area.")

  : (Count parameters<1)
    _SYAlert ($CallerName;"Programming error: too few parameters
              supplied to "+$CallerName+". The process cannot start.")

  : ($1="LIST")
    $Err:=_SYEnablMenu ($CallerName;"LIST_AREA")
    _SYOpnWindow ("";<>vSYStdWin_W;<>vSYStdWin_H;
                  Plain window;"";"_SYCancelBox")
    vSelect:=False
    ALL RECORDS([Sample])
```

```
    OUTPUT FORM([Sample];"qSample_o")
    DISPLAY SELECTION([Sample];*)
    _SYClosWindow
    $Err:=_SYHdlProcess ($CallerName;"EXIT";"";0;
                         String(Current process))

  : ($1="ENTRY")
    $Err:=_SYEnablMenu ($CallerName;"ENTRY_AREA")
    _SYOpnWindow ("";<>vSYStdWin_W;<>vSYStdWin_H;
                  Plain window;"";"_SYEnterBox")
    INPUT FORM([Sample];"qSample_i")
    Repeat
      ADD RECORD([Sample];*)
    Until ((bCancel=1) | (bdrDelete=1) | v4QExitPrcs)
    _SYClosWindow
    $Err:=_SYHdlProcess ($CallerName;"EXIT";"";0;
                         String(Current process))
  Else
    _SYAlert ($CallerName;"Programming error: unrecognized
                         parameter passed to "+$CallerName)
  End case
```

Global Variables

Up to this point I've spoken only about how 4th Dimension handles variables when new processes are spawned. Now I want to address how 4Q Shell uses variables in particular.

If you decide to follow 4Q Shell conventions for designing and handling your entry and list forms, then you will need to understand what variables 4Q Shell relies upon, and how to use these variables.

If you only use 4Q Shell for low-level system operations, such as presenting a control screen and assigning ID numbers, then the following discussion will not apply.

Initializing at Startup

Each table that uses 4Q Shell routines to display records through a list form needs a set of global variables. These variables are used in the 4Q Shell methods. Each table that needs its own set of globals should have values assigned to these variables in a startup method that is specific to that table. These methods should be called from __SYCustomInit, as discussed above.

If you build your list forms using the 4Q Shell model, then they will support a suite of functions that include searching, sorting, and reporting. An example of the standard 4Q Shell structure is provided by the form “qAddress_o”, stored in the [Address] table.

List methods require that you define and assign values to the following global variables. These globals should have names that are unique to every file that you are going to display.

In the following example I’ve given the variables a name that starts with “Sample”. This is different from the variable names that appear in the qAddress_o form.

You should start your global variable names with characters that indicate the table they are related to. For example, variables for an inventory table could begin with “Inven”, “IY” or a similar tag.

<>arSampleSchNam	text array	A popup array displaying search options on the list form. Initialize with size of zero elements.
<>SampleSet	text	Name of set that stores the current selction of the particular table. Assign it a unique name.
<>SampleSrt_A	pointer	Pointer to the 1st sort field. Initialize it by assigning it a pointer to the most likely sort field.
<>SampleSrt_B	pointer	pointer to the 2nd sort field. Initialize it by assigning it the value <>vSYNilPtr. This indicates that the 2nd sort level is currently unused.
<>SampleSrt_C	pointer	Pointer to the 3rd sort field. Initialize it by assigning it the value <>vSYNilPtr. This indicates that the 3rd sort level is currently unused.
<>SampleSrt1UP	boolean	This indicates whether the first sort is an ascending sort, when true, or a descending sort, when false. Set it to whatever is most useful. Usually this would be the value “true”.

<>SampleSrt2UP	boolean	Indicates ascending or descending order of the 2nd sort, when used. Set this to “true”.
<>SampleSrt3UP	boolean	Indicates ascending or descending order of the 3rd sort, when used. Set this to “true”.
SampleWNum_o	boolean	Stores the 4Q Shell’s window number for the current window. This allows you to distinguish different windows in the same process. This is not the same as the 4D window number which is unique across all processes. Initialize this to zero.

These variables could appear in a method that could be called “SampleInit” as follows:

```
Array Text(<>arSampleSchNam;0)
<>SampleSet := “CurrentSampleSet”
<>SampleSrt_A := ->[Sample]MyField1
<>SampleSrt_B := <>vSYNilPtr
<>SampleSrt_C := <>vSYNilPtr
<>SampleSrt1UP := True
<>SampleSrt2UP := True
<>SampleSrt3UP := True
SampleWNum_o := 0
```

Message Passing

Message passing has more to do with what you want your application to do than with the design of your application. However 4Q Shell uses message passing to support its most basic functions. That’s why we need to consider it as part of basic process management.

There are two basic types of messaging activity: active and passive. Active message handling means that you pass a message to a target process or check the current processes message list to retrieve any messages. 4Q Shell does not require that you support any active message passing.

Passive message handling involves listening for messages sent to the current process unexpectedly. You must include certain passive message handling routines in order to use 4Q Shell code. If you place these routines incorrectly, certain 4Q Shell operations will not function.

Active message handling is done through calls to the 4Q Shell methods _SYMsgSend and _SYMsgReceive. Passive message handling is done through the On Outside Call and On Activate phases of 4D forms. For more information about active message handling, consult 4Q Developer Note #24. Passive message handling is all that will concern us here.

_SYLayoutPhase Method

On Outside Call and On Activate form events are triggered by explicit calls from other proceses to the current process. They can also be triggered by bringing a window to the foreground. A window can be brought to the foreground either manually or procedurally.

Each of these form events calls the _SYLayoutPhase method, which checks the message queue for messages. If a message is received its contents are placed in the variable vSYMsgText. 4Q Shell is only programmed to act on one particular message. That is the message “exit_process”, which is interpreted as an indication to quit the current process.

In order for passive message passing to work you *MUST* include a call to the _SYLayoutPhase method in the method of *ALL* forms that are displayed onscreen. This ensures that the 4Q Shell message passing system will function reliably. You do not need to include calls to _SYLayoutPhase in forms used for printing, importing and exporting.

Supporting passive message passing is essential because this is how 4Q Shell quits the application. When the user selects the Quit item from the File menu, a message is sent to all open processes asking them to quit. 4Q Shell waits until all these processes have quit before returning to the Desktop.

v4QExitPrcs Variable

When the _SYLayoutPhase method receives the message “exit_process” it sets the v4QExitPrcs variable to the value “true”. _SYLayoutPhase will then issue the Cancel command to exit the current form. In addition to this, you must be sure to include a test for this variable in any indefinite loop you write that encloses the dislay of forms.

For example, if you have a repeat loop that repeats indefinitely until the user clicks a certain button, you must add an additional exit condition that will be satisfied when v4QExitPrcs has the value “true”.

There are, in fact, few situations where you would enter indefinite loops. The only time this commonly occurs is in the add record cycle where new records keep being created until the user cancels the entry screen. This is one place where you should test the value of v4QExitPrcs and exit if it has the value true.

An example from the SampleUserArea method is:

```
Repeat
    ADD RECORD([Sample];*)
Until ((bCancel=1) | (bdrDelete=1) | v4QExitPrcs)
```

Here the Repeat/Until loop will be exited when either of three conditions prevail: the cancel button was pressed, the delete button was pressed, or the v4QExitPrcs variable is set to true.

By making adding a condition to exit from loops when the v4QExitPrcs variable is true, your processes will terminate when they are told to.

What Happens Inside a Process

All we have done so far is set up variables and explain the structure of processes. We have not yet done anything with the data. We have not displayed anything to the user.

Once a process has been started and initialized we can address the two main user interface areas: the display of lists and the entry of new records. These are the topics we deal with next.

We'll deal with list forms first. List forms involve more 4Q Shell methods than entry forms. These methods include canned routines for locating and displaying information.

In the subsequent chapter we will consider entry forms. There we will examine management of ID numbers.

CHAPTER 4

Menu Bars

4Q Shell uses three sets of menu bars. In addition custom applications can assign associated menu bars to forms.

General Menu Bar Use

4Q Shell uses two menu bars for each of the following three areas:

- Control Screen
- List Forms
- Entry Forms

In each of these areas there is one menu bar for users in the Design group, and a second menu bar for all other users.

Control Screen Menu

Menu Bar #1 is installed when users in the Design group view the control screen. This menu is also installed automatically by 4D when any user first opens the User environment. Menu Bar#2 is installed when users who are not in the Design group view the control screen.

The Control Screen menu bars are installed by the method _SYEnablMenu when it is called with the second parameter “CONTROL”.

The _SYEnablMenu is called before the window is opened in which the control screen is displayed. In the case of the address book example, this method call occurs in the _SYCtrlScreen method. The _SYCtrlScreen method controls the Control Screen user area.

The control screen menu bars both have a File and an Address menu. Menu bar #1, the Designer’s menu bar, also has a Designer menu.

The items on these menus, and the corresponding menu item methods are shown below.

TABLE 1. Control Screen Menu Bars

Menu#1 Items	Menu#2 Items	Methods	Description
File	**File**		
Page Setup	Page Setup	_SYMBPageSetup	Opens 4D's page setup dialog.
Control Screen	Control Screen	_SYMBCntrlScreen	Acts as toggle to alternatively show and hide the control screen.
Preferences	Preferences	_SYMBUserArea	Opens the User Preferences screen.
Maintenance	Maintenance	_SYMBMainArea	Opens the administrator's Maintenance screen.
Quit	Quit	_SYMBQuit	Quits the application.
Addresses	**Addresses**		
List	List	_SYMBAddrList	Opens the Address list screen in its own process.
Add	Add	_SYMBAddrAdd	Opens an Address Entry screen for a new address list screen in its own process.
Designer			
User Preferences		_UTILMBPrefs	Opens the Designer Preferences screen.
Manage Control Screen		_UTILMBCntrlScren	Terminates or restarts the Control Screen process.
Show IP Messages		_UTILMBOpnMsgWin	Opens the Interprocess Message list screen in its own process.
Show User Window		_SYMBShowUserWInd	Opens 4D's splash screen from which you can drop into the User and Design environments.
Hide User Window		_SYMBHideUserWind	Hides 4D's splash screen.

In a database created from the 4Q Shell the Address menu bar would be replaced by another labeled appropriately for the application. Additional items or menu bars should be added as needed.

List Form Menu

Menu bars #3 and #4 are used for list forms in the Address example.They both have the same items, as shown in the table below.

Even though these two menu bars are identical in the sample database, it is a good idea to have a separate menu bar reserved for the Designer. This menu bar can be used for development or testing at any time.

TABLE 2. List Form Screen Menu Bars

Menu#3 Items	Menu#4 Items	Methods	Description
File	**File**		
Page Setup	Page Setup	_SYMBPageSetup	Opens 4D's page setup dialog.
Show Control Screen	Show Control Screen	_SYMBGoCtrlScrn	Brings the control screen to the foreground, opening it if closed.
Quit	Quit	_SYMBQuit	Quits the application.
Selection	**Selection**		
Search by Example	Search by Example	_SYMBSchByExample	Opens a table-specific Search by Example screen.
Show All	Show All	_SYMBShowAll	Displays all the current records in the table.
Focus	Focus	_SYMBFocus	Reduces the current selection to the currently highlighted records.
Sort	Sort	_SYMBSort	Opens the system's Sort dialog.
Sets	Sets	_SYMBSets	Opens the system's Set Management dialog.
Reports…	Reports…	_SYMBReports	Opens the system's Report & Media screen listing only those reports appropriate to the current table.

The methods linked to the items on the Selection menu do not actually perform the actions described. These methods only trigger a script attached to a button on the current list form. These button scripts are responsible for calling the methods that perform the actions described.

The details of how this works is described in the section entitled "List Menu Bar" on page 41 of chapter 5.

Entry Form Menu

Menu bars #6 and #7 are used for entry forms in the Address example.They both have the same items, as shown in the table below. The menu bar reserved for the Designer is meant to be used for development or testing.

TABLE 3. Entry Form Screen Menu Bars

Menu#6 Items	Menu#7 Items	Methods	Description
File	**File**		
Page Setup	Page Setup	_SYMBPageSetup	Opens 4D's page setup dialog.

TABLE 3. Entry Form Screen Menu Bars

Menu#6 Items	Menu#7 Items	Methods	Description
Show Control Screen	Show Control Screen	_SYMBGoCtrlScrn	Brings the control screen to the foreground, opening it if closed.
Quit	Quit	_SYMBQuit	Quits the application.

The items on these menus trigger actions that are not specific to the forms, records or tables being displayed. Because of this the menu item method shown above supply all the code needed to perform the associated action.

In 4Q Shell the entry form menu's do not offer actions that are specific to the form, record or table.

It is possible to customize them to provide such actions, such as record deletion, duplication, or duplicate checking. If you want to do this, then I suggest following the same approach used for items on the Select menu of menu bars #3 and #4.

That is, I suggest you use the menu item methods to trigger buttons on the forms themselves. This will enable you to write very little inappropriate shared code, and instead have the forms retain maximum control over the record displayed.

Another method available for placing table specific actions on menu bars is to use associated menus. This is appropriate if the actions being handled are relevant only to one or maybe a few tables.

An associated menu bar can be created to carry these special items. The menu bar is then linked to only those forms where it's needed using 4D's "Associated Menu Bar" setting in the form properties window.

Entry form menu bars are described further in the section entitled "Entry Menu Bar" on page 69 of chapter 6.

Active Menus and Associated Menus

In order for the installed menu to be active when a form is opened you must assign a negative menu bar number as the form's associated menu. In 4D v6.0 and earlier it was possible to specify the negative value of a nonexistent menu.

In 4D v6.5 assigning a negative, nonexistent menu bar no longer activates the currently installed menu. Instead, you must use the negative of an actually existing menu. If this menu has menu items, then these items are appended to the currently installed menu.

In order to activate the currently installed menu you must assign as an associated menu the negative value of a menu bar that has no items. Menu bar #5 was created for just this purpose.

Assign the value “-5” as the associated menu bar for any form which is to interact with the installed menu bar. The same holds true if you want the menu bar to be active in a form displayed as a dialog.

CHAPTER 5

List Forms

4Q Shell has a comprehensive suite of conventions and methods for the management of list forms. These conventions and utilities do not need to be used in order for your application to be integrated with 4th Quarter Accounting. However, using these tools will give you easy access to sophisticated functions, and unify the appearance of the final integrated application.

All standard 4Q Shell output forms have the same structure. They are called using the 4D command Display Selection (not Modify Selection, as explained below). They display records in a standard 4D output form. All functions are handled from buttons placed in the footer of the list form. Menus provide only basic navigation.

Calling the Output Form

All actions begin at the control screen. The control screen offers the user the choice of displaying a list of records or adding a new record. Both of these options are handled by spawning a new process.

The steps that 4Q Shell suggests you follow to open the list form have been described in the chapter “4Q Shell Processes”. This entails creating a method to respond to the user’s actions in the control screen, and creating a process handling method to control the new processes they spawn.

In this process handling method you call the Display Selection command. This shows the user a list of records in the desired file.

Display versus Modify Selection

4Q Shell uses the Display Selection command rather than the Modify Selection command because of a quirk in how 4th Dimension operates. The quirk is that when you double-click on a record in a list screen, 4D will always open that record in an entry form. If the Modify Selection command is used, then 4D automatically alerts the user if the record is locked.

In our case the indicated record is opened in Read Write mode in another process. As a result, it is sure to be locked in the process that displays the list. In

order to avoid the irrelevant message that the record is locked we must display records using the Display Selection command.

In order to display records in a list form you'll need to create the following elements:

- initialization routines for global variables
- an output form
- methods to handle actions that can be triggered from the output form

I discuss each of these items below in turn.

Global Variables for Output

Initializing at Startup

Each table that displays records through an output form using 4Q Shell routines needs a set of global variables. These variables are first assigned values in a startup method called in the __SYCustomInit method discussed in the previous chapter.

The methods discussed in this chapter assume that you have initialized these variables correctly. Certain 4Q Shell methods require parameters that are pointers to these variables.

4Q Shell List Form

The list form consists of graphical elements placed on the form using the form editor. This list form also relies on code placed in the form method. 4Q Shell suggests a structure for both of these.

The List Form

Header

Above the header line place the column headings for the values that will be displayed in the detail area. Versions of 4th Dimension prior to version 6.5 do not support any active objects in the header. 4D v6.5 and later do allow you to put buttons and other objects in the header. 4Q Shell is currently written for 4th Dimension version 6 and uses the header only for column labels.

Code used to assign value to variables that appear in the header area is placed in that area of the form method that handles the On Header form event. This will be described below.

Detail

The detail area is repeated, or redrawn, once for every record that appears in the window. This means that 4D only loads those records that it displays regardless of the total number of records that are in the current selection.

The detail area can contain fields or variables. It cannot contain active objects like buttons, included forms, or objects set to have the “variable size” property.

Code for the detail area is placed in the On Display Detail area of the form method.

Break

The break level performs no function in the display of records on-screen. The Break Level line is placed on top of the Detail line in the form editor.

Footer

The footer contains all active objects. 4Q Shell uses footer objects to support most user functions. In particular, we use the footer to support functions specific to the current table. This includes searching, sorting, reporting, adding, modifying and deleting. In addition, the footer can be used to display totals or other summary information.

There is no form event corresponding to the display of the footer. There is no easy way to determine when 4D has finished displaying the last record in the on-screen list.

A description of each of the active objects included in the standard 4Q Shell output form is provided below.

The List Form Method

The form method consists of a Case Of statement that tests for particular form events. The basic form events are On Header, On Display Detail, and On Open Detail.

There is also code for additional form methods that play a less obvious, but still important role. We’ll describe each of these execution phases. We’ll refer to the [Address];”qAddress_o” form method in the following description.

(Form event=On Load)

This event occurs only once before the output form is displayed. In this section of your code you should include all initialization and set up that needs to be performed for the list form.

Because the On Load phase will not run again, and because some actions need to be repeated whenever the screen is redrawn, there are some actions that are needed in both the On Load phase and other phases as well. This includes assigning a window title, for example. Actions of this nature are handled in the On Header, or On Load, or Before Selection section described below.

In 4Q Shell we perform the following functions in the On Load phase:

- *Set the visibility of the Select button*

  This button is used when the output form is used to solicit the user's choice of a particular record. If the output list is just used to display records, this button is set to be invisible.

- *Initialize the search popup*

  A call to the method _SHAddrHdlSchPop is made to initialize the search popup object that appears in the footer area. For a description of this method see the Search Popup described in the next section.

- *Assign the UserSet to the current table*

  You must create an empty set named "UserSet" and associate it with the current table. If you fail to do this the pre-existing UserSet will continue to reference which ever table was last assigned.

  Since the UserSet is used as the primary means of determining what records the user has highlighted it is important that it be associated with the current table when this table is first displayed.

- *Assign a value to the form's window number variable*

  4Q Shell's _SYOpnWindow command increments a process variable named vWindowNum every time a window in the current process is opened. It decrements this counter every time a window in the current process is closed.

  By assigning the current value of this window counter to a process variable used strictly with this form, we can determine if the current window is in the foreground whenever the form method runs.

  If the counter value of vWindowNum equals the window number stored with the output form, then the output form is the front most window. If the window counter is greater than the number stored with the form, then there are one or more windows open in front of it.

(Form event=On Display Detail)

This phase runs every time a record is displayed through the list form. This phase is typically used to assign values to variables that present information from the current and related records.

There are various cases where you might use a variable in the detail section instead of a field. For example, you might want to concatenate two fields

together, such as first plus last name. Or the record may store a code whereas you want to display a value that is more meaningful to the user.

The On Display Detail phase runs after the current record has been loaded, and before the current record is drawn to screen.

(Form event=On Open Detail)

This new 4th Dimension version 6 phase enables us to trap a double-click in the output form. The On Open Detail runs after the user has double-clicked but before the record is displayed in the input form.

In this phase you can perform tests of current record, set values based on the current record and, in the case of 4D Shell, display the current record. However, you cannot stop 4th Dimension from opening the current record in the current Input Form after the On Open Detail phase has run.

In 4Q Shell we use the On Display Detail phase to spawn a new process that opens its own window and displays the current record. This is done with a call to a 4Q Shell method named _SYModFromList. This is a generic method which can be called from any output form. This method takes 3 parameters:

_SYModFromList Method

```
$1= name of the method that is calling it (string 80)
$2=pointer to the table being modified
$3=pointer to the ID field in the table being modified
$0=error code, 0 if all ok (longint)
```

_SYModFromList spawns a new process that is managed by the method _SYUAModFromList. This is also a generic method that functions like a black box to display the current record.

Because this method controls a new process it expects that you have made the necessary modifications for initiating processes as described in the Processes chapter of this manual. These changes enable 4Q Shell's black box routines to perform appropriate functions for each of the tables being displayed.

After spawning a process to display the current record, we do not want to display the current record in the current window. That is, we do not want to replace the list view with the input form.

This presents a problem because 4D insists that it will display the current record. In order to accomplish this we use a trick: we set a special blank layout as the input form and let 4D open and then immediately close the current record using this form.

The Blank Entry Form

In order for this trick to work, each file that displays records using output forms must have its own blank entry form. This form should have a single object on it: a no-action highlight button.This button should have a script that contains the single command Cancel. The button should be set to run on the On Load and the On Clicked phases of the entry screen.

You must set the blank layout to be the input form as the last command in your code in the On Open Detail section. 4D will then open the current record in this form. The form will immediately cancel itself and the user will return to the output list.

It is important that there be no unnecessary objects on the blank form in order that its appearance on screen be so brief as to be unnoticeable.

(Form event=On Activate) or (Form event=On Outside Call)

These form events are triggered by an explicit call to this process from another process, or by bringing the window to the foreground. This phase calls the _SYLayoutPhase method. This method checks if any messages have been sent from other processes.

You *MUST* include a call to the _SYLayoutPhase method in *ALL* form methods that correspond to forms displayed on screen. This ensures that the 4Q Shell message passing system will function reliably. You do not need to include calls to _SYLayoutPhase in printing, import and export forms.

4Q Shell's message-passing function is important for two reasons:

- it makes message passing easy
- it is used by the _SYQuitLoop method, which is the method 4Q Shell uses to quit the application.

If the message system is not working correctly, then the user may not be able to quit the application. Since it is so important that message passing be supported, this requirement is worth repeating:

You must include a test for ((Form event=On Activate) | (Form event=On Outside Call)) in all form methods. When either of these events occurs you must call the _SYLayoutPhase method.

_SYLayoutPhase Method

This method can be treated like a black box. You should not make any changes to it or its contents. All you need to know is that when a message is sent from another process (on the same machine) the message will be placed in the variable vSYMsgText. You can check the value of this variable if you need to respond to a message.

The _SYLayoutPhase method takes 5 parameters, with the 5th parameter being optional. It returns an error code.

```
_SYLayoutPhase Method
$1=Name of the calling method
$2=action code (currently unused, pass a blank string)
$3=string value giving the form type: “input”, “output”, “dialog”
$4=pointer to the form's file
$5=pointer to a text variable where the contents of any received
        interprocess message will be placed (optional parameter)
$0=error code, 0 if all OK, 1 if the CANCEL command has been issued,
        or =2 if the user has confirmed that data should be saved.
```

In most cases all you need to do is call the _SYLayoutPhase method when Form Event equals either On Outside Call or On Activate. Pass the string indicating the type of form you are in along with a pointer to the current table.

The _SYLayoutPhase method is described in more detail in the section of Chapter 3 entitled “_SYLayoutPhase Method” on page 27.

Menus, Functions, Buttons & Objects

The above variables are used in 4Q Shell methods, are passed as parameters to 4Q Shell methods, and should be used in certain methods that you create to work in conjunction with 4Q Shell.

On the footer of the list form there is a series of buttons and popups using these variables. In some cases you can use the 4Q Shell methods without writing any of your own code. In most cases you should write a method that is called in the object methods. Your method will handle these variables, and will make the necessary calls to 4Q Shell methods. Each of these footer objects is described below.

Output lists are displayed with a corresponding “list menu”. List menus have a File and a Select menu bar. The action of the items on the Select menu bar is relevant only to list forms. When the user chooses and item on the Select menu it triggers buttons on the layout itself. In this way the actions performed remain under the control of the form itself. This is discussed in detail below.

List Menu Bar

The list menu bars that are supplied with the shell are menu bars 3 and 4. These menu bars are installed by the method _SYEnablMenu when it is called with the second parameter “LIST_AREA”. Menu bar 3 is installed if the user

is a member of the Design group, menu bar 4 is installed if the user is not a Design group member.

These two menu bars may or may not be different. Even if they are not different I feel it is a good idea to have a menu bar reserved for the Designer as it may be useful for development or testing at some future time.

The _SYEnablMenu is called before the window is opened in which the list is displayed. In the case of the address book example, this method call occurs in the _SHAdressUserArea method. The _SHAdressUserArea method controls the Address user area.

Activating the Menu Bar

In order for the installed menu to be active when a form is opened in this manner you must assign a negative menu bar number as the form's associated menu. In 4D v6.0 and earlier it was possible to specify the negative value of a nonexistent menu. If you left the form's associated menu bar setting at the default value of zero, then the menu would appear to be active, but selecting menu items would not run the menu item methods.

In 4D v6.5 assigning a negative, nonexistent menu bar no longer activates the currently installed menu. Instead you must use the negative of an actually existing menu. If this menu has menu items, then these items are appended to the currently installed menu.

In order to activate the currently installed menu you must assign as an associated menu the negative value of a menu bar that has no items. Menu bar #5 was created for just this purpose.

You must assign the value "-5" as the associated menu bar for any list screen which is to interact with the installed list menu bars.

File Menu

This menu triggers the following system-related actions:

- **Page Setup**: opens 4D's page setup dialog box.
- **Show Control Screen**: opens the control screen, if it is not already open, and brings it to the foreground.
- **Quit**: quits the application.

These items call their own methods and these methods behave the same way for all lists.

Select Menu

This menu triggers the following table-specific actions:

- **Search by Example**: opens this particular tables basic search dialog.
- **Show All**: displays all active, or all records, depending on how it is defined to behave in each form.
- **Focus**: limits the current selection to the currently highlighted records.
- **Sort**: opens the system's Sort dialog with sort choices configured for the particular table being viewed.
- **Sets**: opens the system's Set Management dialog to save, restore, or deleted a set of records on disk.
- **Report...**: opens the system's Report and Media dialog a list of available reports that is specific to the table being displayed.

These items do not call their own methods. The methods related to these menu items are called from scripts in buttons on the form itself. This is done so that there is not general menu item code that would steal control of record processing away from the form itself.

These menu items do call menu item methods, but all these methods do is to issue the PostKey event that triggers one of the on or off-screen buttons on the form.

Because the list menu is shared by all list forms, and because the list menu issues specific keyboard equivalents, therefore all list forms must have on- or off-screen buttons to match these menu items. These buttons that correspond to the menu items listed above must be associated with the keyboard equivalents that these menu items send to the form.

The table below lists the menu items on the Select menu, the menu item methods they correspond to, and the keyboard equivalents that each menu item method sends to the form.

TABLE 4. The Application-Wide Behavior of the Select Menu

Select Menu Item	Menu Item Method	Keyboard Event
Search by Example	_SYMBSchByExample	Ctrl (or Cmd) + E
Show All	_SYMBShowAll	Ctrl (or Cmd) + L
Focus	_SYMBFocus	Ctrl (or Cmd) + F
Sort	_SYMBSort	Ctrl (or Cmd) + S
Sets	_SYMBSets	Ctrl (or Cmd) + U
Report…	_SYMBReports	Ctrl (or Cmd) + R

In the following sections on the Search Popup (page 46) and Focus Popup (page 55) you will see that these popup menus control the first three actions listed on the Select menu.

Because these actions are controlled by popup menus, rather than buttons, and because popup menus do not support keyboard equivalents, it is necessary to add off-screen buttons that handle the user's selection of these first three menu items.

These off-screen buttons basically do nothing more than call the appropriate items on the Search and the Focus popups. The mechanism for handling this is discussed in the following sections dealing with the Search and the Focus popup.

Mini-sort & Search Buttons

The little squares and triangles below certain columns on the output layouts are the mini-search and mini-sort buttons. Pressing these buttons either brings up a dialog for specifying criteria to match with the contents of a particular displayed column, or it sorts the information in the displayed column in ascending (triangle up) or descending (triangle down) order.

Mini-Sort

The code behind the sort buttons changes depending on the column they're associated with, and on whether the sort is ascending or descending. For the ascending sort on the [Address]Company field the code is:

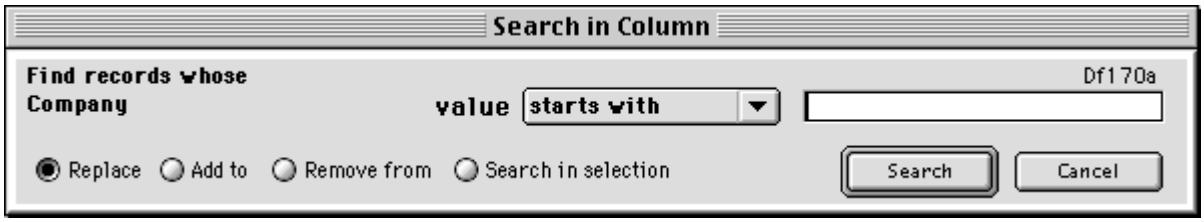
```
$CallerName:="SC-[Address];'qAddress_o':vhbna4"
<>vARSrt_A:=->[Address]Company
<>vARSrt_B:=<>vSYNilPtr
<>vARSrt_C:=<>vSYNilPtr
<>vARSrt1UP:=True
$Err:=_SYSrtManyLevl ($CallerName;->[Address];-><>vARSrt_A;
      -><>vARSrt_B;-><>vARSrt_C;-><>vARSrt1UP;
      -><>vARSrt2UP;-><>vARSrt3UP)
```

The code for the descending sort button on the same column would be the same as that shown above, except that the variable <>vARSrt1UP would be set to "false".

To adopt these buttons for your own tables you need to define and use the sort variables mentioned in the section entitled "Global Variables" on page 24. Replace the references to the [Address]Company field with references to the field you want to sort on. Replace the <>vARSrt_A through C and <>vARSrt1UP through <>vARSrt3UP variables with those that apply to your table.

Mini-Search

The mini-search button displays a dialog for the user to enter a variety of search criteria.



The code behind the mini-search buttons involves calls to three methods: one to search, one to resort, and one to reenable the menu items. The object method of the [Address]Company mini-search button is:

```
$CallerName:="SC-[Address];'qAddress_o':vhbna1"
$Err:=_SYSchInColumn ($CallerName;"";->[Address]Company;
        "Company";<>vARSet)
If ($Err>0)
  $Err:=_SYSrtManyLevl ($CallerName;->[Address];-><>vARSrt_A;
        -><>vARSrt_B;-><>vARSrt_C;-><>vARSrt1UP;
        -><>vARSrt2UP; -><>vARSrt3UP)
End if
$Err:=_SYEnablMenu ($CallerName;"LIST_AREA")
```

To adopt this code to provide mini-search functions for the [Sample]Name table, for example, you would provide a mini-search button below the Name column on the Sample list form and include the following object method:

```
$CallerName:="SC-[Sample];'qSample_o':vhbna1"
$Err:=_SYSchInColumn ($CallerName;"";->[Sample]Name;
        "Name";<>SampleSet)
If ($Err>0)
  $Err:=_SYSrtManyLevl ($CallerName;->[Sample];
        -><>SampleSrt_A;-><>SampleSrt_B;
        -><>SampleSrt_C;-><>SampleSrt1UP;
        -><>SampleSrt2UP; -><>SampleSrt3UP)
End if
$Err:=_SYEnablMenu ($CallerName;"LIST_AREA")
```

The call to _SYSchInColumn will display the mini-search dialog shown above. You can pass a pointer to a field of type string, integer, longint, real, date, boolean, or time. You can't search on BLOBs, pointers, text, or subtable fields.

If the search is performed, the _SYSchInColumn method returns the value "1". If the search is canceled it returns "0".

When the search is performed you'll need to resort the selection to regain the order previously selected by the user. This is done with the call to _SYSrtManyLevl.

Finally, because the mini-search dialog has disabled the menu items, you need to reenable the menu items. This is done with the final call to _SYEnableMenu.

Search Popup

You must write a search handling method

This popup uses the <>arSampleSchNam text array described in the section "Global Variables" on page 24. The popup displays a list of search alternatives. Each alternative can either perform a search, or can open a dialog for the user to provide further specifications.

Since the system installs a Select menu for every list form, and the Select menu offers Search by Example and Show All, among other items, your Search Popup should offer these two options as well.

While this is not strictly necessary, the search popup is the place where all query options should be placed. Consequently it is good interface design to reiterate these options on the Search popup.

As you will see in the following discussion of the Search popup method, the Search by Example and Show All actions will be controlled through your custom search popup method.

The search popup is controlled by a method that you write. You should write a different search popup method for each of the different tables that will support searching. This method's code structure is the same in all cases and can be copied from the _SHAddrHdlSchPop provided with 4Q Shell.

In the following example I'll name the method SampleHdlSchPop. The method performs either of two actions. The action it performs depends on the value of the action code passed as the second parameter.

Initializing the Popup

The first action it performs is to initialize the popup. This occurs when the method is called with the second parameter set to "Initialize". Initializing means giving the search popup text array a number of elements and populating those elements with labels that indicate search options. It is your responsibility to write the SampleHdlSchPop method so that it responds to the "Initialize" value in the second parameter.

Call this method with $2= “Initialize” in the On Load phase of the list form. The syntax of this call would be:

```
$Err:=_SHAddrHdlSchPop ($CallerName;"INITIALIZE";<>SampleSet;~
->><>arSampleSchNam;"")
```

Performing the Search

The second action the method performs is to act on a user’s selection from the popup menu. This is indicated by passing the value “Do_Popup” as the second parameter. It is your responsibility to make sure the method responds properly to each of the possible selections in the search array that the user can make.

In the example below the only additional method that you must write is *SampleSch*, whose function is to respond to the user’s choice of the second search option.

Call the SampleHdlSchPop method with the “Do_Popup” parameter from the popup array’s object method. The popup array will appear in the footer of the output form and the syntax of this call will be:

```
$Err:=SampleHdlSchPop ("SC-[Sample];'qSample_o':<>arSample-
SchNam";"DO_POPUP";<>SampleSet; Self;"qSample_i")
```

Search Handling Method Parameters

The search handling method has 5 parameters. In addition to a second parameter that tells the method whether to initialize or to perform a search, four other parameters need to be passed. These are as follows:

SampleHdlSchPop Method

$1= name of calling method (string 80)

$2= action code (text values “Initialize” or “Do_Popup”)

$3= name of set where search results are to be placed

$4= pointer to the search popup text array, in this case <>arSample-SchNam

$5= name of the input layout to restore after the search is performed

$0= error code, returns >=0 if operations are normal, <0 if there’s a programming error.

Basic Format of Your Search Handling Method

You can add as many or as few search items to your search popup as you like. These items will trigger additional searches or search methods that can do anything you want.

In order for this method to support the Search by Example and the Show all options offered on the Select menu it is necessary to provide at least these two options.

In addition, the method I've used to the items on the Select menu is for the form to set the Search popup to the element that corresponds to the appropriate action, and then call the Search popup method to execute this action.

In the following example code Search by Example is offered as the first item on the popup menu, and Show All is offered as the second. When the user chooses Search by Example from the Select menu the form sets the popup menu variable to the value "1" and then calls the search method. When the user chooses Show All from the Select menu the form sets the popu menu variable to the value "2" and calls the search menu.

Notice that the popup menu need not have these items in these popup array elements. They could be located in other elements. Everything that determines which element corresponds to which action , and that subsequently handles these actions is managed from within the search method itself.

For consistency throughout your application I suggest that you do include a Search by Example and a Show All option on all of your Search popup menus. I also suggest that these two options be displayed in the same place on all list forms. In the 4th Quarter Accounting application we place Search by Example as the first, and Show All as the last options on each search popup.

The following format of this procedure is taken from _SHAddrHdlSchPop method. The essential point is that you can support whatever search routines you care to create.

```
$Err:=0
$SetName:=$3    `This is the set name stored in <>SampleSet.
$arSchOptns_:=$4
$InputLayout:=$5
$SearchDone:=0
Case of
  : ($2="INITIALIZE")
`Set array size, assign names of searches to appear in search popup.
    $Size:=2        `Set this to eqaul the number of choices you'll offer
    ARRAY TEXT($arSchOptns_->;$Size)
    $arSchOptns_->{0}:="Search..."
    $arSchOptns_->{1}:="Example"
    $arSchOptns_->{2}:="All Records"
`... assign additional choices here.
    $arSchOptns_->:=0    `set to display the "Search" text

  : ($2="ALL")
```

```
   $arSchOptns_->:=2

: ($2="EXAMPLE")
   $arSchOptns_->:=1

: ($2="DO_POPUP")
   $ChosnOption:=$arSchOptns_->
   Case of
    : ($ChosnOption=1)
`You must write the following search method yourself.
`It can take any parameters you want.
      $SearchDone:=SampleSch ($CallerName;"BY_EXAMPLE";~
                  $SetName;$InputLayout)
       $Err:=_SYEnablMenu ($CallerName;"LIST_AREA")
: ($ChosnOption=2)
      ALL RECORDS([Sample])
      $SearchDone:=1
   End case

If ($SearchDone=1)    `this indicates the selection has changed and
                       needs to be resorted.
     vAskdDblSRT:=False    `This signals the user should be asked to
                       confirm double sorting when specified.
     CREATE SET([Sample];$SetName)
     $Err:=_SYSrtManyLevl ($CallerName;->[Address];~
       -><>SampleSrt_A ;-><>SampleSrt_B;-><>SampleSrt_C;~
       -><>SampleSrt1UP;-><>SampleSrt2UP;-><>SampleSrt3UP)
   End if    `($SearchDone=1)
  End if    ` ($arSchOptns_»>0)
  $arSchOptns_->:=0


Else        `This occurs if you make a mistake and pass a bad value.
 $Msg:="Programming error: incorrect parameter passed to "~
          +$CallerName+" by "+$1
_SYAlert ($CallerName;$Msg)
  $Err:=-1
End case
$0:=$Err
```

Linking the Popup Menu with the Select Menu

In order to have the Search by Example and Show All methods triggered by either the search popup, and the corresponding items on the Select menu, you can place two off-screen buttons below the footer area of your list form.

These buttons must be associated given keyboard equivalents that match those issued by the items on the Select menu. In the case of Search by Example the Control+"U" (on the PC) or Command+"U" (on the Mac) are used. For Show All the Control+ or Command+"L" keyboard events are used.

The scripts that you place in these buttons make two calls to the search method. The first call sets the search popup menu to the appropriate element for the corresponding action. The second call tells the method to perform the action.

In order for the search method to support these two calls it must be written to that end. The example given above shows how this is done. Basically the search method contains a large Case Of statement which tests the value of the action parameter passed to it in $2.

When the action parameter has the value "EXAMPLE", the method sets the popup to that element associated with the Search by Example function. When the action parameter of "ALL" is passed, the method sets the search popup that that element. In the example above element 1 is used for Search by Example, and element 2 is used for Show All.

Once the search popup has been set to the appropriate element it is called again just as it is when the user actually chooses an item from the popup meuu. This is handled by passing the action parameter "DO_POPUP", as shown in the above sample code.

The actual contents of these two off-screen buttons on the qAddress_o form is given below.

```
$CallerName:="SC-[Address];'qAddress_o':
            <>varARSchNam"
C_LONGINT($Err)
$Err:=_SHAddrHdlSchPop ($CallerName;"EXAMPLE";
            <>vARSet;->◊ varARSchNam;"qAddress_i")
$Err:=_SHAddrHdlSchPop ($CallerName;"DO_POPUP";
            <>vARSet;->◊ varARSchNam;"qAddress_i")
```

and

```
$CallerName:="SC-[Address];'qAddress_o':vbnaAll"
C_LONGINT($Err)
```

```
$Err:=_SHAddrHdlSchPop ($CallerName;"ALL";
        ◊ vARSet;->◊ varARSchNam;"qAddress_i")
$Err:=_SHAddrHdlSchPop ($CallerName;"DO_POPUP";
        ◊ vARSet;->◊ varARSchNam;"qAddress_i")
```

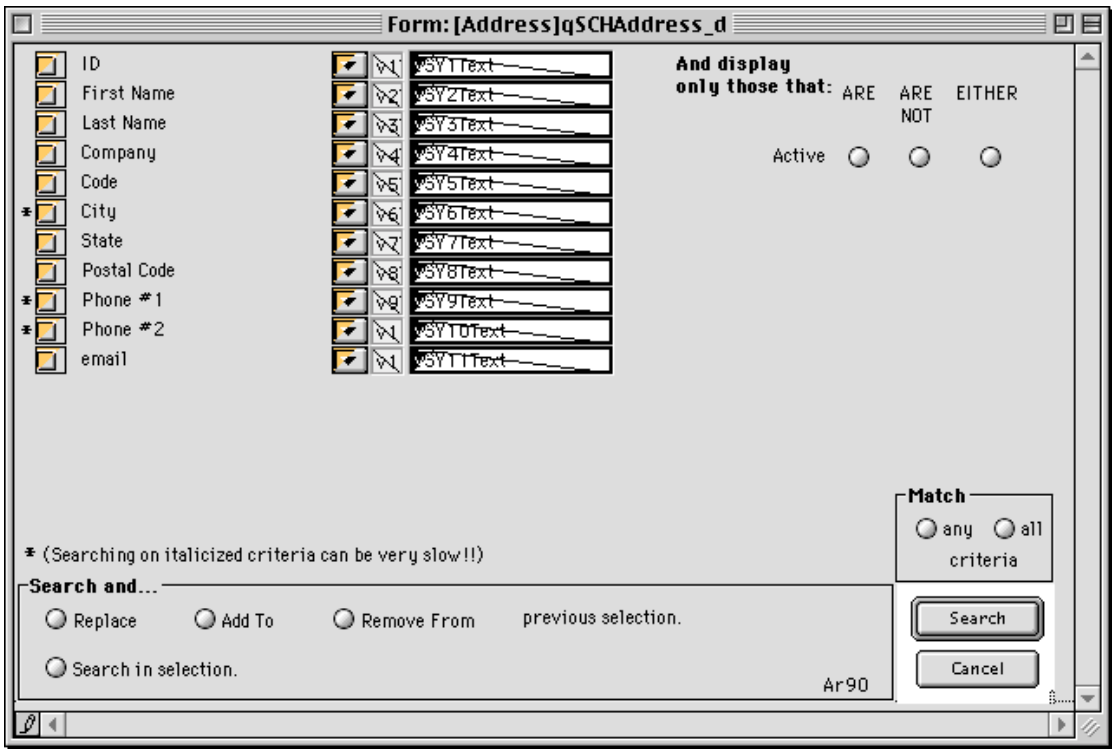
Search Entry Screen

The search popup can be written to bring up any entry screen for creating and processing a search. The shell contains a sample search on the Address table and you can adapt this entry screen for your own purposes.

Once the user selects the search labeled “Example” in the popup, the method _SHAddressSch is called. This opens the form [Address]; qSCHAddress_d in a separate window using the Dialog command.

```
$WinNum:=_SYOpnWindow ("";<>vSYStdWin_W;<>vSYStdWin_H;
      Movable dialog box;"";"_SYCancelBox")
DIALOG([Address];"qSCHAddress_d")
_SYClosWindow
```

The interprocess variables <>vSYStdWin_W are used to size the new window to a standard height and width <>vSYStdWin_H. These variables are assigned values when the application is initialized.



This screen is not as flexible as 4D's Query Editor, but it is easier to use. It has many generic elements that can be adapted to searches performed on other tables and fields.

The screen functions by recording the user's search criteria in the text fields. All search criteria are entered in text variables because it is easier to control 4D entry and formatting filters for string values than it is for other types of variables. The values entered will later be converted to the type that matches the field being searched.

Completing the search specifications requires the user set the following values:

- Check the fields to be include in the search using the check boxes on the left hand side of the screen.
- Use the popup menus to the left of each entry field to indicate the type of search (greater than, less than, equals, etc.)
- Enter a matching value in one or more of the search specification entry areas. These areas have individual entry and display screens that match the type of information allowed. For example, dates and phone numbers only allow numerals and separators. For string fields the entry filters allow the user to append the "at" sign as a wildcard.
- The three-way radio buttons in the upper right apply to fields that store boolean values. The three choices are "true", "false", and "either". You can add additional radio buttons for additional boolean fields if necessary.
- Use the radio buttons in the "Match" box on the lower right to indicate whether multiple criteria should inclusive (matching every value entered) or alternate (matching at least one value entered).
- In the "Search and…" box at the bottom indicate how to display the located items with respect to those records already being displayed in the selection.

The structure of the search specification screen must precisely match the search code that lies behind the screen. That is, if you add a field, or change the type of a field, then you must change the code in the search method accordingly.

Search Method

The qSCHAddress_d form is used to collect the users search specifications. This form performs entry filtering and checks that the user has entered acceptible information. It does not perform the search.The search itself is performed in the code of the _SHAddressSch method.

The _SHAddressSch method consists of a For loop that builds a custom search according to the user's specifications. This For loop contains a Case Of statement with a separate case for each of the possible search criteria. It has the following format:

```
For ($j;1;$NumFields)
      `loop through each of the search variables that may be involved.
      $SchPrefix:=""
      `search is only done if this prefix is assigned one of the
      `meaningful values.
      Case of
        : ($j=1)
         $pField:=->[Address]Address_ID
         $pSchValu:=->vSY1Text
          $SchPrefix:=v1text*Num(vcbSY1=1)
        : ($j=2)
         $pField:=->[Address]First_Name
         $pSchValu:=->vSY2Text
          $SchPrefix:=v2text*Num(vcbSY2=1)
…<etc.>…


End case

     If ($SchPrefix # "")
      $SchCount:=$SchCount+1
       Case of
        : ($SchCount=1)
      `the first search must be done without a preceeding operator
          QUERY([Address];$pField->;$SchPrefix;$pSchValu->;*)
        : ($FindAny)
          QUERY([Address]; | ;$pField->;$SchPrefix;
            $pSchValu->;*)
         Else
            `criteria must be inclusive: records must meet
            `all criteria.
          QUERY([Address]; & ;$pField->;$SchPrefix;
            $pSchValu->;*)
       End case
     End if
   End for
```

The For loop executes once for each possible search criteria. Each criteria is handled in a separate in case. If the user has not check the corresponding check box for that search, then a blank value will be assigned to the variable $SchPrefix and this criteria will not be used.

When the user checks a particular search and enters a search value, the variable in which their criteria has been entered is assigned to the local pointer $pSchValu:

$pSchValu:=->vSY1Text

Below the Case Of statement a search is built using the Query command and a series of pointers to the field, comparator, and search value. Notice that the search value is $pSchValu in every case, and this variable points to one of the text variables listed on the Search form.

In the current version of 4D you can search various types of fields using specifications stored in a text variable. 4D converts the value in the text variable appropriately. For example, when searching on the AddressID field which stores longint values, we can indicate the values we want to locate by placing their ID's in the text field vSY1Text. The query will convert the text in vSY1Text into a longint values and locate the correct values.

After the For loop the search code handles the boolean fields specified by the three radio buttons. This code always executes when either the "are" or "are not" radio button is set. If the "either" option is set, then the variable cvrb3 will have the value 1 and this code is skipped.

```
If (cvrb3 # 1)   `inactive accounts
      $SchCount:=$SchCount+1
       $ActivityValue:=cvrb1
      Case of
        : ($SchCount=1)
       `the first search must be done without a preceeding operator
          QUERY([Address];[Address]ActiveKeyValue=
              $ActivityValue;*)

        : ($FindAny)
          QUERY([Address]; | ;[Address]ActiveKeyValue=
              $ActivityValue;*)

        Else
       `criteria must be inclusive: records must meet all criteria.
          QUERY([Address]; & ;[Address]ActiveKeyValue=
              $ActivityValue;*)
      End case
    End if
```

This search is being done on the ActiveKeyValue field. This field is a longint and has the values "1" when the record is active, and "0" when it's considered inactive.

The Case Of statement tests whether this is the first search being done, then the query does not begin with a concatenation operator such as & or |. If this is not the first search, then either a & or | is used according to the users settings.

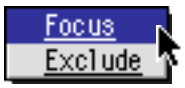
At the end of the _SHAddressSch method the Query command issued to tell 4D to perform the built search. The results of the search are then place in sets . 4D set commands are used to either added to, removed from, located in, or used to replace the previous selection fo address records.

```
If ((Serr=0) & ($SchCount>0))
   MESSAGES ON
   QUERY([Address])   `this command performs the built search.
   MESSAGES OFF

   CREATE SET([Address];"LastVendor")
   Case of
     : (avrb2_d=1)
      UNION("LastVendor";$ResultSet;"LastVendor")
     : (avrb3_d=1)
      DIFFERENCE($ResultSet;"LastVendor";"LastVendor")
     : (avrb4_d=1)
      INTERSECTION($ResultSet;"LastVendor";"LastVendor")
     : (avrb1_d=1)
   End case

   USE SET("LastVendor")
   CLEAR SET("LastVendor")
   $Err:=1
 End if
```

Always remember to clear your temporary sets after you're finished with them. Otherwise they will remain in memory, taking up otherwise useful space, until the process is terminated.

Focus Popup

The Focus popup provides the options "Focus" and "Exclude". Both of these act on the currently highlighted records. The first reduces the current selection to those items highlighted; the second removes the highlighted records from the current selection.



Both of these operations preserve the current sort order of the displayed records. That is, if the records are sorted on 1, 2 or 3 levels, then after the indicated records have been excluded or focused, upon the remaining records will be resorted in the same order.

The focus function operates using a single call that is placed in two active objects. One object is the varFocusOn text array that is assigned to a Menu/ Drop-down List object on the layout. The method for this object is:

```
_SYDoFocusPop ($CallerName;->[Sample]; "Samples";~
<>SampleSet; Self;<>SampleSrt_A; <>SampleSrt_B;
<>SampleSrt_C; <>SampleSrt1UP; <>SampleSrt2UP;
<>SampleSrt3UP)
```

The Focus action is linked to the key equivalent Control-F (or Command-F on the Mac). This is also the keyboard equivalent that is issued by the Select menu when the **Focus** item is selected. Choosing this item from the Select menu does nothing more than issue this keyboard equivalent. The actual focusing is done by the same code that lies behind the **Focus** popup menu.

In order to support this equivalence we have placed a button on the form that is set to respond to this key stroke. This is the off-screen button assigned the variable vbna12. This is a no-action button.

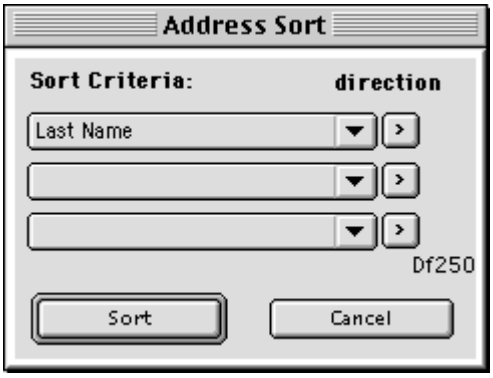
This off-screen button has the following object method:

```
_SYDoFocusPop ($CallerName;->[Sample]; "Samples"; <>Sam-
pleSet;->varFocusOn; <>SampleSrt_A; <>SampleSrt_B;
<>SampleSrt_C; <>SampleSrt1UP; <>SampleSrt2UP;
<>SampleSrt3UP; "FOCUS")
```

As long as you assign the correct values to the variables passed in these calls, then the Focus popup will perform as a black box.

Sort Button

The sort button calls the _SHSRTAddress method. This method is particular to the Address table and contains calls to the generic routines for interacting with the user and maintaining sort order. Pressing the Sort button brings up the Sort dialog.



The sort button is linked to the key equivalent Control-S (or Command-S on the Mac). This is also the keyboard equivalent that is issued by the Select menu when the **Sort** item is selected. Choosing this item from the Select menu does nothing more than issue this keyboard equivalent. Control of all aspects of the sort operation is done by the same code that lies behind the **Sort** button.

Rather than simply describe the code in this routine I'll describe code you would write if you wanted to handle sorting for your own table.

Let's say your Sample table has four fields that you want to allow the user to sort on. Say that these are the ID, Name, Type, and Date fields.

You create a new method that you call SampleSRT. You call this method from the Sort button, which you place in the footer of your Sample list form. This method would appear as follows:

SampleSRT Method

```
$CallerName:="SampleSRT"
   `$1=Name of the calling method;
   `$0=error code, 0 if all OK.
C_STRING(80;$1)
C_LONGINT($0;$Err)
$Err:=0

ARRAY TEXT(varSRTName1;4)
ARRAY POINTER(varSRTPtr1;4)
varSRTName1{1}:="ID"
```

```
varSRTName1{2}:="Name"
varSRTName1{3}:="Type"
varSRTName1{4}:="Date"

varSRTPtr1{1}:=->[Sample]ID
varSRTPtr1{2}:=->[Sample]Name
varSRTPtr1{3}:=->[Sample]Type
varSRTPtr1{4}:=->[Sample]Date

$Err:=_SYGenSRT_d ($CallerName;"AskSort";->[Sample];
            ->><>SampleSrt_A; -><>SampleSrt_B;-><>SampleSrt_C;
            -><>SampleSrt1UP;-><>SampleSrt2UP;
            -><>SampleSrt3UP;"Sample Sort")
$Err:=_SYEnablMenu ($CallerName;"LIST_AREA")
$0:=$Err
```

In this code you make use of the two arrays varSRTName and varSRTPtr1. Both arrays are sized the same. The first is assigned the names of the sort fields that are displayed to the user. The second is assigned pointers to the fields in the Sample file with which these names correspond.

You also need the six sorting variables that are unique to the Sample file, as we have mentioned before. These are the interprocess variables <>SampleSrt_A through C, and <>SampleSrt1UP through 3UP. The first three store the pointers to three sort criteria. The second three store the direction of each of these sorts, ascending or descending.

The _SYGenSRT_d method will display a dialog to the user, assign the selections to the interprocess variables, and perform the sort.

If the user elects to sort on fewer than three criteria, the unused pointers will be assigned the value <>vSYNilPtr. This option is also handled automatically by _SYGenSRT_d.

After the call to _SYGenSRT_d you must call the method _SYEnableMenu with $2=”List_Area”. This will enable the menu bar for the list form, which is disabled by the sort criteria dialog box.

Nil Pointer

If you need to test whether a pointer is set to a valid object, or is set to the value <>vSYNilPtr you can make the call

```
$IsNil := _SYNil($CallerName; <pointer variable>)
```

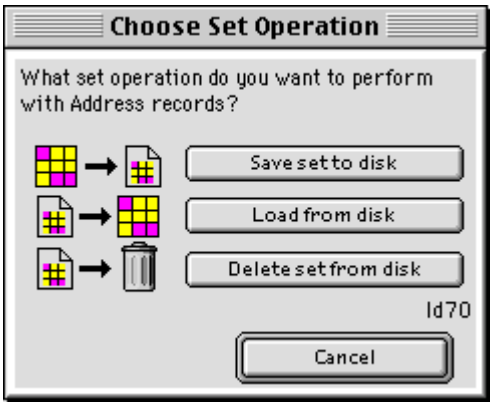
This method returns the value true if the pointer variable has the value <>vSYNilPtr. <>vSYNilPtr is a 4Q Shell system variable that you can assign to an unused pointer variable. Its value should never be changed.

Sets Button

4Q Shell stores sets of selected records in files on disk. These selections are preserved after the user quits the application and may be restored when they log back on at a later time.

The **Sets** button is linked to the key equivalent Control-U (or Command-U on the Mac). This is also the keyboard equivalent that is issued by the Select menu when the **Sets** item is selected. Choosing this item from the Select menu does nothing more than issue this keyboard equivalent. All set management is done by the same code that lies behind the **Sets** button.

Pressing the Sets button brings up the Choose Set dialog.



The sets function involves three methods. Calls to these methods can be placed in the object script of the Sets button itself, as is demonstrated on the Address table list form. The method of this object is:

```
$CallerName:="SC-[Address];'qAddress_o':vbna9_o"
If (1=_SYHdlSet ($CallerName;"SET_DIALOG";->[Address];
                 "Address";<>vARSet))
  $Err:=_SYSrtManyLevl ($CallerName;->[Address];-><>vARSrt_A;
          -><>vARSrt_B;-><>vARSrt_C;->v<>ARSrt1UP;
          -><>vARSrt2UP;-><>vARSrt3UP)
End if
$Err:=_SYEnablMenu ($CallerName;"LIST_AREA")
```

Modifying this code for use with our Sample table involves substituting the Sample table and Sample table variables in place of those related to the Address table. The result of this substitution is:

```
$CallerName:="SC-[Sample];'qSample_o':vbna9_o"
If (1=_SYHdlSet ($CallerName;"SET_DIALOG";->[Sample];
                 "Sample";<>SampleSet))
  $Err:=_SYSrtManyLevl ($CallerName;->[Address];
          -><>SampleSrt_A;-><>SampleSrt_B;-><>SampleSrt_C;
```

```
            ->SampleSrt1UP;->SampleSrt2UP;
            ->SampleSrt3UP)
   End if
   $Err:=_SYEnablMenu ($CallerName;"LIST_AREA")
```

Add Button

The Add button creates a new process that displays a blank address record. The method of the Add button object is:

```
$CallerName:="SC-[Address];'qAddress_o':vbnaAdd"
C_LONGINT($Err)
ARRAY TEXT(var1SYText;1)
var1SYText{1}:=vSYquo+"ENTRY"+vSYquo
$Err:=_SYOpenProcess ($CallerName;"NEW";
      "_SHAdressUserArea"; vSYDfltPrcs;
      "AddressEntry";True;"";0;->var1SYText)
```

This involves a single call to the _SYOpenProcess method. The system array var1SYText is used to pass a parameter to the new process. You can reuse this array when you call this method in other places.

The new process is controlled by the method _SHAdressUserArea. For you to use this method of adding records you must create your own process controlling method. This method has already been discussed in Chapter 3, “Your Process Handling Methods” on page 23.

Modify Button

The modify button contains a simple object method. It does not open a new process for the modification of the selected record. The method for the button is:

```
$CallerName:="SC-[Address];'qAddress_o':vbna11_o"
If (0=_SYModRcrd ($CallerName;->[Address]; "qAddress_i";
      "qAddress_i"))
  $Err:=_SYSrtManyLevl ($CallerName;->[Address];->vARSrt_A;
      ->vARSrt_B;->vARSrt_C;->vARSrt1UP;-
>vARSrt2UP;
      ->vARSrt3UP)
End if
```

The _SYModRcrd method opens the record in the UserSet in the current window using the Modify Record command and the indicated entry form. If the UserSet contained a single recor,d the _SYModRcrd method returns “0”. The current selection is then resorted with a call to _SYSrtManyLevl.

If the UserSet contains no record or more than one record, the _SYModRcrd method returns a negative number.

Delete Button

The _SYCekSet2Delet method checks whether the UserSet contains the specified number of records.

```
$CallerName:="SC-[Address];'qAddress_o':vbnaDelete"
CREATE SET([Address];"TempAddressSet")
$Err:=_SYCekSet2Delet ($CallerName;"Cancel,Only_One";
          "Address")
If ((Err=1))
  $Err:=_SHDeleteAddress ($CallerName;"")
  If ($Err=0)
   USE SET("TempAddressSet")
   $Err:=_SYSrtManyLevl ($CallerName;->[Address];
      -><>vARSrt_A; -><>vARSrt_B;-><>vARSrt_C;-><>ARSrt1UP;
      -><>vARSrt2UP;-><>vARSrt3UP)
  End if
End if
CLEAR SET("TempAddressSet")
```

In this instance the string “Only_One” is passed in $2 to indicate to this method that only one record can be selected at a time. When it does, the user is asked to confirm the deletion before the record is deleted.

If the UserSet passes this test and the user confirms the desire to delete the record the _SYCekSet2Delet method returns the value 1. Otherwise it returns zero.

The method _SHDeleteAddress is then called to delete the record. To use this template for your own tables you will need to write your own record deletion method. This could be as simple as a call to the 4D function Delete Record. More commonly, however, there will be related records that may need to be checked, unlinked, or deleted.

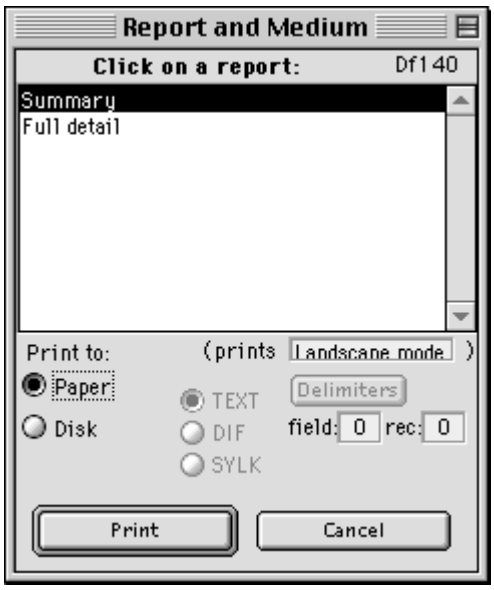
If the record is deleted, the _SHDeleteAddress returns zero and the selection is resorted.

Reports Button

The reports button opens a dialog that presents the user with a variety of reports that are specific to the table being viewed. You can indicate the options associated with each report.

The sort button is linked to the key equivalent Control-P (or Command-P on the Mac). This is also the keyboard equivalent that is issued by the Select menu when the **Report...** item is selected. Choosing this item from the Select

menu does nothing more than issue this keyboard equivalent. Handling of reports is done by the same code that lies behind the **Report...** button.



The report button method contains a call to the address printing routine and a call to reenable the menu items after the Report dialog is closed.

```
$CallerName:="SC-[Address];'qAddress_o':vbna7_o"
C_LONGINT($Err)
_SHPRTAddress ($CallerName;<>vARSet)
$Err:=_SYEnablMenu ($CallerName;"LIST_AREA")
```

The _SHPRTAddress method, like the _SHSRTAddress discussed above, is a method that you create to manage the generic report routines provided by 4Q Shell.

_SHPRTAddress Method

```
$CallerName:="_SHPRTAddress"
   `$1=Name of the calling method;
   `$2=name of set holding current selection
   `$0=error code;
   `To reuse this as generic code you must create the print and disk
   `layouts whose names are passed to the following procedures.
C_BOOLEAN($4QS980825)
C_STRING(80;$1)
C_TEXT($2)
C_LONGINT($0;$Err)
$Err:=0
Case of
```

```
    : (0>_SYSetPrtSpecs ($CallerName;"INITIALIZE"))
        `error in initializing.
    : (0>_SYSetPrtSpecs ($CallerName;"ADD_ELEMENT";
        "Summary";"Landscape mode";"Both";"qPRT1Address_o";
        "qDISK1Address_o";False))
        `error in setting options.
    : (0>_SYSetPrtSpecs ($CallerName;"ADD_ELEMENT";"Full detail";
        "Portrait mode";"Both";"qPRT2Address_o";
        "qDISK2Address_o";False))
  Else    `no errors yet.

    $ItmSelected:=_SYSetPrtSpecs ($CallerName;"HANDLE_LIST";
          "1"; String(Table(->[Address])))
    $DiskFormat:=("TEXT"*cvrbToText)+("DIF"*cvrbToDif)
          +("SYLK"*cvrbToSYLK)
    READ WRITE([Address])
    OUTPUT FORM([Address];"qAddress_o")
    Case of       `if item selected requires custom processing.
                  `handle custom report printing here.
      : ($ItmSelected=3)`...whatever,
      : ($ItmSelected=4)
                        ` etc..
    End case
  $Err:=_SYSetPrtSpecs ($CallerName;"CLEAR")
  End case
  $0:=$Err
```

The structure of this method involves the use of a Case Of statement that makes repeated calls to the _SYSelPrtSpecs method. The _SYSelPrtSpecs method behaves differently depending on the value passed in the second parameter.

The first call to _SYSelPrtSpecs must pass the value $2=”INITIALIZE”. This resizes all arrays to zero length and resets other variables that will be needed to handle the report dialog.

Additional calls to _SYSelPrtSpecs method with $2= “ADD_ELEMENT” then place values in the report arrays. Every call to _SYSelPrtSpecs with $2=”ADD_ELEMENT” adds another element to each array used by 4Q Shell’s report handler. Repeated calls made in this fashion fill the report arrays so that the full list of reports is shown to the user.

If these calls are successful, _SYSelPrtSpecs returns the value zero. A negative value is returned if an error occurs. The Case Of statement continues issuing

calls to _SYSelPrtSpecs until either a negative value is returned, indicating an error, or until it reaches the Else statement.

The final call to _SYSelPrtSpecs is made after the Else statement. Here the method is passed $2= “HANDLE_LIST”. This displays the report dialog and processes the user’s report selection.

To use _SYSelPrtSpecs you must know what other parameters this method requires. These parameters are:

_SYSetPrtSpecs Method

$1=Name of calling method

$2=action to take: INITIALIZE, CLEAR, ADD_ELEMENT, HANDLE_LIST

$3=action code: If $2= “ADD_ELEMENT” then $3 is the name of the report shown to users. If $2= “HANDLE_LIST” then $3 is a string value giving the array element to be selected by default.

$4=action code: If $2=”ADD_ELEMENT” this is the report orientation (usu. landscape mode or portrait mode). If $2= “HANDLE_LIST” then $4 is a string giving number of file that is to be printed.

$5= media options={“Paper”; “Disk”; “Both”, “Neither”};

If $5= “Neither”, then you must determine appropriate action in the calling method.

$6=name of layout used for printing using PRINT SELECTION, blank name indicates nonstandard printing. In this case the procedure will not execute printing, and you must handle it yourself in the calling procedure.

$7=name of layout used for printing using one of the EXPORT… commands; blank name indicates nonstandard printing. In this case the procedure will not execute printing, and you must handle it yourself in the calling procedure.

$8=boolean indicating if each report is TEXT ONLY to disk (T or F)

$0=longint error code:

> 0 gives the line of the selected report;

=0 means user canceled the report dialog;

< 0 if error.

If you are printing reports using 4D’s standard Print Selection command, you can pass enough information to _SYSelPrtSpecs so that it can completely handle the selection and printing of the report.

If you support exporting information to disk using the Export Text, Export Dif and Export Sylk commands, then these too can be completely handled by _SYSelPrtSpecs.

If you are printing a report by some other means, or printing a report that needs additional specifications, then _SYSelPrtSpecs can be used to query the user and to return the user's selection. You must handle the actual printing.

In the case where _SYSelPrtSpecs does not perform the printing, you specify blank values in $6 and $7 when you make the call to add the report element. The user will see the name of the report, as indicated in $3, and will be able to select it.

For example, say I wanted to add a third Address table report that did some complex printing. I want this report option shown to the user under the name "Special Report". To do this I would add a call in the Case Of statement in the form:

```
: (0>_SYSetPrtSpecs ($CallerName;"ADD_ELEMENT";
                     "Special Report";"";"Neither";"";"";False))
```

If the user selected this report and pressed the Print button, the _SYSelPrtSpecs method would pass back the line number corresponding to the report selected. That is, it would return the number of the line on which the displayed report appeared in the Report dialog. Ît would not attempt to print anything. Printing would be done from custom code added to the print handling method.

Referring to the code of the _SHPrtAddress method, if the user selected this "Special Report", then the _SYSelPrtSpecs would return the value 3. After the call to _SYSelPrtSpecs with $2= "HANDLE_LIST", I would add a test for a return value of 3. When this occurred I would call a new method, which I'll call _SHPrtSpecialAddress.

```
Case of          `if item selected requires custom processing.
                 `handle custom report printing here.
   : ($ItmSelected=3)
     _SHPrtSpecialAddress ($CallerName)
   End case
```

After the report has been printed I make a call to _SHPrtAddress with $2="CLEAR". This resets all the report arrays to zero elements.

Select Button

The Select button allows the user to indicate one or more records by high-lighting them. The select button is only appropriate in those circumstances where the user is performing an action that requires records to be selected. In the address table example in 4Q Shell there is no need for this action, so the select button is not displayed.

To disable or enable the select button, you set the vSelect variable to true or false before entering the list form. This variable is tested in the On Load phase of the form method and the Select button is controlled with a call to

```
Set Visible (vbnaSelect; vSelect)
```

When the select button is visible it performs its actions with a one-line object method.

```
_SYCekSelUsrSet ($CallerName;True;"address")
```

The second parameter, given as true here, indicates whether the user can select only a single record. If the value false is passed, then the user can select multiple records.

The third parameter is a string used in communicating with the user. In this case the word “address” is passed so that if the selection isn’t satisfactory the user can be told how many “address” records they can select.

If the user’s selection is acceptable, the _SYCekSelUsrSet method issues the Accept command and returns the value 1.

Return Button

Return is simply a cancel button set to be the default button. This button responds to three different keyboard equivalents.

- Since it is the default, it is automatically triggered by the enter key.
- Since it is a cancel button, it responds to the cancel sequence Command+period on the Mac, Control+period on Windows.
- It is manually set on the Variable Page of the object properties window to respond to the Return keystroke.

Pressing the Return button simply cancels the List screen. If the list screen is running in its own process, then pressing Return closes the List window and terminates the process.

CHAPTER 6

Entry Forms

4Q Shell has conventions to help you construct entry screens. For the most part you can follow any convention or standards for the handling of input forms. The only element of the record entry process that must be coordinated with the rest of your application is the assignment of ID numbers to new records.

The ID System

When ID's Are Assigned

Optimistic

4Q Shell uses an optimistic approach to the assignment of ID numbers: it assumes that any time a new record is displayed the user is going to enter it.

4Q Shell is optimistic in the sense that it assigns ID numbers to new records and updates the ID counter for that table. It does this as soon as the user displays the new record in the entry screen. This means that whether the user enters the record or not, the ID number has been assigned and the counter incremented.

Pessimistic

In contrast, there is the pessimistic approach in which the ID number is only assigned after the user enters the record. In this approach the ID counter is only updated when the ID is actually assigned to a saved record. Each approach has its benefits and drawbacks.

There are two benefits to the optimistic approach:

- The ID number displayed is the actual ID number that will be assigned to the record if the user chooses to accept it.
- There is no chance that a locked ID counter record can prevent the entry of the new record, because the ID has already been assigned by the time the user gains access to the new record.

The drawback to the optimistic approach is that it generates “holes” in the sequence of ID numbers that are assigned to records. That means that if the new record screen is displayed and then canceled, the ID number assigned to that unsaved record is never reused. As a result there is a gap in the sequence of ID numbers for records in that table.

In general, the existence of gaps is of no relevance to the management of the database. In addition, there is substantial benefit that arises from being able to display the actual ID number in the new record’s input form.

If you find yourself in a situation where you must assign unique numbers without gaps, then you will have to modify the conventions used in 4Q Shell. You will have to assign ID numbers after the user enters a record. In that case you can still display the value of the next ID number that will be assigned when the record is saved. You can’t be sure that value won’t be taken by another user before the user looking at the record gets around to clicking on the Enter button.

In either the optimistic or pessimistic cases you can use the ID assignment methods in 4Q Shell. Only the optimistic approach is demonstrated in the Address entry form code that is discussed below.

Global Variables for Entry

Initializing at Startup

4th Dimension requires that only process or interprocess variables appear on forms. Local variables cannot appear. This forces us to use process and interprocess variables in spite of the fact that their use is contrary to good design principles.

In particular, this means that all of our buttons must be assigned to process variables. As a result there arises a danger when a nested series of forms is shown to the user. The danger is that one form will modify the values of process variables that appear on other forms. This is called the problem of variables being “overwritten”.

Button Variables

In the Address entry screen in 4Q Shell we are using six buttons:

Previous	bprPrevious, previous record button
Next	bnrNext, next record button
Delete	bdrDelete, delete record button
Cancel	bCancel, cancel button

OK	bnaValidate, no action, default button
vbiStrtTrac	invisible button, discussed in the next section

With the exception of vbiStrtTrac discussed below, these variables are declared in the compiler_4QpSH method. They do not need to be explicitly defined because 4D will automatically assign them the value zero when the form is displayed.

If you create an entry form for another table, you may or may not want to reuse these variables. For the greatest safety from the risk of overwriting variables you should create separate process variables for each form.

However, this becomes unwieldy as the number of forms grows large. It is also bad programming style. You will suffer various difficulties from the use of these global variables. Most of the problem comes from losing control over where these global variable values are set.

One treads a fine line between declaring new global variables for special purposes, and reusing existing global variables. I suggest that you declare a set of process variables for each table and that you reuse these variables where ever possible on forms associated with that table.

The Entry Form

There are four elements on the form that play special roles.

Menu bar	menu bars 6 or 7 are enabled for entry forms
Form Number	text typed directly onto the form
vLayoutDate	text variable that stores the form version date
BGround Rect	rectangle object
vbiStrtTrac	longint assigned to tiny button on the lower right hand side of page 0

Entry Menu Bar

The entry form menu bars that are supplied with the shell are menu bars 6 and 7. These menu bars are installed by the method _SYEnablMenu when it is called with the second parameter “ENTRY_AREA”. Menu bar 6 is installed if the user is a member of the Design group, menu bar 7 is installed if the user is not a Design group member.

These two menu bars may or may not be different. Even if they are not different I feel it is a good idea to have a menu bar reserved for the Designer as it may be useful for development or testing at some future time.

The _SYEnablMenu is called before the window is opened in which the list is displayed. In the case of the address book example, this method call occurs in the _SHAdressUserArea method. The _SHAdressUserArea method controls the Address user area.

Activating the Menu Bar

In order for the installed menu to be active when a form is opened in this manner you must assign a negative menu bar number as the form's associated menu. In 4D v6.0 and earlier it was possible to specify the negative value of a nonexistent menu. If you left the form's associated menu bar setting at the default value of zero, then the menu would appear to be active, but selecting menu items would not run the menu item methods.

In 4D v6.5 assigning a negative, nonexistent menu bar no longer activates the currently installed menu. Instead you must use the negative of an actually existing menu. If this menu has menu items, then these items are appended to the currently installed menu.

In order to activate the currently installed menu you must assign as an associated menu the negative value of a menu bar that has no items. Menu bar #5 was created for just this purpose.

You must assign the value "-5" as the associated menu bar for any entry screen for which the entry menu bars are to be active.

File Menu

This menu triggers the following system-related actions:

- **Page Setup**: opens 4D's page setup dialog box.
- **Show Control Screen**: opens the control screen, if it is not already open, and brings it to the foreground.
- **Quit**: quits the application.

These items call their own methods and these methods behave the same way for all entry forms.

The entry menu bar has no items on it that are specific to the record or the table being edited. All record or table specific actions are controlled either through buttons or items on a custom-assigned associated menu.

Form Number

Every form should be assigned a code number that is visible to the user. This number facilitates communication between users, programmers and designers. It provides an unambiguous way for people to refer to a form.

I use a form number that consists of two leading characters that identify the table, with an appended number that's different for each form. In addition I append the letters a, b, c, etc. to signify the page that's being displayed.

Version Variables

I call the text variable vLayoutDate a "version variable". I include this variable on every page of every form. I never assign a value to this variable, and it never appears in any expression.

The sole purpose of this variable is to provide me with a place to store a date value that indicates the date upon which each page of the form was last modified. This is stored in this vLayoutDate variable's object method.

If you open the method of the vLayoutDate variable on page zero you will see:

```
C_BOOLEAN($4QC950617_;$4QU961029_)
```

If you open the same variable as it appears on page one you'll see:

```
C_BOOLEAN($4QC950617a;$4QU981101a)
```

The local boolean variables declared in each object method contain the date when the corresponding page was created and the date it was last modified. This information is embedded in the name of the variable using the following convention.

The second two characters in the variable names indicate that the object was created and modified by Braided Matrix. The fourth character indicates whether the variable is a tag for the creation ("C") or last modification ("U") date. The 5th through 10th characters are a date in the form YYMMDD. The final character indicates the page this date applies to: "_" for page zero, "a" for page one, "b" for page two, etc.

Whenever I make a change to the page of a form, I manually change the version variable to reflect the current date.

The reason these dates are stored in variable names rather than variable values, or simply as text on the form, is because it allows me to use 4D Insider to locate all forms modified within a given date range. This has proved to be a valuable method of locating recent changes.

For more information about 4Q Shell's use of version variables, consult Developer Note #11: "Version Control", which is available from Braided Matrix.

Button Background Color

BGround Rect

This rectangle is used as a background for the main action buttons: Previous, Next, Delete, Cancel, OK. The method _SYLayoutDsply, discussed below, sets the color of this rectangle when the form is first displayed.

The user can change the color that is assigned to BGround Rect objects. Three different colors can be assigned, a different one for list, entry and dialog forms.

The form's button background color setting is accessed by pressing the Layout Colors button located on the System page of the User Preferences screen. To access this screen, go to the Control screen and select the File: Preferences menu item.

These color settings are stored with the 4Q_Preferences file that is automatically written to the System Folder:Preferenced:ACI folder. These preferences are assigned and stored separately for each user, and are read from disk when 4Q Shell is launched.

Emergency Screen

vbiStrtTrac — the Back Door button

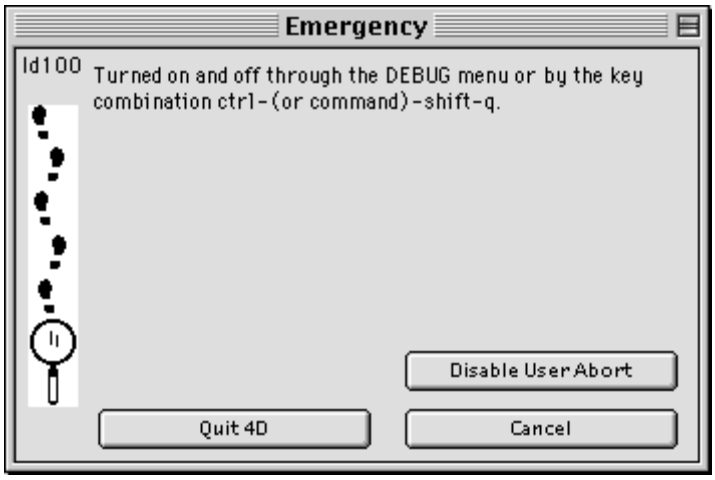
This invisible button is placed on the extreme lower right-hand corner of page zero of every entry form and dialog, or on page 1 of every list form. This button is assigned the keyboard equivalent Command-Shift-Q.

Placing this button on every screen ensures that the user can always press the Command-Shift-Q keystroke to reach the Emergency screen shown below. This screen provides the user with a way out of the application in the event that some programming error has disabled any means of reaching the Quit menu item.

The Quit 4D button immediately issues the Quit 4D command. It skips the usual quitting process and should only be used in emergencies.

The button labeled "Disable User Abort" provides the developer with a means of turning off On Event Call.

The Cancel button dismisses the Emergency Screen.



The Entry Form Method

Internal Variables

The entry form method uses several process variables.

vAddAnother	boolean, set to true when accepted if another new record is to be displayed
v4QExitPrcs	boolean, system variable set to true when the message is received to exit the process
vSYMsgText	text system variable that's assigned the value of any message received

The vAddAnother variable is used to pass a message back out from the entry form to the _SHAdressUserArea method. It appears there as the argument of the Until statement:

```
Repeat
   ADD RECORD([Address];*)
Until ((bCancel=1) | (bdrDelete=1) | v4QExitPrcs)
```

The v4QExitPrcs variable is a system variable that is used throughout 4th Quarter Accounting and 4Q Shell. It is set in the method _SYLayoutPhase in response to a message to exit the process. This message could be received

from any other process. It is sent whenever the user wants to close the process or quit the application.

In addition to these varibles, several other process variable arrays are used as popup items on the form itself. These are particular to the address example, and are not part of 4Q Shell's reusable structure.

Form Event = On Load

The form method consists of a Case Of statement that tests for various form events. The On Load event executes when the form is first displayed on screen.

```
: (Form event=On Load)
   C_BOOLEAN($4QU980914)
   C_LONGINT($Err;$ID)
   If ((Record number([Address])=Is new record) |
            (Record number([Address])><>vSY4DTranRN))
     $WindAddress_i:="Address Record Entry: 1 of 1"
   Else
     $WindAddress_i:="Address Record Entry: "+String(
       Selected record number([Address]))+" of "+
       String(Records in selection([Address]))
   End if
   SET WINDOW TITLE($WindAddress_i)
   _SYLayoutDsply ($CallerName;"INPUT")
   $Err:=_SHAddressEntry ($CallerName;"INITIALIZE")
   vAddAnother:=False
   If ($Err<0)
     POST KEY(Ascii("."); Command key mask)
   End if
```

The $4QU980914 variable is a version variable, as discussed above. Encoded in its name is the information that this form was last updated on September 14, 1998. This variable's name is changed each time the form method is modified. The variable itself is never assigned a value or used in any expression.

<>vSY4DTranRN Variable

The if statement tests to determine if the current record is a new record. This is done by testing the record's number. If the number equals the 4D constant "is new record", then the record is new.

However the record is also new if it has been saved in the context of a 4D transaction. In that case the record number will not equal "is new record" but will be greater than <>vSY4DTranRN. This is a 4Q Shell variable whose

value is constant. It should be used only for this purpose, and its value should never be changed.

The window title is then set to reflect whether we're looking at a new record, or one in a series of existing records.

The method _SYLayoutDsply is then called with $2="INPUT". This method sets the color of the BGround Rect according to the value passed in $2. The values of $2 recognized by this method are "INPUT", "OUTPUT", and "DIALOG".

Next is a call to initialize the form for displaying the current record. This is done with a call to a central procedure using the argument $2="INITIALIZE". This method will assign variable values and get a new ID number if necessary.

```
$Err:=_SHAddressEntry ($CallerName;"INITIALIZE")
```

An error value of zero is returned if no problems were encountered. If a problem is encountered a negative value is returned. When this happens the entry screen cancels itself. It does this by issuing the Post Key command for the command-period keystroke (or the control-period keystroke on Windows). This triggers the Cancel button, which exits the form.

It is not possible to exit the form by issuing a Cancel command from within the On Load phase.

Form Event = On Clicked

The On Clicked event runs whenever a button is clicked. It is used here to take actions based on the main navigation buttons. Actions triggered in response to buttons other than the main action buttons are handled in these button's object methods.

```
: (Form event=On Clicked)
   Case of
    : ((bnaValidate=1) | (bprPrevious=1) | (bnrNext=1))
      $Err:=_SHAddressEntry ($CallerName;"TEST")
      If ($Err=0)
         [Address]Salutation:=vySalutation{0}
         [Address]Phone1_Type:=vyPhoneType1{0}
         [Address]Phone2_Type:=vyPhoneType2{0}
         [Address]QuickCity:=[Address]City
         If (bnaValidate=1)
            ACCEPT
            vAddAnother:=True
         End if
```

```
    Else
      $Err:=_SYInputReject ($CallerName)
    End if
  End case
```

bnaValidate, bprPrevious, and bnrNext are all entry buttons. The last two perform automatic actions that will cause an Accept to be issued. I first check that the data entered is satisfactory. This is done through a call to _SYAddressEntry. This is the same method used to initialize new records, only on this occasion it's called with $2="Test".

The _SYAddressEntry method returns the value zero if all is well. If this is the case, the code proceeds to assign values to fields. If the user pressed the OK button, then the command to accept the record is issued. If the user pressed the Previous or Next buttons, there is no need to manually issue this command as it's done automatically.

The vAddAnother variable is set to true to tell the calling method that it should continue looping and reenter the data entry screen with another new record. This is handled by the Repeat/Until loop in the _SHAdressUserArea method, as mentioned before.

If the record failed data entry tests, then the _SHAddresEntry method returns a value other than zero. If this is the case, and if one of the automatic action buttons was pressed, then we must procedurally reject the automatic action. This is done with a call to the _SYInputReject method.

Form Event = On Activate or Form Event = On Outside Call

These form events are triggered either when this process is called from another process, or when this window is brought to the front to become the active window. This can be done either by the user or through 4D code.

```
: ((Form event=On Activate) | (Form event=On Outside Call))
   $Err:=_SYLayoutPhase ($CallerName;"";"INPUT";
                  ->[Address]; ->vSYMsgText)
   If ($Err=2)
    bnaValidate:=1
   End if
  End case
```

This is where forms check for messages sent from other processes. Messages are retrieved in the _SYLayoutPhase method. This is the same method that was called in the list form method. The only difference here is that the value of "INPUT" is passed as the second parameter to _SYLayoutPhase.

If a message is sent to the current process to exit, the method _SYLayoutPhase will handle it. If the user has made any changes to the current record, they will be asked if they want to save their changes before exiting. If they answer “No”, then _SYLayoutPhase issues the Cancel command. If they say “Yes”, then _SYLayoutPhase does not exit the entry screen and the user has the opportunity to enter the record.

ID Numbers

Obtaining an ID Value

The ID number for a new record is obtained in the _SHAddressEntry method. The code to obtain a new ID number can be placed anywhere in the On Load phase, but I prefer to centralize all code having to do with initializing the record in a single method. In this case I use the _SHAddressEntry method.

The _SHAddressEntry method consists of two parts, one to initialize the entry form, and another to test the data entered into the form. The first part of the method executes when the parameter $2=”INITIALIZE” is passed to the method. Of this we’ll only concern ourselves with the code that actually obtains the ID number

```
: ($2="INITIALIZE")
  If ((Record number([Address])=Is new record)  |
        (Record number([Address])><>vSY4DTranRN))
   If (_SYSetPrcdLock ($CallerName;Table name(->[Address])))
    $ID:=_SYIncrmntSeqNm ($CallerName;<>vSeqAddr)
    UNLOAD RECORD([Procedure_Lock])
    If ($ID>0)
  `
  ` perform various assignments
  `
   Else
    vErrorText:="A new record can not be created now because a
        record ID number "+"is unavailable. Try again in a
        moment."
    $Err:=-2
   End if
  End if
```

This code starts by determining whether the record is a new or preexisting record. An ID number is only needed for new records. The method that obtains new ID numbers is called _SYIncrmntSeqNm. Before this is called there is a call to the _SYSetPrcdLock method.

The _SYSetPrcdLock method takes a string value as its second parameter. It treats this value like a semaphore but instead of using 4D's built-in semaphore system, it loads a record in the Procedure_Lock table.

Procedure_Lock table

If a record by this name does not exist in the Procedure_Lock table, this method will create one. If this record can be loaded in a read/write state the _SYSetPrcdLock method loads it and returns the value true. If it cannot be loaded in read/write state, it means that it is in use by another user. In that case the method returns the value false.

If the requested Procedure_Lock record is locked, the _SYSetPrcdLock method will wait for access to the indicated record, and it will give the user the option of canceling the waiting process. If access has not been obtained after 15 seconds, the method gives up and returns the value "false".

After _SYSetPrcdLock returns the value "true", the _SHAddressEntry method requests a new ID number. This is done by the call:

```
$ID:=_SYIncrmntSeqNm ($CallerName;<>vSeqAddr)
```

This method takes a longint value as its second parameter. The variable <>vSeqAddr is passed. This variable is used as a constant. I refer to the value of this varaible as the sequence number. The sequence number identifies the record in the ID_Number table that is used to assign sequence numbers to Address records.

The sequence number is set to a particular value at startup and is never changed. Each table has its own sequence number interprocess variable. These variables act as constants.

The _SYIncrmntSeqNm method locates the indicated ID_Number record, reads the value of its Next_Number field. Increments the Next_Number field by one. Saves the ID_Number record and returns the value of Next_Number field to the calling method. This number is then assigned to the local variable $ID.

_SYIncrmntSeqNm considers it a fatal error if it cannot find an ID_Number record with the indicated sequence number. This is because a failure in the ID system can cause severe integrity errors in the datafile. In this event _SYIncrmntSeqNm will automatically quit the application as a precaution. Refer to the following section for the description of how to create your ID_Number records.

After the call to _SYIncrmntSeqNm it is necessary to unload the Procedure_Lock record. If you fail to unload this record it will remain in

read/write state. This would prevent other users from getting a new address ID number.

Defining a Sequence Number

Each record in the ID_Number table is distinguished by its sequence number. The sequence number acts as the primary key for these records. Each sequence number identifies a particular ID sequence. There should be a separate sequence for every table to which ID numbers are assigned.

While it is not strictly necessary to declare an interprocess variable to be used as a constant for storing each sequence number, it is recommended as a way of preventing programming errors. It's easier to remember or to locate the name of a variable used as a constant than it is to remember the value of the sequence number itself.

The sequence number value is assigned to the <>vSeqAddr variable in the method __SH_AddressInit. Since this method always runs at startup, we are assured that this interprocess variable will always be assigned.

It is up to you to declare a sequence number variable for the tables that you create. It is also your responsibility never to change values assigned to these variables. To more easily recognize sequence number variables you should follow the 4Q convention of assigning them a name that starts with "<>vSeq".

4th Quarter Accounting reserves the use of sequence numbers between 100 and 200. You are free to use any other numbers to identify your ID_Number records.

Creating an ID_Number Record

Whenever a new datafile is created, the necessary ID_Number records must also be created. These records are created with a call to _SY_IDRecord. The form of the call to create an ID_Number record for the Sample table would be:

```
$Err:=_SY_IDRecord ($CallerName;"CREATE";<>vSeqSample;1;
      Table(->[Sample]);Field(->[Sample]ID))
```

This call takes the following parameters

$1= name of the calling method (string)
$2= action word "CREATE" (text)
$3= sequence number to assign to the new record (longint)
$4= value of the next ID number to assign to the new record (longint)
$5= pointer to the table the ID number will be used for
$6= pointer to the key field in this table
$0= error code (longint)

Before creating a new ID_Number record the _SY_IDRecord method will search ID_Number table to see if the indicated record already exists. It will only create a new ID_Number record if the indicated record is not found.

If a new record is not needed the method will return an error code of 0. If a new record is needed and can be created, it will also return the value 0. If a new record is needed and the method cannot create it, it will return a negative error code.

The call to create the ID_Number record should be placed in that part of the initialization method that runs when the application is first launched. In 4Q Shell this appears in the __SY_AdressInit method.

CHAPTER 7

Address Example

The Address database included in 4Q Shell provides an example of how you can handle record access, printing, entry and display.

In 4D the user interface is closely tied to how you enter and access data. 4D uses the concept of a “current record” and a “current selection” to handle lists of records and individual records. These concepts are used by 4D’s interface commands like Modify Selection, Export Text, Print Selection, etc.

The address database provides a set of forms to handle all address related tasks. It is possible to consolidate the number of forms you use by using the same for a variety of purposes. Since this kind of form reuse is not appropriate in general we have not done it in the address database example. Instead, we have created a form for each of a dozen general tasks.

When you create your own tables you must create your own set of forms for that table. You can use the address forms as templates that tell you what forms you’ll need, and how these forms can be structured. Your forms will have different fields and will probably have different buttons and lables. It’s likely that you’ll need additional forms to support special functions. The address forms provided with the shell provide a basis that you can build on.

Address Forms

Any use of forms requires a knowledge of how entry, list and dialog forms are used. Refer to your 4th Dimension documentation for details.

We have already discussed the structure of list forms in chapter 5 and entry forms in chapter 6. We use the following convention for the naming of forms:

The name of each form begins with “q”

A suffix is added to the name of each form to indicate it’s designed use. These suffixes are:

entry form	“_i”
list form	“_o”
included subform	“_ic”
dialog	“_d”

A prefix is added to the name of each form if the form is used for one of the following purposes:

printing	“PRT”
writing to file	“DISK”
queries	“SCH”
User environment use	“UTIL”

The structure of forms in the address table is self explanitory. The printing forms, for example, support 4D’s Print Selection command. The disk export forms are used in connection with the Export Text and other 4D export commands.

The following list gives the name of each of the forms in the address database and a brief explanation of each.

qAddressBlank_i

Blank form that's set as the input form used with Display Selection and Modify Selection. This form is automatically opened by 4D when a user double clicks on a record. The form then automatically closes itself without ever

appearing on-screen. The qAddress_i form is then used to display the actual record in separate process.

qAddressList_d

This dialog form is used to display an array of address records to display to and solicit a selection from the user.

qAddress__d

This is a dialog that displays address information. Because it's a dialog it can not include enterable fields. All enterable values must be supported as variables.

qAddress_i

This is the address entry form. The record must be unlocked in order for the fields to be enterable.

qAddress_ic

This is an included list form. It provides a list of addresses that can be placed as a subform on another parent form.

qAddress_o

The address output or list form. Used with Display Selection and Modify Selection commands.

qDISK1Address_o

A form used to export the summary form of address records using Export Text.

qDISK2Address_o

A form used to export the detail form of address records using Export Text.

qPRT1Address_o

Used to print the summary form of address records using Print Selection.

qPRT2Address_o

Used to print the detail form of address records using Print Selection.

qSCHAddress_d

A custom Address table search screen. This screen is used instead of 4D's generic search screen in order for you, the developer, to create smarter, simpler, and faster queries. You can also use 4D's generic search screen simply by calling Query([Address]).

qUTILAddress_i

A utility entry form to enable you to have complete access to the address records through the User environment. When used in the User environment this form must be set as the default entry form.

qUTILAddress_o

A utility list form to enable you to have complete list of address records through the User environment. When used in the User environment this form must be set as the default list form.

Checklist for Adding a New Table

The following list enumerates steps you can follow when you create a new table. In each step you create new objects, or copy and use existing address objects as templates.

Add the Table and its Fields to the Structure

Add any child Tables and establish relations

Create and copy all of the example Address forms for the table added. The scripts are copied with the form objects but you must copy the form method by its self.

(Don't do any modifications yet)

Make the qUTILxxxx_I the input form, and the qUTILxxxx_I the output form.

Create a Popup object on the Control Screen and add its Script

Create a List to populate the Popup object

Decide on a module prefix for the methods you'll create to handle this new table. The prefix is two-characters that you'll add to the start of each method name. Let's use "AB" for this example. Do not prefix your method names with underscores. Leading underscores should remain reserved for 4Q Shell methods.

Create a Method for initialization like __SH_AddressInit and declare the process and interprocess variables. Give this method a name like "ABInit". Populate the arrays and make a call to _SY_IDRecord for any Keys or sequence numbers to ensure that they exist.

Add the above method to __SYCustomInit

Add the forms to _SYCustomDefltLayouts

Add a method that's launched when the user opens your new area. This method corresponds to the existing _SHAdressUserArea method. Name this method something like "ABUserArea"

Place references to your "ABUserArea" method in the calls to _SYOpenProcess that appear in the _SYCtrlPanelPop method.

Add the Tables to _SYCustomReadOnlyAll and _SYCustomUnload

CHAPTER 8

System Methods

This chapter presents an abbreviated list of system methods. They are first listed by function. They are then listed alphabetically with a brief description of what they do.

Methods in the System Module carry the prefix "_SY" and they perform general tasks like managing arrays, handling windows, etc. They are designed to perform a common function or to encapsulate one of 4D's internal commands when this improves the function of those commands.

Methods whose names begin with an double underscore (__) are called exclusively as part of the system or new process startup routine. Methods whose names start with a single underscore are also involved in these initialization routines but the double underscore methods appear only in these routines.

The following breakdown first lists the system methods by their area of function and then describes each with a few sentences. The descriptions are divided into three parts. The first part gives the purpose, the second part the nature of the information that must be passed to the method, and the third part describes any special conditions or features.

I am not listing the parameters passed and returned by each method since these may change in future releases of the 4Q Shell. Refer to the header in the method itself for a list of the number, identity and type of each required (or optional) parameter.

Not all system methods are included in this list. I have ommitted those methods that apply to the 4Q Full and Core accounting programs, even though these methods may be included in the source code provided with the shell.

Parameter Structure

Return an Error Code

Most of these methods are functions. They return an error code. An error code value that's less than zero usually indicates an error, if one occurs. A returned value of zero indicates no error. A positive number indicates no error plus additional information. However, some of these calls return booleans. Refer to the actual code for details.

Caller Name

As a general rule, all calls are passed the name of the method that calls them in their first parameter. There are a few exceptions, such as _SYNil and _SYOpnWindow. There is no good reason for these exceptions and in the future I hope to pass the name of the calling method to all methods without exception.

Action Parameter

Many methods take an action value as a second parameter. This is a text value that provides additional information to the method as to what it is to do, or how it is to behave. This is another convention that I am currently applying to all new methods whether they need additional specifications at they time they're first created, or not.

a. Methods by Function

Addresses

_SYAddrParent
_SYFilAdrFields
_SYHdlAddresAra
_SYHdlAddress
_SYPrtAdrLabels
_SYSchTypeAdres
_SYSelAddrsList

Arrays

_SYBulletAraLin
_SYCtrlPanelPop
_SYFillDateAray
_SYFillHistAray
_SYEnterTextVar
_SYFndInAray
_SYHdlAsignPop
_SYHdlSeltPop
_SYMoveAraHiLit
_SYMoveAraRow
_SYPrtHistAray
_SYSelFromArray
_SYSELinTextAra
_SYSetArayMrk
_SYTransposElms
_SYUserDfltAray
_SYFillGnrlAray

Blobs

_SYBlobVariables

Customization

__SYCustomInit
_SYCustomDefltLayouts
_SYCustomReadOnlyAll
_SYCustomUnload

Dates

_SYAsgMnYearDat
_SYCekDateEntrd
_SYDateTimeStamp
_SYEndOfMonDate
_SYEntryDatePop
_SYFillDateAray
_SYHdlDateEntry
_SYMonNameNum
_SYSchPastDates
_SYSetDateRange

DB Structure & Administration

_SYDBReset
_SYHdlDfltBool
_SYHdlPictEdit
_SYIncrmntAlpha
_SYIncrmntSeqNm
_SYLogHistory
_SYMBAbout
_SYMBQuit

_SYSetDelimiter
_SYSetPrcdLock

Disk Tables

_SYCreateFile
_SYMakeDiskFile
_SYParsePath
_SYReadFileToAr
_SYSetDelimiter
_SYTestFilePath

Event & Error Handling

_SYAbortKeyTrap
_SYErrorMsgs
_SYOnErrCall
_SYOnEventCall
_SYSetCanclKey

Layout

_SYAskSaveChgs
_SYCustomDefltLayouts
_SYDefltLayouts
_SYHdlSchbyLay
_SYInputReject
_SYLayoutDsply
_SYLayoutPhase

Menus

_SYDisableMBs
_SYEnablMenu
_SYMBAbout
_SYMBCntrlScren
_SYMBFocus
_SYMBGoCtrlScrn
_SYMBReports
_SYMBSchByExample
_SYMBPageSetUp
_SYMBSort
_SYMBSets
_SYMBQuit
_SYMrkAssocMenu

Multi-user

_SYFndNmOnNtwrk
_SYUserInGroup

Printing

_SYHndlPrtSpecs
_SYMBPageSetUp
_SYPrtAdrLabels
_SYPrtHistAray
_SYPrtStandard
_SYSetPrtSimple
_SYSetPrtSpecs
_SYSetFilChgMrk
_SYUserDflt
_SYUserDfltAray
_SYWait
_SY_IDRecord

Processes & Transactions

_SYHdlProcess

_SYMBGoCtrlScrn
_SYMsgCentral
_SYMsgReceive
_SYMsgSend
_SYMsgUtility
_SYNewProcess
_SYOpenProcess
_SYPrcsAttribs
_SYPrcsMessage
_SYShowProces
_SYQuitLoop
_SYTransaction

Records

_SYCEKJust1Valu
_SYCustomReadOnlyAll
_SYCustomUnload
_SYHdlAddresAra
_SYHdlAddress
_SYDistinctValu
_SYFilAdrFields
_SYFndDupsOfOne
_SYHdlAddress
_SYGlobChange_d
_SYHdlDfltIntrvl
_SYHdlGlobChg
_SYHdlHistSumry
_SYHistorySumry
_SYMakNewTxtRec
_SYModRcrd
_SYPopAllOffStk
_SYReadOnlyAll
_SYSchTypeAdres
_SYSELAddrsList
_SYShoDupOption
_SYShoNAskDups
_SYTransposRows
_SYUnLoadAll
_SY_IDRecord

Search & Sort

_SYAskUsrMnySRT
_SYGenSRT_d
_SYHdlSchbyLay
_SYHdlSRTDirct
_SYSchEngine
_SYSchInColumn
_SYSchPastDates
_SYSrtManyLevl

Selections

_SYCekSet2Delet
_SYClearCurrSel
_SYDistinctValu
_SYDoFocusPop
_SYGlobChange_d
_SYHdlGlobChg
_SYReadOnlyAll
_SYTotAmtInSEL
_SYUnLoadAll

Sets

_SYCekSelUsrSet
_SYLoadSet
_SYTestEqualSet

String

_SYApndWildCard
_SYASCIInChar
_SYCaps
_SYCEKChar
_SYCenterText
_SYFormatStr
_SYHdlEmbedTxt
_SYIncrmntAlpha
_SYInsrTxtInStr
_SYParseAlpha
_SYParsePath
_SYPhoneFormat
_SYRemovChars
_SYWrapText

User Interface

_SYAlert
_SYArrayOrTextMsg
_SYAsk3Choices
_SYAsk4Choices
_SYCustomReqst
_SYCustomCnfrm
_SYHdlSchbyLay
_SYMessage
_SYMBCntrlScren

Utilities

_SYDistinctValu
_SYHdlDfltIntrvl
_SYHdlEmbedTxt
_SYHideUserWind
_SYLogHistory
_SYMsgReceive
_SYMsgSend
_SYNil
_SYParseAlpha
_SYParsePath
_SYSeconds
_SYShowUserWind
_SYUserInGroup
_SYWorkArounds

Windows

_SYCancelBox
_SYClosWindow
_SYEnterBox
_SYOpnWindow
_SYSetWindSize

b. Alphabetical List of Methods

__SYCustomInit

Performs initialization of variables, sets, selections and other items that need to be done at application startup or when a new process is started. Called automatically by 4Q when the application is first started and when new processes are spawned.

-

Receives an action code that has the value “Startup” when the application is first started. Action code is blank for subsequent calls made when new processes are created. Returns an error code that should be 0 if there are no errors.

-

Place the startup actions for all your tables here. Initialize interprocess objects when the action code “Startup” is received. Initialize process objects when the action code of “Startup” or a blank action code is received.

_SYAbortKeyTrap

On event call method installed by _SYSetCancelKey to trap the user "abort" key sequence of command+period.

-

Takes no parameters.

-

Sets ◊vUserAbort to TRUE when abort key sequence is intercepted.

_SYAddrParent

Locates parent records to a specified address record. Makes the parents the current records in their tables. Addresses can be related one-to-many, many-to-one, or one-to-many across several tables.

-

Takes an action parameter, the table number and parent record ID, or alternatively the address ID and type of address records to screen for.

-

Returns the number of parent records found. Can be instructed to terminate after the first related record is located.

_SYAlert

Displays the passed string in an alert dialog or writes it to a disk file.

-

Takes the string to be displayed.

-

Does the same thing as 4D's Alert function, except that the message can be sent to a text file instead of the screen. In this case the system's exectution does not stop and wait for the user's confirmation.

_SYApndWildCard

Adds a "@" to the end of the passed string if that string does not already end in an "@" character.

-

Takes the string to be appended, returns the appended string.

-

Must be passed an alpha value and will return an alpha value.

_SYArrayOrTextMsg

Displays a long text message or the full contexts of a text array in a new window. Array elements are concatenated and displayed in the scrollable text area. You can also supply a window title, caption, and button lables. The screen has a Print button allowing the user to print the text to paper.

-

Takes strings giving the window title and two button lables. If the second button lable is blank then the button is hidden.

Takes a pointer to the text array to display. The 0th element of the array is used as the caption.

-

Returns a value indicating which of the two buttons was pressed.

_SYASCIIInChar

Performs mapping to and from ASCII characters and their ASCII numbers, also provides a string descriptor of the character. Uses the standard Geneva character map.

-

Takes pointers to character, character value, and string description variable. Also requires boolean flag to indicate which direction to make the translation.

-

By passing a pointer to both a character and numeric value the method will either consider the character and assign its number, or look at the number and assign the character, depending on the flag setting. In either case the method assigns a text description of the assigned character. For example it will assign "carriage retn." if passed a ASCII character number of 13.

_SYAsgMnYearDat

Accepts a date specified in string form as MM/YYYY and converts this to a date type that it assigns to the field or variable pointed to. The method correctly handles both MM/DD/YY and DD/MM/YY formats.

-

Takes a pointer to a string and a pointer to a date variable or field.

-

This assigns a date of the first day in the specified month.

_SYAsk3Choices

Opens a dialog window that presents the user with a text description and three buttons for their response. The calling program must monitor the settings of the buttons to determine the response.

-

Takes text to display in a dialog box and the text to place in each of the three buttons.

-

See the method heading for the names of the buttons. One of which will be set to = 1 upon completion of the method.

_SYAsk4Choices

Opens a dialog window that presents the user with a text description and FOUR buttons for their response.
The calling program must monitor the settings of the buttons to determine the response.

-

Takes text to display in a dialog box and the text to place in each of the four buttons.

-

See the method heading for the names of the buttons. One of which will be set to = 1 upon completion of the method.

_SYAskSaveChgs

Called from input layouts or dialogs when a message has been received to exit the layout. If the record has been modified it asks the user if what they want to do.

-

Takes an action parameter and pointer to the table of the currently displayed record. Returns a code indicating if the record needs to be saved.

-

If the record needs to be saved it passes a message back to the calling method.

_SYAskUsrMnySRT

This is called before automatically performing a multiple criteria sort.

-

Takes a pointer to the table being sorted. Returns a boolean in $0.

-

Checks if the number of records in the selection to be sorted exceeds the maximum number set in the user preferences. If so, it checks with the user to see whether to do the multiple sort and returns their answer in $0 as a boolean. The method will not ask for confirmation if the variable vAskdDblSRT is true. This variable indicates that the user has already been asked this question.

_SYBlobVariables

Receives an action code, a pointer to a blob and an unspecified number of additional pointers to variables.

-

The action code “STORE” tells the method to store the variables to which the pointer refere in the blob. The action code “RECALL” tells the method to extract the variables from the blob and place them in the variables that are pointed to.

-

Will not store variables of type pointer, arrays of pointers, or two-dimensional arrays. Returns a 0 error code value if no error occurs.

_SYBulletAraLin

Places or removes a bullet character in the specified text array.

-

Takes a pointer to the text array that displays a bullet when selected; and the value of the selected array element; and the character to use as a bullet.

-

If the selected element of the referred array equals the bullet character then the character is removed. Otherwise the bullet character is assigned to the array element.

_SYCancelBox

Passed as a parameter to _SYOpnWindow to be installed as a window “close box” method.

-

Runs when the user clicks on the close box. Issues the Cancel command.

-

Takes no parameters, returns no parameters.

_SYCaps

Capitalizes the first character of each word in the string passed to it.

-

Takes the string to be modified. Returns the altered string in $0.

-

Calls the method _SYCEKChar that capitalizes any character that follows a single particular character. By calling _SYCEKChar repeatedly capitalization is effected following various different characters (i.e. blank, dash, etc.)

_SYCEKChar

Capitalizes the every character that follows a specific character.

-

Takes the string to be modified, and the character following which characters are to be capitalized. Returns the altered string in $0.

-

_SYCekDateEntrd

Checks an entered date to see if its within an acceptable range.

-

Takes pointer to the field or variable where the date is entered; number of days in the past and a number of days in the future. Returns error code in $0.

-

The referenced date is checked against the range defined with respect to the current date and the number of days in the past and the number of days in the future.

_SYCEKJust1Valu

Called to examine any table's current selection to see if all the records have the same value in one of their fields.

-

Takes a pointer to the field to be checked. Returns TRUE if all the records have the same value in that field.

-

Changes the current record of the selection but does not change the selection.

_SYCekSelUsrSet

Checks the number of records in the UserSet. Alerts the user if the number of records is inappropriate.

-

Takes a boolean indicating whether it's alright to have more than one record in the UserSet. Also takes a string that gives the name of the records so that the method can refer to them when alerting the user.

-

To be used in output layouts when the user indicates an action based on highlighted records. This does not change the selection or the UserSet. Issues an ACCEPT command when satisfied.

_SYCekSet2Delet

Checks the UserSet to see if it contains an acceptible selection of records that can be deleted. Alerts the user if not.

-

Takes a boolean variable indicating whether multiple records can be deleted at once. Takes a string indicating the name of the displayed records for alerting the user.

-

Does not change the UserSet of selection. Issues the CANCEL command if satisfied.

_SYClearCurrSel

Clears the current selection stored for an indicated table.

-

Takes a pointer to the table whose selection is to be emptied.

-

Results in their being no current selection and no current record for the table indicated.

_SYCenterText

Centers text by adding equal numbers of nonbreaking spaces to either end of each line. Each line ends up with the maximum specified length.

-

Takes an action code, a pointer to the text to be centered, and the value of maximum line length.

-

Blank spaces are not stripped, each line should be marked with a carriage return and be shorter than the maximum line length.

_SYCreateFile

This encapsulates 4D's Create document and Append document commands. It will create the specified TEXT document if it doesn't exist or will ask you if you want to append to it if it already exists.

-

Takes a document path and name and returns a document reference.

-

Performs system level error handling. For example it will inform the user if the docuemnt is already open.

_SYCtrlPanelPop

This long method handles all the pop up events associated with the Control Panel screen. It sets various variables and calls methods depending on which element of which pop up was chosen.

-

Takes a pointer to an array used as a pop up variable.

-

Does array handling, uses PostKey to translate pop up choices to menu bar selections.

_SYCustomCnfrm

Encapsulates 4D's Confirm function by presenting a configurable confirm dialog.

-

Takes a string indicating whether the OK or Cancel button is default and the message to present. Returns 0 or 1.

-

This dialog can display more text than 4D's Confirm function.

_SYCustomDefltLayouts

Sets the default forms for tables added to custom versions of 4th Quarter. Called automatically by 4Q when the application is first started and when new processes are spawned.

-

Receives an action code that has the value “Startup” when the application is first started. Action code is blank for subse-

quent calls made when new processes are created. Returns an error code that should be 0 if there are no errors.

-

Assignment default forms for all your tables here. These should be assigned both at the startup of the application and when new processes are spqwned.

_SYCustomReadOnlyAll

Sets all the tables added in a custom version of 4th Quarter to Read Only mode. Called automatically by 4Q when the application is first started and when new processes are spawned.

-

Receives an action code that has the value “Startup” when the application is first started. Action code is blank for subsequent calls made when new processes are created. Returns an error code that should be 0 if there are no errors.

-

Call this method, in conjunction with _SYCustomUnload, at the begining of all process, when exiting output forms, and at the end of any process that may leave records in various tables in a loaded and locked state.

_SYCustomReqst

This encapsulates 4D's Request function. It allows for screening, setting the display font, and displaying more instructions.

-

Takes the prompt string, display font, response length, as well as well as various filtering, formatting and default response instructions. If a date format is specified (eg. “Short”, “Long”, etc.) the dialog limits the user to entering a date.

-

Only returns strings of 80 characters or less.

_SYCustomUnload

Unloads the curent record in all the tables added in a custom version of 4th Quarter to Read Only mode. Called automatically by 4Q when the application is first started and when new processes are spawned.

-

Receives an action code that has the value “Startup” when the application is first started. Action code is blank for subsequent calls made when new processes are created. Returns an error code that should be 0 if there are no errors.

-

Call this method, in conjunction with _SYCustomReadOnlyAll, at the begining of all process, when exiting output forms, and at the end of any process that may leave records in various tables in a loaded and locked state.

_SYDateTimeStamp

Call to assign the server’s current date or time to both a variable and a text field.

-

Takes the action code of “Date” or “Time” and two pointers. One pointer is to a date or time variable, the other to a text field. The date or time variable is set to either the current date or time. The text field is prefixed with the current date or time. The date is returned in “short” format. The time in “HH MM AM PM” format.

-

The date/time pointer can be nil, in which case only the text field is modified. The pointer to the text field can be absent or nil, in which case only the date/time variable is modified.

_SYDBReset

Performs management functions relating to sequence counter records.

-

Takes an action code and returns an error value.

-

Should only be changed to add reference to user-created ID records. The form and action codes of the method should not be changed.

_SYDefltLayouts

Called to reestablish the default layout assignments.

-

Takes no parameters.

-

Call upon returning to the control screen or upon entering a user area.

_SYDisableMBs

Disables all menus on the current menu bar.

-

Takes an action code whose values can be “Quit”, or “Associated”. “Quit” causes the Quit item on the File menu to remain enabled. “Associated” leaves items on the associated menu enabled. If no code is passed then all items on all menus are disabled.

-

Call before entering a User Area that doesn't use the menu bar.

_SYDistinctValu

Analyzes the values in an indicated field of the current selection and returns an array containing only distinct values.

-

Takes a pointer to the field to be analyzed, and a pointer to the array in which to place the distinct values.

-

Operates on indexed fields of any type. Referenced array must exist and be of a type consistent with the indicated field. Returns the number of distinct values in $0.

_SYDoFocusPop

Reduces the current selection either by focusing on or excluding the UserSet.

-

Takes a pointer to the table, pointer to the pop up variable, 3 pointers to sort fields, and whether to sort up or down, as well as other variables.

-

Called from the Focus/Exclude pop up located in the footer of most layouts.

_SYEnablMenu

Sets either the User or Developer menu bar.

-

Takes no parameters.

-

Called when returning to the control screen.

_SYEndOfMonDate

Calculates the last day in the month.

-

Takes any date and returns the last date in the given month.

-

_SYEnterBox

Passed as a parameter to _SYOpnWindow to be installed as a window “close box” method.

-

Runs when the user clicks on the close box. Issues the Post-key command for the Enter button. Triggers the default button.

-

Takes no parameters, returns no parameters.

_SYEnterTextVar

Opens a window and asks the user to enter text.

-

Takes the window title, initial value, displayed instructions, and entry filter. Also takes a pointer to the variable which is assigned the entered text.

-

Allows for the entry of about a paragraph of text. Presents Enter and Cancel buttons.

_SYEntryDatePop

Handles the named-date pop up. Assigns the indicated date to a variable.

-

Takes a pointer to the date variable to be assigned the date chosen from the popup, handles all pop up array management.

-

The dates are specified in terms of first of fiscal year, quarters, and months.

_SYFilAdrFields

Searchs address table and assigns located address to the passed variables.

-

Takes an action code, the number of the linked table, the ID number of the linked record, and the type of address to look for. Also takes pointers to 11 variables that are assigned the located address fields.

-

Called whenever stored addresses need to be displayed.

_SYErrorMsgs

Translates the Macintosh OS and 4th Dimension error codes into their corresponding text descriptions.

-

Takes the error code (an integer, usually negative)and returns an error string translation.

-

Called when testing for an error, as in testing for the existence of a disk file. Returns blank if the passed code is 0.

_SYFillDateAray

Initializes the named-dates arrays used as a pop up array for faster data entry.

-

Takes no parameters.

-

This initializes two arrays, one is of type text and stores the date names as displayed to the user. The other parallel array of type date stores the corresponding date values.

_SYFillGnrlAray

method fills arrays that are used to cache rarely changing database information.

-

Takes an action code and a key word indicating whether to update only process
variable or both process and interprocess variables.

-

• Delimiters: handles caching of array containing possible field and record delimiters used in importing and exporting text files.
• Payroll_GL_Accounts: Fills arrays with the names, numbers, and ID's of General Ledger payroll accounts.
• Cash_Subaccounts: Fills arrays with the names, numbers, and ID's of accounts associated with journals of Receipt/Disbursement type.
• DateArray: handles arrays used to present user with various date options.
• CreditTerms: handles arrays that present credit term options in invoices, PO's, and other entry screens.

• GLSalesTax: Handles an array of account ID's for those GL accounts marked as being of sales tax type. This is indicated by a tag value in the TaxType field.
• BugReports: This adds import and export of bug reports to the table transfer arrays.

_SYFillHistAray

Called to initialize the arrays used in the [System_Default]qShowHistory_d dialog.
-
Takes 6 pointers to arrays that hold dates and corresponding historical information.
-
Collates the information passed into totals for reporting periods. This can be called in any context where the qShowHistory_d is used. This layout is meant to be reusable.

_SYFndDupsOfOne

Checks for records that hold duplicate information to that stored in the current record.
-
Takes a pointer to an array whose elements are pointers to the fields that are to be used to test for uniqueness. Returns the number of records besides the current record that contain all of the specified values.
-
Used to test for uniqueness according to multiple, simultaneous criteria. This resets the current selection to contain all records meeting the specified conditions.

_SYFndInAray

Find the element in an array that best matches a specified value.
-
Takes instructions for how to perform search, pointer to an array of any type, value to be located and pointer to variable set to the value located.

-
The elements of the array to be search must be in ascending order. The array is not modified.

_SYFndNmOnNtwrk

Finds the number of users on the network.
-
Takes an action code with additional specifications. Returns the number of users of specified types that are currently logged on the network.
-
Only returns number of users accessing the datafile.

_SYFormatStr

Called to modify the contents of a passed text or string variable, such as
to examine leading characters for prefix values and removing those that are
redundant or misplaced.
-
Takes two action codes (two text parameters to specify actions), a pointer
to the variable to be checked or modified. Returns an error code.
-
Use to clean up strings used for concatenation of search instructions and
search values.

_SYGenSRT_d

Handles the general sort dialog, setting variables according to user choices.
-
Takes pointers to table or subtable, 3 pointers to variables where sort field pointers are stored, 3 pointers to 3 variables that store TRUE if the sort is ascending.

-

Contains several set up and processing methods for this general dialog.

_SYGlobChange_d

Enables the user to make changes to the specified fields in all the highlighted records.

-

Takes an action indication code, pointer to the table being modified, pointer to dispaly text array of fields that can be modified (can be aliases), also takes pointers to arrays containing special instructions for how to handle entry for each case.

-

The method examines the fields pointed to and displays a different layout page for each different type of field value. The UserSet must contain the highlighted records.

_SYHdlAddress

Handles the creation, modification, and deletion of address records.

-

Takes an action indication code, and record number of address record or related parent record.

-

Supports a variety of methods relating to related address record management.

_SYHdlAsignPop

Handles an alpha or numeric popup and assigns a corresponding value to an indicated variable. Does not handle date arrays.

-

Takes a pointer to the array being chosen from, pointer to variable that will be assigned a value, and a pointer to a corresponding array to take the assignable value from (can be the same array as was used to make the choice).

-

If no item is chosen the previous array value is restored. When new item is chosen the variable is reassigned. By using two arrays the user can choose from an array of names and have a related code or value assigned to the variable.

_SYHdlDateEntry

Called to handle various aspects of the entry of date data.

-

Takes an action indication code and a pointer to a date value.

-

Responds to user's actions to perform various checks and modifications to entry variables.

_SYHdlDfltBool

Handles the radio buttons used to display the [System_Default]sfBoolean subrecord values.

-

Takes an action indication code, the subrecord identifying code, pointers to the radio buttons appearing on the layout controlling the Yes/No options.

-

The method will do such actions as: READ, SET, or INIT subrecords. The later creates and initializes all system subrecords. Use this method where ever you need to access the [System_Default] boolean subtable records,

_SYHdlDfltIntrvl

Handles the initialization, and setting and reading values from the [System_Default]sfInterval subrecords. These customizable records store a number, used as an interval for some purpose, and a primary key code value that's used to distinguish different interval subrecords.

-

Takes an action code, the subrecord identifying code, and an optional identifying code and initial value. Returns the value

of the specified interval as a longint, or a negative value if there is an error.
-
The method takes the action codes: READ, SET, or INIT subrecords. The INIT code creates and initializes a specified interval subrecord. Use this method to access [System_Default] sfInterval subtable records.

_SYHdlEmbedTxt

Reads or writes a substring value that's embedded within a larger, concatenatedstring value, and that's sandwiched between two string markers.
-
Takes an action code, pointer to the larger string and pointer to the substring, and the strings marking the beging and end of the target substring. Returns and error code.
-
Handles the extraction and insertion of demarcation substrings into a longer string.

_SYHdlGlobChg

Formats and makes changes to the interface in response to a selection of a global replace field name in the popup.
-
Takes an action indication code.
-
called by scripts in variables of layout-[System_Default];'qGlobalChange_d'

_SYHdlHistSumry

Works in conjunction with layout [System_Default]qShowHistory, performs all related processing.
-
Takes an action indication code and a pointer to a variable.
-

Initializes the display, resets displayed values in response to user's request. Supports any kind of history; uses arrays not record selections.

_SYHdlPictEdit

Handles actions related to the [Picture] records.
-
Takes an action indication code.
-
Handles adding, deleting, searching, and other functions.

_SYHdlProcess

Handles the creation of new processes. All processes should be created
through calls to this method.
-
Takes an action code, and specifications for the processes name, size, and method. Also takes an optional pointer to an array of text parameters. The elements in this array of parameters are passed to the indicated new process method.
-
Any type of value can be passed in the array of parameters. The parameters must appear as they would appear if they were actually supplied as method parameters. That is, if string values are supplied, then the elements being passed must have leading and trailing double-quote characters. These double-quotes would be in addition to the double-quotes used to define the elements as text.

For example, to pass a real value as the first parameter, a variable name as the second, and a text value as the third parameter you would use a three-element process text array, such as var1Text, and assign the elements as follows:
var1Text{1}:=”1999.1230”
var1Text{2}:=”MyVar”
var1Text{3}:=ASCII(double quote)+”Some string value.”+ASCII(double quote)

You would then pass a pointer to var1Text as one of the parameters in the call to _SYHdlProcess.
-
Checks available memory before starting new processes.

_SYHdlSchbyLay

Handles automatic checking and unchecking of search criteria in Search by
Layout-type dialog. Also erases contents of search fields when check box is
manually unchecked.
-
Takes action code, pointer to variable containing search criteria
(or 1st radio button), pointer to check box to turn search on this
criteria on and off, pointer to 2nd radio button (optional). Returns error code.
-
Generic method that can be used to handle button actions on dialogs used
for specifying searches.

_SYHdlSeltPop

Performs array management for selection of an item from a pop up array of alpha, numeric, or date types. Call from a pop up's script. No additional actions are triggered.
-
Takes a pointer to the array, and a flag indicating whether or not to redraw the screen.
-
It places the number of the element chosen in the 0th element of the array. If no element is chosen the method restores the array's value to that stored in its 0th element.
If the array is text based the number is stored as characters, if it's date based it's stored as the day number. In this case the 0th element will only store a reference of up to the 31st item.

_SYHdlSet

Set handling method with methods to perform different actions.

-

Takes parameters specifying the action, a pointer to the table acted on, name of set to act on or with, name of the set to intersect with for screening purposes.

-

• Set_Dialog: asks user what operation to perform (i.e. save to disk, read from disk, delete from disk)
• Delete: delete a set file from disk
• Save: save a set to disk.
• Load: load a set from disk, asks for further instructions on what to do with the current selection (i.e. replace, add to, etc.)

_SYHdlSRTDirct

Handles generic Sort dialog's response to the User's selection of sort order.

-

Takes pointers to the sort direction flag variable, which is one of the variables storing perpetual sort information. Also a pointer to the sort direction indication button.

-

Sets button text according to passed instructions. No sorting is performed.

_SYHideUserWind

Hides the User/Runtime window behind the menu bar.

-

Takes no parameters.

-

_SYHistorySumry

Sets up the dialog used to display historical information passed in arrays.

-

Takes variables indicating the window title, display text, pointer to date arrays, pointer to the real array of consecutive monthly debits, pointer to real array of accumulated debit balances for each month.

-

Requires free access to about half a dozen generic arrays of all types.

_SYHndlPrtSpecs

Called in conjunction with the generic print routines. This method makes changes to the report selection dialog in response to a selected report.

-

takes not parameters, using the layout's global variables instead.

-

Handles the display of the report options associated with each report as specified by the user when the generic dialog was called.

_SYIncrmntAlpha

Takes any alpha string and increments it to “the next higher” value.

-

The algorithm increments only alphanumeric characters in the ranges 0-9 and A-Z. Lower case characters are converted to upper case before incrementing. Characters outside of this range are ignored.
Right-most characters are converted first. The range of characters from 0-9 and A-Z is considered one, large cycle. Character value of 9 is converted to A, character value of Z is converted to 0. When the character cycles to 0, the next character to the left in the string is then incremented. An extra

character will be added if necessary, as in the case where “ZZz” is incremented to “1000”.

-

Can be modified to return only a string with only the incrementable characters (placeholders stripped out). For example, if passed “abC-999/9” it would return “ABC999A”. It can also be given a maximumstring length.

_SYIncrmntSeqNm

Increments one of the ID numbers stored in the data file.

-

Takes the code used to distinguish the various ID number records and the quantity ID values that will be assigned. Returns the ID number, or a value < 0 as an error code.

-

This assumes that the user has obtained clearance for the ID update by locking the appropriate [ProcedureLock] record. If multiple ID numbers are needed the counter is incremented by the corresponding number.

_SYInputReject

Called to handle processing when an input layout has been unable to save it's record.

-

Takes no parameters.

-

Call from input layouts when an automatic action needs to be halted.

_SYInsrTxtInStr

Replaces highlighted text with specified string.

-

Takes pointer to object string with highlighted characters, also takes the text
source with which to replace the selected records.

-
If no text selected this inserts the source text at the object text's insertion point.

_SYLayoutDsply

Called by all layouts to set screen color buttons.
-
Takes an instruction code distinguishing the three types of layouts: input, output, and dialog.
-
Also triggers the output layout resize button which calls the output resize script.

_SYLayoutPhase

Called to execute standard code for layout phases such as Activated, Deactivated,
and Outside call.
-
Takes an action code, indicator of calling layout method's type (input,
output, or dialog), pointer to the layout's table, and pointer to a text
variable that will be assigned any message that is received from another
process.
-
Checks the interprocess mailbox for messages sent to
the calling process. Takes care of handling v4QExitProcs messages.

_SYLoadSet

Presents user with different options for loading a set.
-
Takes as parameters the name of the set, a pointer to the set's table, and the name of a set that can be used to intersect with the first set for screening.

-

Opens a dialog for the user to chose how to apply the named set to the current selection (i.e. replace, add to, etc.)

_SYLogHistory

Handles the creation, deletion, examination, display, and printing of system log records. These are records that are stored in the [LogHistory] and are created to record significant system events, such as daily maintenance, periodic events like posting to the GL, and system errors. These records are inaccessible to the user and can only be viewed, edited, or printed from the Maintenance screen.

-

Takes as parameters an action code whose recognized values are Check_Interval, Show_History, Add_History, Delete_History, Print. Also takes a pointer to a text array whose variable number of elements are used to pass parameters into the method. Returns a longint which is assigned various positive values depending on conditions.

-

The system automatically calls the _SYLogHistory method whenever _SYAlert displays a message that contains the word “error.” Log history records are dated and are only preserved for a limited time before being discarded. The time that they’re retained is set by the administrator.

_SYMakeDiskFile

Creates a text file, of sepcified name, on disk.

-

Takes no parameters, returns a document reference parameter.

-

Presents the system’s file management dialog enabling user to enter a file name.

_SYMakNewTxtRec

Called to create a new record in the [fTextRecords] fie.

-

Takes a longint key value and a text identifier.

-

The key value holds together records related to the same job, the text identifier is used for additional specification. [fTextRecords] with ID <0 are reused if available.

_SYMBAbout

This method displays the applications About box.

-

Attached to the application's About menu item. Takes no parameters.

-

This is installed as the system's default About box handling method.

_SYMBCntrlScren

Called from the menu bar to reenter or to exit the Control Screen loop.

-

Called from the Designers menu. Takes no parameters.

-

Asks the user which action they want. Accessed through the menu by developers who return to the runtime environment and want to redisplay the Control Screen.

_SYMBGoCtrlScrn

Called from other processes to bring the control screen to the foreground.

-

Takes no parameters.

_SYMBReports

Called from list screen menu bars to open the report selection dialog appropriate for the current list.

-

Issues the Command+P (Mac) or Ctrl+P (Win) keystroke. Takes no parameters.

_SYMBSchByExample

Called from list screen menu bars to open the search by example dialog appropriate for the current list.

-

Issues the Command+E (Mac) or Ctrl+E (Win) keystroke. Takes no parameters.

_SYMBFocus

Called from list screen menu bars to focus the selection down to the currently highlighted records.

-

Issues the Command+F (Mac) or Ctrl+F (Win) keystroke. Takes no parameters.

_SYMBSort

Called from list screen menu bars to open the sort dialog appropriate for the current list.

-

Issues the Command+S (Mac) or Ctrl+S (Win) keystroke. Takes no parameters.

_SYMBSets

Called from list screen menu bars to open the sets dialog.

-

Issues the Command+U (Mac) or Ctrl+U (Win) keystroke. Takes no parameters.

_SYMBPageSetUp

Displays the Print Settings.

-

Menu call, takes no parameters.

-

Executes the Print Settings command.

_SYMBQuit

Hanldes the Quit menu item.

-

Takes no parameters.

-

Tests the current user and provides Designer the option of returning to the User environment. Also writes the preference file to disk.

_SYMessage

Writes a message to the message window.

-

Takes the message to write and instructions on whether to open a new window and whether to erase current contents of the message window.

-

Wraps text to a specified line width. Displays message in a custom dialog. Closes the window if a blank message is passed. _SYMessage is not part of 4th Quarter's interprocess message system.

_SYModRcrd

Opens a record modification window for an indicated table.

-

Takes a pointer to the table, name of input layout to be used for record modification, name of input layout to be restored afterwards.

-

Replaces the current selection after the modifications have been accepted or canceled. Does not restore the record order.

_SYMonNameNum

Takes either a month's number and returns its name, or vice versa.

-

Takes a pointer to a string containing a month's name, and a pointer to a longint variable storing a month's number. Which ever is not supplied should be set to nil.

-

Acts on which ever pointer is not nil, translates the passed value either into a word or into a number, and assigns the value in the location indicated by the other pointer.

_SYMoveAraHiLit

Move the indicated element of the indicated array up or down within the array.

-

Takes a positive or negative number giving the number of places to move the element, a pointer to the array being affected, a special processing code, and a boolean indicating if cyclic movement is allowed.

-

_SYMoveAraRow

Moves the indicated element up or down in up to 12 arrays at once.

-

Takes number of places to move the elements, the array element to move, a special processing indicator value, and pointers to up to 12 arrays.

-

Will accept any number of pointers to arrays as long as there is at least one. Does not perform cyclic permutations.

_SYMrkAssocMenu

Places or removes a check mark from the 1st or 2nd associated menu.

-

Takes the number of the menu and item, whether to enable or disable, and the characters to use for marking.

-

Will apply instruction to one or to all items in specified associated menu. Can be used to apply blanks to all menu items to remove all check marks.

_SYMsgReceive

This is one of the systems core message-passing routines. It is called to receive a message addressed to the current process. It takes an action parameter, a pointer to a text variable and a pointer to a longint variable. If there is a message for the current process its text value is placed in the text variable indicated by the passed pointer. The indicated longint variable is then assigned the sender's process number.

-

Messages can only be passed between processes on the same machine. Messages are handled by the system post office process. This process is handled by the method _SYMsgCentral. Refer to this method and to _SyMsgSend for more information on message passing options.

-

A Developer's Note is planned for Spring 1999 on the subject of 4th QUarter's message system.

_SYMsgSend

This is one of the systems core message-passing routines. It is called to send a message addressed to the indicated process. It takes an action parameter, the text of the message, and the process number to which it is to be sent. It also takes a message status parameter indicating how the message is to be handled.

-

There are two options for sending messages: standard messages are sent without return receipt notification, while RSVP messages are sent with return receipt notification. To send standard messages pass a message status value of zero. To send return receipt messages pass the parameter ◊vSYMsgRSVP as a status value. This variable acts as a system constant whose value is recognized by _SYMsgSend.

-

When messages are sent, the system post office will repeatedly attempt to deliver them to the addressed process. Messages will be discarded if the addressed process does not exist, or if the messages cannot be delivered within a certain time limit that's set in the _SYMsgCentral method. If a message has been sent with RSVP then a return message will be sent to the sender once the message has processed, either by being received or by being discarded.

_SYMsgUtility

This is one of the systems core message-passing routines. It is not called by the programmer directly. Rather, it controls the post office process that's started when the first message is sent. The post office process remains in memory in a paused state throughout the 4th Quarter session and is unpaused whenever messages need to be delivered.

_SYNewProcess

This method is normally called only by the _SYHdlProcess method in order to ensure that sufficient memory is available before starting a new process. However it can be called directly.

-

Takes an action code, method name, stack size, process name, and a pointer to a text array whose elements are passed as parameters to the newly created process. Refer to the description of _SYHdlProcess for details on use of the parameter array.

-

Returns the number of the newly created process or, if unable to create a new process it returns 0. If an error occurs a negative value is returned.

_SYNil

Called to determine if the passed parameter is a nil pointer. Replaces 4D
NIL command, which does not work in some cases.
-
Takes a pointer to a field or variable. Returns TRUE or FALSE.
-
Tests both whether the passed pointer points to the value that 4th Quarter
uses to indicate nil, and also whether the passed pointer is recognized as
nil through the use of the NIL command.

_SYOECReturnErr

To be installed as an Error event handling method.
-
Takes no parameters.
-
Places the value of the systems Error variable in the variable ◊vError to that it can be accessed by other processes.

_SYOnErrCall

Encapsulation of 4D's ON ERR CALL command.
-
takes name of method to install, and special instructions
-
Installs and deinstalls On Err Call method while using a IP variable for keeping track of the number times error handling has been installed. This prevents a second process from deinstalling the error handling of the first.

_SYOnEventCall

Encapsulation of 4D's ON EVENT CALL command.
-
takes name of method to install, and special instructions
-
Installs and deinstalls On Event Call method while using a IP variable for keeping track of the number times event handling has been installed. This prevents a second process from deinstalling the event handling of the first.

_SYOpenProcess

Called to open a process or wake up a process that may be paused.
-
Takes an action code, process name, size, and method, whether multiple
instances are allowed, any message to pass to the process, and message
handling instructions.
-
Prevents multiple instances if specified to do so. Can perform various
actions on the process by performing one of the message passing options.

_SYOpnWindow

Encapsulates 4D's OPEN WINDOW method.
-
Takes the width, hight, type, title, and distance down from top of screen. Keeps track of incrementing the window counter. Stacks new windows if indicated as a user default.
-
Returns the reference number of the opened window, or a negative value if an error occurs.

_SYParseAlpha

Called to extract an alphabetic or a numeric part from a passed string value. Takes an action code, a string value to be parsed, a pointer to a field or variable that will be assigned the result of the parsing operation. Returns an error code.

-

The substring to be extracted can either be located by parsing that begins either on the left or right of the passed string. The default action is to start parsing at the right and moves left.

-

Takes the action codes “Return_Numeric” or “Return_Alpha”. A 0 is assigned to the pointer variable if no embedded number is found, an empty string ("") is returned if no embedded string is found.

_SYParsePath

Called to strip out either the file name or the preceeding directory heirarchy from a full path name.

-

Takes an action code, the directory separator symbol, the full path name, and a pointer to a text variable. Returns an error code.

-

Either the file name, or the preceeding directory is returned in the text indicated by the passed pointer. If the directory symbol is not found in the full path, then the full path is assumed to be equal to either the file name or the directory.

_SYPhoneFormat

Reformats passed string as a telephone number of indicated type.

-

Takes pointer to text or string field or variable, and phone number code.

-

Type code is trapped in a "Case of" statement and handled accordingly.

_SYPopAllOffStk

Pops all records off the stack for a particular table.

-

Tales a table pointer.

-

Keeps popping records off the stack until there are no more.

_SYPrcsAttribs

This is an extension of 4D's Process Attributes call that offers more flexibility and returns more information.

-

Takes an action code, a process number, and four pointers to variables that are assigned the process name, state, cumulative time in ticks, and an operational description.

-

The value assigned to the description field can be tested to determine whether the indicated process is of system origin or if it is a process that operates under programmatic control.

_SYPrcsMessage

For sending and receiving interprocess messages.

-

Takes an action code and pointers to two variables. If the action is sending a messages the pointers reference the ID of the process being addresses and the message being sent. If receiveing the ID returns the sender's ID and the message that was sent.

-

A passed "mode" variable determines how to manage the process and and error code is returned.

_SYPrtAdrLabels

Prints records in the [Address] table that are related to current selection of records in another table.

-

Takes action indicator code, pointer to ID field in related table on which to perform the action indicated. Also takes special instructions code.

-

Recall that [Address] records are distinguished by related table, related record ID, and by address type. Refer to the actual method for details.

_SYPrtHistAray

Prints historical records stored in predefined arrays using PRINT LAYOUT.

-

Takes various booleans determining whether to print value, or a graph or period contributions or running balance. Takes 3 column headings.

-

Draws on information stored in numerious system arrays. Used in conjunction with the [System_Default].qShowHistory_d layout.

_SYPrtStandard

Encapsulates 4D's PRINT SELECTION and EXPORT commands applied to a particular table.

-

Takes instructions of whether to print to paper or disk, the disk format, the paper orientation, the names of the layouts to use, and a pointer to the table to act on.

-

Handles standard disk and paper printing options that do not require customization. Does not initiatiate any break processing.

_SYQuitLoop

Called in a new local method to send messages to all processes requesting
them to exit. Process then waits for them to exit and, if successful, issues a

Quit 4D Command to exit the application.
-
Takes no parameters.
-
Manages such things as making sure that only one Quit process is active,
and monitors whether remaining windows remain open after a certain length
of time. If, after a certain "time-out" period, other processes do not exit,
then the _SYQuitLoop process exits without quitting.

_SYReadFileToAr

Called to read all, or a fixed number of records from a disk file to a specified text array.
-
Takes parameters indicating what file to open or whether to query the user to select a file, what array to place each whole record into, what character to use as record delimiter, and whether all or a fixed number of records should be read.
-
Allow the user to open a file, or will open a file specified by name or document reference, and read all or a fixed number of records into into an array. Additional elements are added to the array as needed.

_SYReadOnlyAll

Called to set all tables to Read Only state. This prevents records from becoming locked when they’re loaded for the purposed of looking up information.
-
Takes no parameters.
-
This is called when first entering a user area.

_SYRedelOldDel

Called to search for records that should have been deleted but are still in the DB and then to delete them.

-

Takes a special action code and a field pointer to the boolean "deleted" field.

-

Called mostly for DB maintenance tasks or similar tasks that might use the indicated table's "deleted" field.

_SYRemovChars

Removes characters from a string according to the action specified.

-

Takes an action code, the original text and passes back the modified text.

-

Its actions include removing only beginning and trailing blanks, removing all blanks, removing all numeric characters, removing all nonnumeric characters.

_SYRsrvdCode

Handles assignment of the reserved system suffix to create unique key field values.

-

Takes an action indicator code and a text value such that when the reserved suffix is appended the result is an unique key field value. Also takes pointers to field used to store key value.

-

When user is allowed to assign their own key value this method assigns a suffix reserved for system use.

_SYSchEngine

Executes the Search command according to the parameters passed.

-
Takes an action code, a pointer to the field to be searched, the search value in text form, a search specifier (“equal to”, “contains”, etc.). Returns an error code.
-
This method will perform one of a dozen differently specified searches on alpha, numeric, date, or boolean field.

_SYSchInColumn

Is triggered by the quick search buttons below output layout columns. Displays
the search dialog, performs the search, and places records in set.
-
Takes an action code, pointer to field to be searched on, name of field to be
displayed in layout, name of set for located records. Returns an error code.
-
Returns 0 if search is canceled, 1 if search was successful.

_SYSchPastDates

Searches a specified date field for records in a specified date range.
-
Takes a pointer to the date field to be searched, a starting and an ending date value, the name of a set in which to place the results, and the name of a set to be used to form an intersection with the records located. Returns an error code.
-
The method searchs the table associated with the date field supplied. The date range includes the starting and ending date. If a blank name is passed for the intersection set, then no intersection is performed. The intersection, if performed, affects the records in the passed set and the current selection.

_SYSchTypeAdres

Searches [Address] table for addresses linked to specified table and ID field, and of specified type(s).

-

Takes action indicator code, table number, record ID number, 1 to 5 different address types. Returns the number of address records found

-

Establishes a selection of the specified address records.

_SYSeconds

Gives the number of seconds in different time frames: either seconds in
the current day, or seconds since some reference time in the past.

-

Takes an action code and also an optional "*" indicating that times should
be read from the server's system clock.

-

_SYSELAddrsList

Displays a text array of addresses to allow a user to choose 1 address.

-

Takes pointers to ID field in parent table, longint to be assigned the address record number, variable to be assigned the full address, and title to be displayed. Returns 1 if items chosen, 0 if nothing chosen.

-

Used to handles selection for records that are assigned multiple addresses.

_SYSelFromArray

Makes current selection of referenced table correspond to the record references stored in an array.

-

Takes pointers to a key field and to an array that contains key field values. Setting afords choice of whether or not to install On Event Call method. Returns number of records in selction.

-

Array type must match field type.

_SYSELinTextAra

Displays a text array for the user to choose and item.

-

Takes pointers to text array, variable to be assigned chosen element's value, variable to be assigned chosen element's number, window title, and text to display below list. Returns value of 1 if an item is chosen, 0 if not.

-

Overwrites several general system arrays.

_SYSeInToSeIn

Creates a selection in one table according to values stored in records in another table.

-

Takes a pointer to the relating field in the table where the selection is to be established, and a pointer to the relating field in the table where the selection already is established. Returns the number of records in the newly established current selection.

-

This handles both 1-many, many-1, and many-many related tables. No table linking is required. Types of the two fields must match.

_SYSetArayMrk

Called to set or remove a mark stored in an array element.

-

Takes a character used as a mark, pointer to the array clicked in, pointer to the array whose first value can hold the mark.
-
Mark array must be string or text. If first character is the mark then it is replaced with a blank, otherwise the array element is set to equal the mark.

_SYSetCanclKey

Called to install or remove event monitoring for the Command-. combination as a trigger for setting the ◊vUserAbort variable to TRUE.
-
Takes action indicator code, text message to display, special processing code.
-
Turns event handling on and off, monitors the number of times event handling has been turned on to insure that it doesn't turn it off in one process when being used in another.

_SYSetDateRange

Called to solicit a date range from the user.
-
Takes pointers to starting date variable, ending date variable, and takes a string to be used as explanatory text. Returns 1 if the user pressed the ENTER button.
-
Opens a dialog that's an entry screen for 2 date variables.

_SYSetDelimiter

Asks the user to set delimiter characters that are used when writing to disk.
-
Takes no parameters.
-
Opens the [fDialog];qPrintDelimit_d window for user entry.

_SYSetFilChgMrk

Locates a [ProcedureLocks] record according to a code and sets its time counter to the current time.

-

Takes pseudophore identifier string. Returns 0 if record could be accessed and set.

-

Creates a new record if specified one isn't found. Will do nothing if record exists but is locked.

_SYSetPrcdLock

Called to load a [ProcedureLocks] record in read/write mode. This is used to indicate a access to a certain function.

-

Takes a pseudophore name, returns TRUE if loaded in Read/Write mode or FALSE if Read/Write access could not be obtained.

-

Creates the indicated record is it doesn't exist. If the record is already locked the method keeps trying to gain access until the user cancels the operations.

_SYSetPrtSimple

Displays the simplest print selection dialog for user to press Accept or Cancel.

-

True/False choices for allowing export in Text file format. Takes paper orientation and a pointer to a variable that indicates what format will be used in writing to disk.

-

Asks whether the user wants to print to paper or to disk, and if to disk what format to write in.

_SYSetPrtSpecs

Called by the standard print selection dialog when user changes the selected array item.

-

Action value indicating action, name of report, report orientation, print format, name of layouts to use for Print Selection and for Export methods.

-

Asks whether the user wants to print to paper or to disk, and if to disk what format to write in.

_SYSetWindSize

Encapsulates the window management external package. Takes instructions and sets the window size accordingly.

-

The instuction code tells the method to resize the current window if it is greater than, not equal to, or less than indicated width or height. It also takes an optional "*" parameter which indicates that the result of any resizing operation should keep all parts of the window visible. Refer to the method itself for explanation of how to construct this code.

-

Returns the value 1 if the window has been resized.

_SYShoDupOption

Called to display duplicate checking search criteria for user's selection.

-

Takes pointers to pointer array with pointers to fields to search on, text array with field lables, text array with value to search on. Pointer to a boolean array storing whether or not to search on the corresponding fields.

-

This method does not actually do the uniqueness search.

_SYShoNAskDups

Called to display the duplicate records that have been located.

-

Takes text message, pointers to 1st through 6th fields to display.

-

Concatenates the contents of the arrays to create a single array element for display.

_SYShowUserWind

Opens the User/Runtime window after it's been hidden behind the menu bar.

-

Takes no parameters.

-

_SYShowProces

Called to display and bring a process to the foreground. Acts only on currently active processes.

-

Takes an action code and a pointer to an array in which process ID's are stored.

-

If the process is paused and hidden it's activated and displayed, if it's just in the background it's brought to the foreground.

_SYSrtManyLevl

Generic sort method that handles up to 3 sort field each sorted on their own direction.

-

Takes table or subtable to be sorted, pointers to sort variables that store pointers to the 1st, 2nd, and 3rd fields to sort on. Also takes 3 indicators of sort direction.

-

Handles multiple criteria table sorting, queries the user if the selection is too long.

_SYTestFilePath

Called to test whether the specified file path indicates the existence of a file on disk that can be opened.

-

Takes file and/or volumn path string. Returns OS File Manager error code, if one occurs.

-

_SYTestEqualSet

Called to tell if two sets have the same elements.

-

Takes names of the two sets to compare. Returns an indication code.

-

Checks for set errors and reports when they occur.

_SYTotAmtInSEL

Performs a summation over a referenced numeric field in the current selection.

-

Takes pointers to the field to be summed, the text variable where the sum is displayed, and a real variable assigned the sum value.

-

Sum is displayed as a string according to the formatting instructions in the global variable "vSubTotFmt."

_SYTransaction

Encapsulates 4D's Start Transaction, Validate Transaction and Cancel Transaction commands.

-

Takes code indicating which action to perform. Returns error code that must be monitored to know whether the action was performed.

-

Keeps track of whether a transaction is already in effect and prevents a second transaction from starting.

_SYTransposElms

Exchanges elements in a set of indicated arrays. .

-

Takes an action code, a boolean indicating whether to exchange with previous or following element, and pointers to up to six arrays of any type.

-

Used to move the contents of a row up or down when displayed using grouped arrays. Does not return an error code.

_SYTransposElms

Called to transanpose the elements in the 1 to 6 passed arrays. The arrays
can be of any type.

-

Takes an action code, boolean element indicating whether to move elements up
or down, six pointers to arrays. Only one pointer is required.

-

Value of the first array pointed to indicates what element is to be transposed.
Used to transpose rows of grouped arrays.

_SYTransposRows

Reverses the order of two displayed record is they support special sort fields.

-

Takes a pointer to the sort field, instructions on whether to move selected record up or down, pointer to variable that stores previously selected record's number.

-

To by used here a table must have a field used only for establishing display order.

_SYUNLoadAll

Called to unload records, remove error and event processing, and cancel any on-going transactions if only one process is running.

-

Takes an action code that is the concatenation of table numbers, passed as strings, each separated by the "/" character. Tables whose numbers are passed in this action code are not unloaded.

-

This is called before returning to the Control Screen and unlocks any records still loaded.

_SYUserDflt

Reads from and writes to the 4Q_Preferences file stored in the Preferences
folder inside the System folder.

-

Takes an action code, return an error code.

-

This disk file is used to store user preferences.

_SYUserDfltAray

This creates arrays that store the names of the variables used to store User

defaults.
-
Takes pointer to a text array containing variable names, and another text array
storing variable types.
-
These arrays are used during Start Up. If you add your own user preferences you
need to specify them here.

_SYUserInGroup

Acts as a wrapper around 4D's User in group command. Tests whether the specified user is a member of the specified group.
-
Takes three parameters: an action code, a user name, and a group name. Returns and integer set to 1 if the user is in the group, 0 if they're not.
-
Since the end user can redefine the name of users and groups, this method allows the developer to control how the User in Group test is handled.

_SYWait

Called to delay execution.
-
Takes as a parameter the number of ticks to wait. Returns after that has elapsed.
-
Call this method inside of loops that would otherwise repeatedly pole the network or waste processing time.

_SYWorkArounds

Performs various actions required to work around bugs in 4D, the Mac OS, Windows, or other software used by 4th Quarter.

When $2="WinListRedraw" this method performs a screen refresh needed to properly display the contents of the footer area in list screens on Windows machines. Nothing is done if the machine is a Mac.

-

Takes the name of the calling method, an action code, and an optional pointer. Returns an error code.

-

Refer to comments the actual method for details on what actions it performs.

_SYWrapText

Performs text wrapping by inserting carriage returns between words to insure
that lines do not exceed a specified length.

-

Takes pointer to text, action code, and the maximum line length.

-

• General_case: Considers each block of text that's does not contain any carriage returns and inserts carriage returns between words.
• Single_line: Does not look for carriage returns in existing text and merely inserts carriage returns between words in the string so that no line exceeds the specified length.

_SY_IDRecord

Creates and updates the ID sequence number records in [ID_Number] table.

-

Takes string indicating action to perform, the sequence number of the ID records, the value of the next ID number to assign to the ID record (when appropriate), and a pointer to the field that is assigned this ID number.

-

This method only creates or resets the ID counters, it is not used to increment them.

CHAPTER 9

Change History

Record of changes made to the source code.

Updates

05/18/00

Methods Modified

COMPILER_4QpSYe
compiler_Longint1
compiler_Longint2
__SY_InitSysVar
_SHAddrHdlSchPop
_SYCtrlPanel_d
_SYCtrlScreen
_SYEnablMenu
_SYHdlSet
_SYHideUserWind
_SYMBAbout
_SYMBFocus
_SYMBReports
_SYMBSchByExample
_SYMBSets
_SYMBSort
_SYPrcsAttribs

Forms Changed

F-[Address] qAddress_o

Form Methods Changed

FM-[Address];'qAddress_o'

Menu Bars Changed

MB#5 - new (blank)

MB#6 - new

MB#7 - new

02/21/00

Updated all bground tbutton objects to remove the command-shift-q key-stroke that was interfering with the backdoor button's keystroke.

10/24/99

Methods Deleted

_ADHdlAcDfltPgs

_ADHdlsfRec

_ADSelDoc4Imp

CHAPTER 10

Resources

This section lists other sources of information for using 4Q Shell and 4th Quarter Accounting. Technical support and written materials are available from Braided Matrix.

Getting Help

Starting Simple

It is difficult to begin a project whose initial target is the management of non-accounting information and whose ultimate goal is the incorporation of accounting functionality. However, this is commonly what businesses need.

In most situations the business that engages in software development understands their basic business model better than they understand what they need in the way of computer accounting. It is often more attractive to begin programming a simple business system and to add accounting at a later stage.

4Q Shell is designed for just this situation. Using 4Q Shell will enable a 4D programmer to focus on building a basic application knowing that it can be extended later to include accounting.

A Two-Tiered Effort

In addition to licensing the source code for 4th Quarter Accounting Solution, we at Braided Matrix also provide design and programming services for businesses working to develop applications using 4th Quarter Accounting.

It often happens that our clients manage the development of that aspect of their application that handles nonaccounting data. They then come to us for help integrating this work with 4th Quarter Accounting. This results in a very successful two-tier development effort. The distinct nature of accounting in business, and the modular design of 4th Quarter Accounting, make this possible.

We can also help those using 4Q Shell in understanding how to get started and how to put together strategic development plans. Please call us for our current hourly consulting rates.

Additonal Documentation

Database development is a remarkably complex area of computer technology requiring a comprehensive knowledge of computer programming, interface design, business rules and software design.

Few people know enough to author a complete business system, and those who do don't have the time to complete the task themselves. As a result, we find that every job we've done has required extensive communication and education.

For the past ten years I've been working to make the learning process easier by creating educational materials. The more general of these writings have appeared in Dimensions Magazine and 4D-Zine, while the more specific topics have remained part of the 4th Quarter Accounting technical documentation.

All of this material is now available from us by email. It will soon be available directly on our web site.

All the materials listed here apply to 4th Quarter Accounting Solution. Those documents dealing specifically with accounting do not apply to 4Q Shell, but they would be worthwhile to anyone using 4Q Shell in preparation for the later incorporation of accounting.

Database Design

4D vs. SQL

Strategies for Complex Projects

The Debugging Series I: Design

Working Toward Step One, a method for deriving a 4D file structure

Database Programming

Group to Group Relations

Implementing a Multi-Window Choice List in Version 3

Managing Multiple Record Entries, Parts I & II

Maintainable Code I: Naming Conventions

Maintainable Code II: Clarity

Maintainable Code III: Stable Code

A temporary Subrecord Method for Entering Related-Many Records

Database Accounting

Accounting the OO and RDB Way, Part I: Concepts

Accounting the OO and RDB Way, Part II: Methods

Accounting the OO and RDB Way, Part III: Application

Accounting Transactions I: Debits & Credits

Accounting Transactions II: Cash Transactions

Accounting Transactions III: Batch Processing

Accounting Transactions IV: Invoices

Database Accounting 101a

Structure of Accounting Systems: How to Store Accounts

4th Quarter Accounting Product Description

4 Screen Shots

4Q's Table Structure

Dynamic Allocation in 4Q — AR & AP

4Q's Features & Benefits

4Q Printed Tour — Accounting

4Q Printed Tour — Inventory

4Q Product Description

4Q Product Prices

A Sample of Source Code for 4Q

4Q Specifications

Virtual Invoices in 4Q

4th Quarter Accounting Developer Notes

Notes that are underlined apply to both 4Q Shell and 4th Quarter Accounting. All other notes apply only to 4th Quarter Accounting.

#2 — Procedure Call Tracing

#3 — Naming Conventions

#5 — List Management Screens

#6 — Use of the Menu Bars

#7 — Password Protection of Accounting Functions

#8 — The Quit Loop

#9 — Handling Multiple Processes in 4Q

#11 — Version Numbering and Version Control

#12 — Assigning ID's and Updating Counters

#13 — List Dictionary

#14 — Modifying 4Q Procedures Using Patching

#15 — Types of Transaction Records

#16 — Using 4D Insider Libraries

#17 — Invoice Transactions

#18 — Application Version Numbering

#19 — Managing Transactions

#20 — Default Accounts

#21 — Inventory Accounts

#22 — Replacing Boolean Fields

#23 — Assigning Accounts

#24 — Messaging

CHAPTER 11

Index