Affordable, Adaptable ERP Software

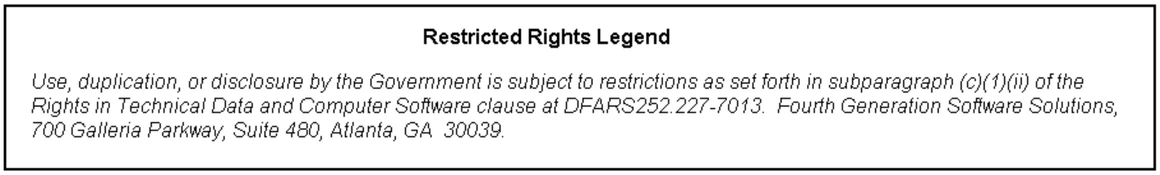
VDT Screens and Menus

Course Workbook

Version 5.40

Fourth Generation
Fitrix™ Visual Development Tool (VDT)
Screens and Menus Course Workbook

Version 5.40



Copyright

Software License Notice

Licenses and Trademarks

FITRIX MANUALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, FURTHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE FITRIX MANUALS IS WITH YOU SHOULD THE FITRIX MANUALS PROVE DEFECTIVE, YOU (AND NOT FOURTH GENERATION SOFTWARE OR ANY AUTHORIZED REPRESENTATIVE OF FOURTH GENERATION SOFTWARE) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR, OR CORRECTION IN NO EVENT WILL FOURTH GENERATION BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF OR INABILITY TO USE SUCH FITRIX MANUALS, EVEN IF FOURTH GENERATION OR AN AUTHORIZED REPRESENTATIVE OF FOURTH GENERATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY IN ADDITION, FOURTH GENERATION SHALL NOT BE LIABLE FOR ANY CLAIM ARISING OUT OF THE USE OF OR INABILITY TO USE SUCH FOURTH GENERATION SOFTWARE OR MANUALS BASED UPON STRICT LIABILITY OR FOURTH GENERATION'S NEGLIGENCE SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS, WHICH VARY FROM STATE TO STATE

Fourth Generation Software Solutions
700 Galleria Parkway, Suite 480
Atlanta, GA 30339
http://www.fitrix.com

Corporate: (770) 432-7623
Fax: (770) 432-3447
E-mail: info@fitrix.com

Welcome to the Fitrix, Visual Development Tool (VDT) Screen and Menu Course Workbook. This manual is designed for use in the Fitrix VDT Training class. We hope that you find all of this information clear and useful.

Please keep in mind that programs created by the Visual Development Tool are graphical programs. The Visual Development Tool itself runs only in character mode.

The programs created by the Visual Development Tool are written in FourJ's Genero Business Development Language. This is a fourth Generation Language and is shortened to just 4GL in this workbook.

We hope you enjoy using our products and look forward to serving you in the future!

Table of Contents

Chapter 1

Using a Generated Input Program

Main topics:

- Setting Environment Variables
- Using a Generated Input Program
- Using Toolbars
- Accessing Zooms
- Using AutoZooms
- Using Lookups

Setting Environment Variables

In order to create and run programs with *Screen*, you must set certain Linux environment variables and export them. These variables are set and exported by the startup script that is executed when you login using the click the Fitrix Icon on your desktop.

1. The `$fg` variable should point to the directory where your *Screen* product is installed. For example, the following command sets `$fg` to the `/fitrix/fx_dev` directory:

   ```
   fg=/fitrix/fx_dev ; export fg
   ```

2. The `$INFORMIXDIR` variable should point to your `informix` directory. For example, the following command sets `$ INFORMIXDIR` to the `/usr/informix` directory:

   ```
   INFORMIXDIR=/usr/informix; export INFORMIXDIR
   ```

3. The `$PATH` variable should include both `$fgtooldir/bin` and `$INFORMIXDIR/bin` directories:

   ```
   $fgtooldir/bin
   $INFORKIXDIR/bin
   ```

4. The `$DBPATH` variable must include two additional `$fg` directories:

   ```
   $fgtooldir/lib/forms
   $fgtooldir/codegen/data
   ```

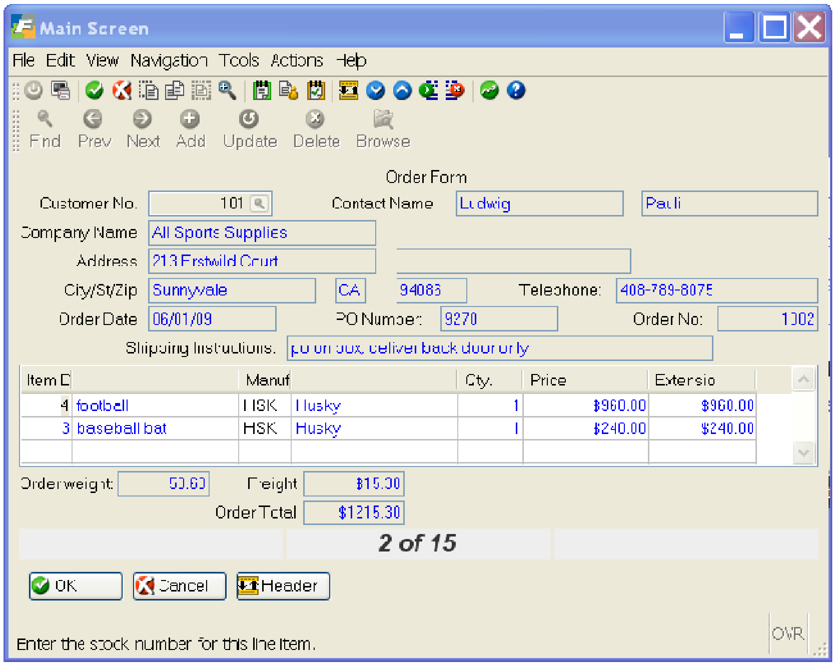
5. The `$FGLDBPATH` variable should point to where you keep the database schema files. For example,

   ```
   $FGLDBPATH=$fg/data; export $FGLDBPATH
   ```

Using a Generated Input Program

Fitrix VDT Application Code Generator lets you create sophisticated input programs. The following figure illustrates an input program built by *VDT Application Code Generator*.

You can create sophisticated input programs with Fitrix VDT Application Code Generator.


Main Screen
File Edit View Navigation Tools Actions Help
Find Prev Next Add Update Delete Browse
OK
Cancel
Header
OVR

Order Form

Customer No. 101 Contact Name Ludwig Pauli

Company Name All Sports Supplies

Address 213 Frstwild Court

City/St/Zip Sunnyvale CA 94083 Telephone: 408-789-8075

Order Date 06/01/09 PO Number: 9270 Order No: 1002

Shipping Instructions: po on box, deliver back door only

Item D		Manuf		Qty.	Price	Extensio
4	football	HSK	Husky	1	$960.00	$960.00
3	baseball bat	HSK	Husky	1	$240.00	$240.00

Orderweight 50.60 Freight $15.30

Order Total $1215.30

2 of 15

Enter the stock number for this line item.

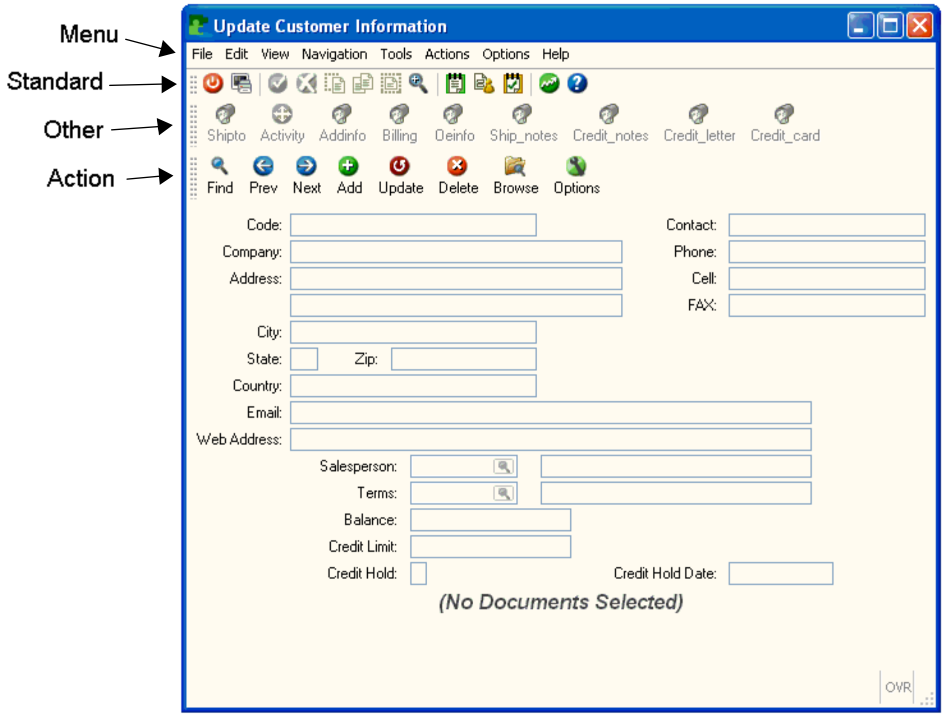
All input programs contain a toolbar interface located at the top of the window.

Using the Top Menu and Toolbars

You use the tool bars to tell the program what commands to perform. For example, you can tell the program to access help, add a record, delete a record, or exit a program.

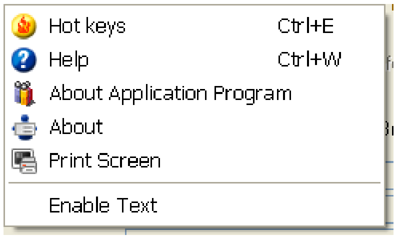
There are three toolbars in the Fitrix generated programs and some may contain a fourth toolbar. The options available on each toolbar may vary depending upon what type of input program you are in and whether you are in update or view mode.

Toolbars:


Menu
Standard
Other
Action
Update Customer Information
File Edit View Navigation Tools Actions Options Help
Shipto Activity Addinfo Billing Oeinfo Ship_notes Credit_notes Credit_letter Credit_card
Find Prev Next Add Update Delete Browse Options
Code:
Company:
Address:
City:
State:
Zip:
Country:
Email:
Web Address:
Contact:
Phone:
Cell:
FAX:
Salesperson:
Terms:
Balance:
Credit Limit:
Credit Hold:
Credit Hold Date:
(No Documents Selected)
OVR

Note

To enable/disable the text that displays beneath each icon, right click at the beginning of the toolbar and select enable text from this drop down list.



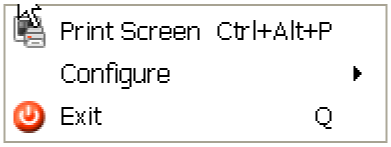
Note

To move the position of the toolbar, left click at the beginning of the toolbar and drag the toolbar to the desired position on the screen. You may do this if you prefer to have the toolbars positioned on the side or bottom of the screen rather than at the top of the screen.

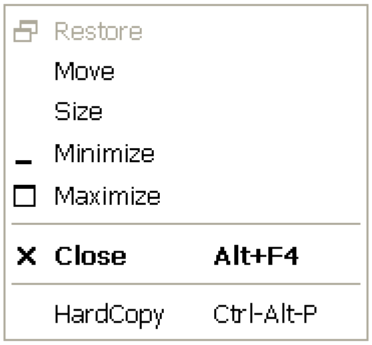
Top Menu

This is the top most menu on the screen and contains the following options, some of which are not available depending upon the type of screen program you are in and whether or not you are in update mode. If an option is unavailable for use it is grayed out.

File:



Print screen - to print the screen you must type Ctrl Alt P. You can also print the screen by clicking on the top left corner of the screen. If you do this a drop down list displays:

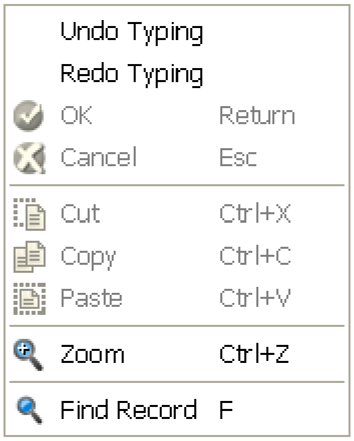


Configure - there are two options:

Fonts - displays instructions on how to change your font size.

Color - if you are in a program, this option is not functional. To change your color scheme you must be on the main system menu, click execute on the toolbar, and then click configuration. The color configuration manager is described in more detail in the Visual Development Tools Technical Guide.

Edit:



Undo Typing- displays instructions on how to undo typing which is done by right clicking on the field and then selecting Undo Typing.

Redo Typing - displays instructions on how to undo typing which is done by right clicking on the field and then selecting Redo Typing.

OK - exits update mode and saves any changes you made to the data.

Cancel - exits update mode and does not save any changes you made to the data.

Cut - used to cut text. You can also press Ctrl X.

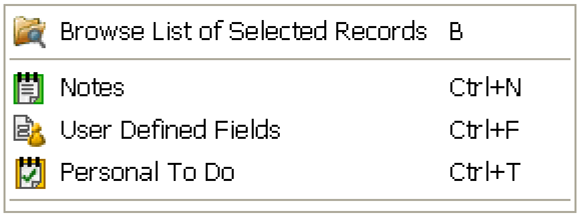
Copy - used to copy text. You can also press Ctrl C.

Paste - used to paste text. You can also press Ctrl V.

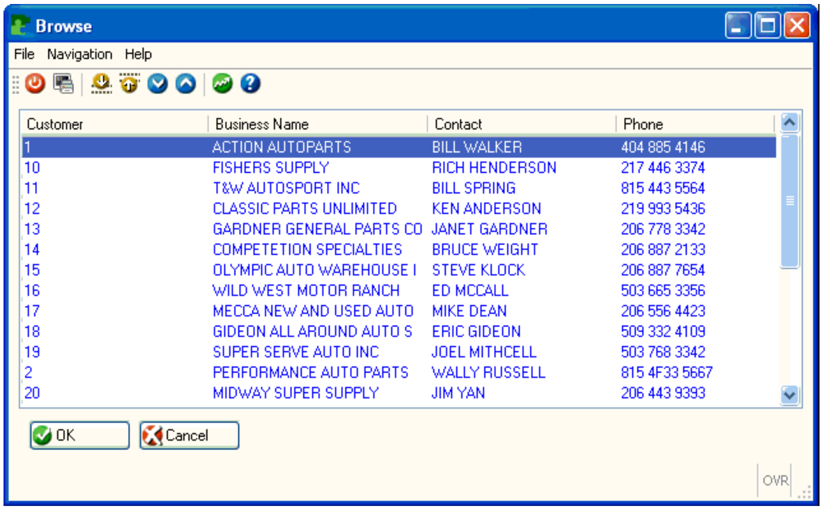
Zoom - if the cursor is in a zoom field (a zoom field is any field that has the magnifying glass icon), selecting this options will display the zoom screen. You can also press Ctrl Z

Find Record - this option will put you in find mode so that you may search for data.

View:

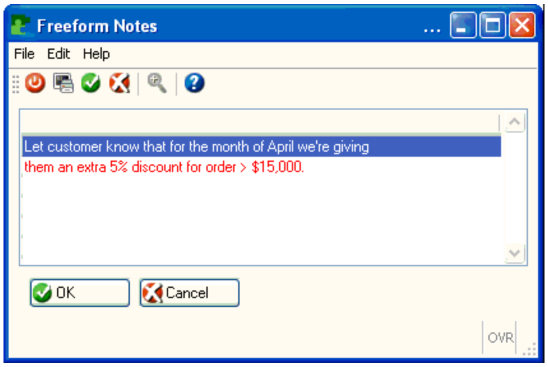


Browse - displays a summary list of selected data. For example, if you do a Find and find all customer records and then select browse, this browse screen displays:



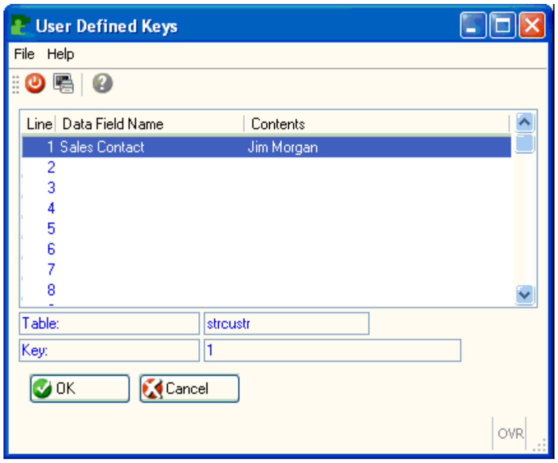
Customer	Business Name	Contact	Phone
1	ACTION AUTOPARTS	BILL WALKER	404 885 4146
10	FISHERS SUPPLY	RICH HENDERSON	217 446 3374
11	T&W AUTOSPORT INC	BILL SPRING	815 443 5564
12	CLASSIC PARTS UNLIMITED	KEN ANDERSON	219 993 5436
13	GARDNER GENERAL PARTS CO	JANET GARDNER	206 778 3342
14	COMPETETION SPECIALTIES	BRUCE WEIGHT	206 887 2133
15	OLYMPIC AUTO WAREHOUSE I	STEVE KLOCK	206 887 7654
16	WILD WEST MOTOR RANCH	ED MCCALL	503 665 3356
17	MECCA NEW AND USED AUTO	MIKE DEAN	206 556 4423
18	GIDEON ALL AROUND AUTO S	ERIC GIDEON	509 332 4109
19	SUPER SERVE AUTO INC	JOEL MITHCELL	503 768 3342
2	PERFORMANCE AUTO PARTS	WALLY RUSSELL	815 4F33 5667
20	MIDWAY SUPER SUPPLY	JIM YAN	206 443 9393

Notes - this option displays any notes that have been entered in view mode. If a record does have notes attached to it the word Notes will display in the bottom left hand corner of the screen.



Let customer know that for the month of April we're giving
them an extra 5% discount for order > $15,000.

User Defined Fields - this option displays the user defined fields that you have defined in update mode.

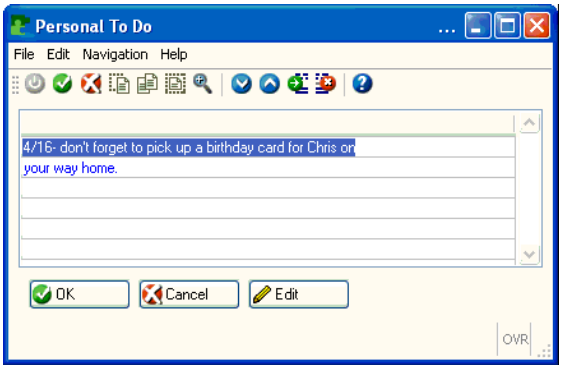


Line	Data Field Name	Contents
1	Sales Contact	Jim Morgan
2		
3		
4		
5		
6		
7		
8		

Table: strcustr

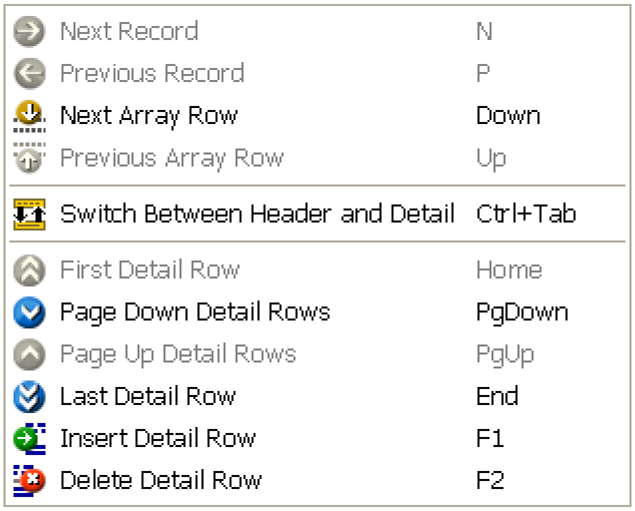
Key: 1

Personal To Do List - this optional displays your personal to do list in view mode.



4/16- don't forget to pick up a birthday card for Chris on your way home.

Navigation:



Next Record - if you have selected a group of records, clicking on this option (or pressing N) will move you to the next record.

Previous Record - if you have selected a group of records, clicking on this option (or pressing P) will move you to the previous record.

Next Array Row - moves the cursor down one row.

Previous Array Row - moves the cursor up one row.

Switch Between Header And Detail - moves cursor between header and detail sections of the screen.

First Detail Row - moves to the first detail row.

Page Down Detail Row - moves the cursor down a page of rows at a time.

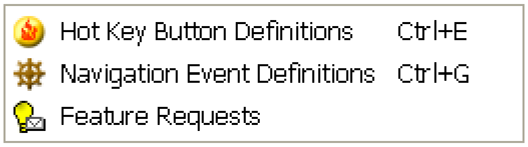
Page Up Detail Rows - moves the cursor up a page of rows at a time.

Last Detail Row - moves to the last detail row.

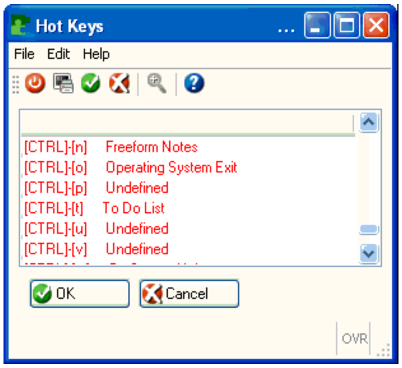
Insert Detail Row - inserts a detail row if the program allows insertion of a row (F1 also does this).

Delete Detail Row - deletes a detail row if the program allows deletion of a row (F2 also does this).

Tools:

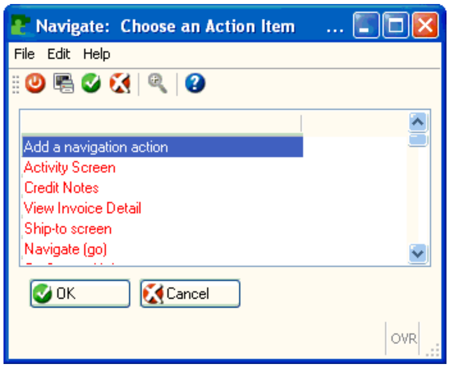

Hot Key Button Definitions Ctrl+E
Navigation Event Definitions Ctrl+G
Feature Requests

Hot Key Definitions - this option displays a list of hot key definitions. A hotkey is a key that has a navigation event attached to it. An example is of pressing Ctrl N launches the notes screen program. For more information on hotkeys and navigation events, see Chapter 2 in the Fitrix Enhancement Toolkit Technical Reference.



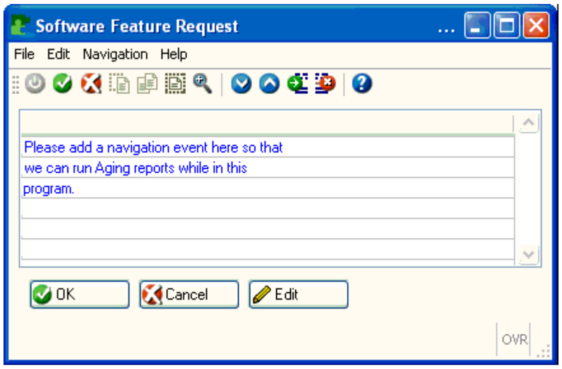
```
[CTRL]-[n]    Freeform Notes
[CTRL]-[o]    Operating System Exit
[CTRL]-[p]    Undefined
[CTRL]-[t]    To Do List
[CTRL]-[u]    Undefined
[CTRL]-[v]    Undefined
```

Navigation Event - this option displays a list of navigation events. A navigation event is a short cut that allows you to launch other programs from within a program. An example of one is the ability to launch the Update Ship-to program when in the Update Customer Information program.



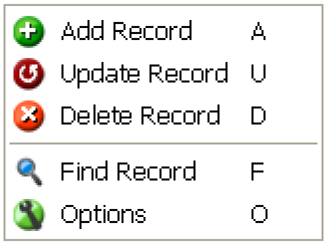
```
Add a navigation action
Activity Screen
Credit Notes
View Invoice Detail
Ship-to screen
Navigate (go)
```

Feature Request - this option launches a screen program where you can enter any program features you need. This information is then logged in the errlog file so your system administrator can review it and make the requested changes.



Please add a navigation event here so that
we can run Aging reports while in this
program.

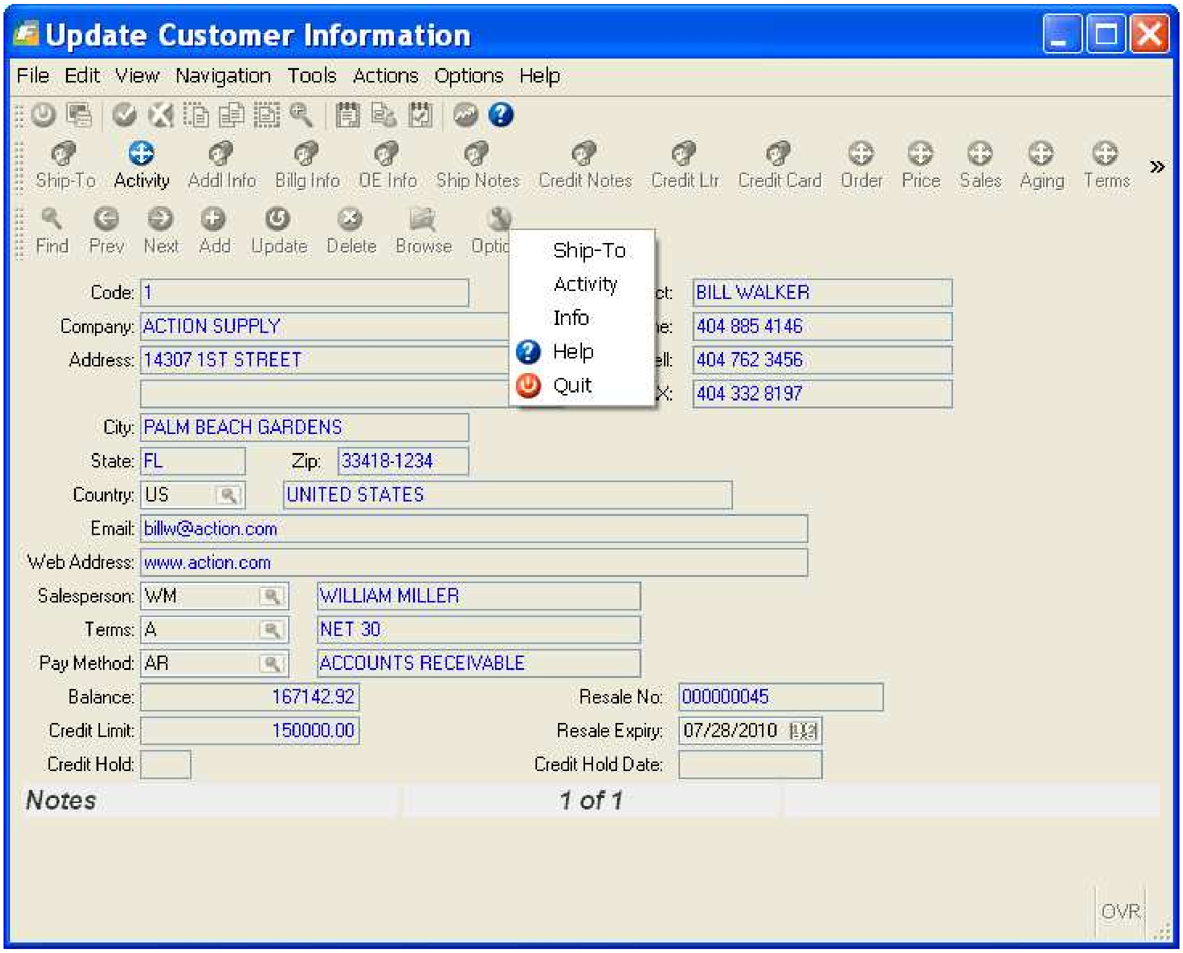
Actions:



This drop down list contains the same actions as found on the action toolbar. Please see the discussion regarding the action toolbar below.

Options:

Any programs accessible from within a program can be accessed via the Options drop down list. A good example of this is the Customer Activity screen program that can be accessed from the Update Customer Information program. These programs are also accessible from the Custom toolbar.


Update Customer Information
File Edit View Navigation Tools Actions Options Help
Ship-To Activity Addl Info Billg Info OE Info Ship Notes Credit Notes Credit Ltr Credit Card Order Price Sales Aging Terms
Find Prev Next Add Update Delete Browse Optic
Ship-To
Activity
Info
Help
Quit
Notes
OVR

Code: 1

Company: ACTION SUPPLY

Address: 14307 1ST STREET

ct: BILL WALKER

ne: 404 885 4146

ell: 404 762 3456

X: 404 332 8197

City: PALM BEACH GARDENS

State: FL Zip: 33418-1234

Country: US UNITED STATES

Email: billw@action.com

Web Address: www.action.com

Salesperson: WM WILLIAM MILLER

Terms: A NET 30

Pay Method: AR ACCOUNTS RECEIVABLE

Balance: 167142.92 Resale No: 000000045

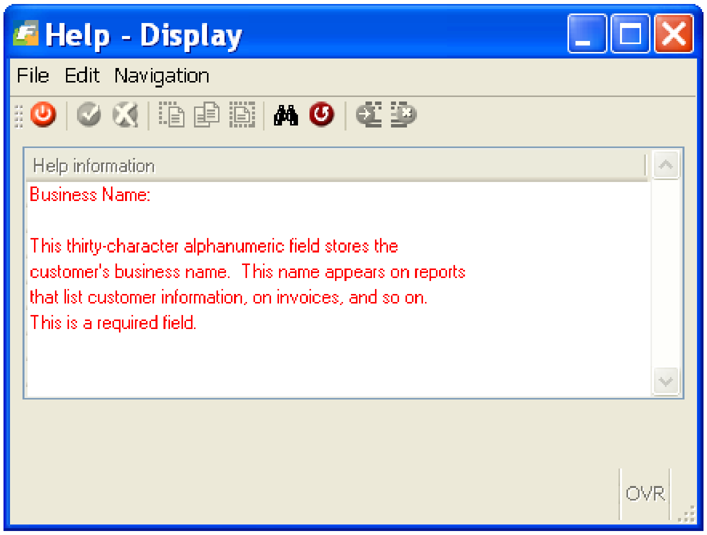
Credit Limit: 150000.00 Resale Expiry: 07/28/2010

Credit Hold: Credit Hold Date:

1 of 1

Help:

Application Help- this option displays information about the program you are using and, when in update mode, about the current field on the screen in which the cursor is located.



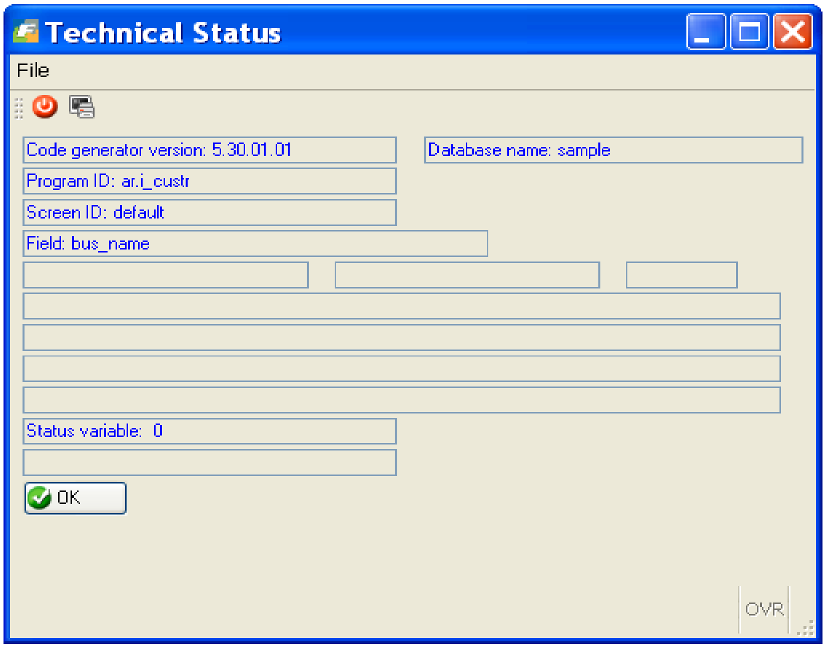
Business Name:

This thirty-character alphanumeric field stores the customer's business name. This name appears on reports that list customer information, on invoices, and so on.
This is a required field.

Note

The text can be edited to fit your specific business rules by clicking edit, and then update on the toolbar on this screen, or pressing U.

Technical Status - this option displays technical status of the program such as the program name. This is useful in the event you are having problems with a program and need to obtain the program name.



Code generator version: 5.30.01.01

Database name: sample

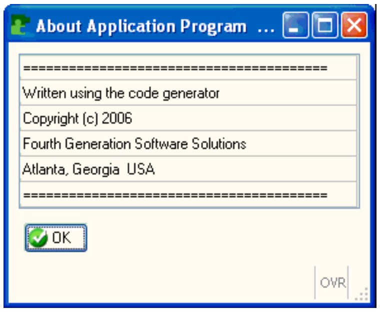
Program ID: ar.i_custr

Screen ID: default

Field: bus_name

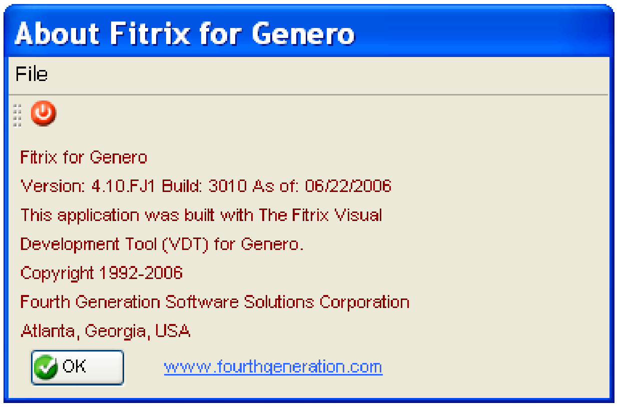
Status variable: 0

About Application Program - this option displays copyright information about the program.



```
========================================
Written using the code generator
Copyright (c) 2006
Fourth Generation Software Solutions
Atlanta, Georgia  USA
========================================
```

About Fitrix For Genero - this options display version information and also a link to the website for Fourth Generation.



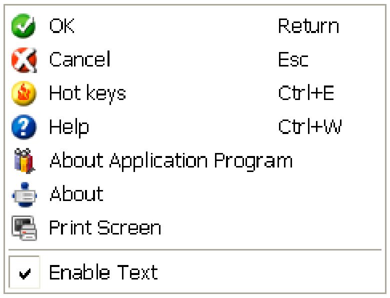
Fitrix for Genero
Version: 4.10.FJ1 Build: 3010 As of: 06/22/2006
This application was built with The Fitrix Visual Development Tool (VDT) for Genero.
Copyright 1992-2006
Fourth Generation Software Solutions Corporation
Atlanta, Georgia, USA

Main Toolbar

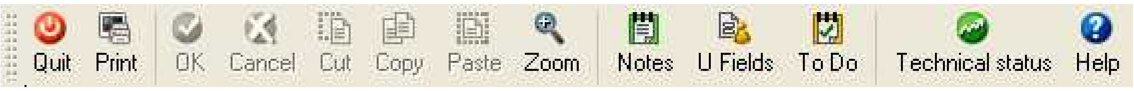
The next toolbar on the screen is called the Main ToolBar.

Note

If you want to enable text so that each icon on this toolbar has a label, right click at the beginning of the toolbar and then click Enable Text.



Just like with the Menus toolbar, if the option is not available for use it is grayed out. In the example below, the OK, Cancel, Cut, Copy, and Paste options are not available unless in update mode.



Quit - exits the program.

Print - displays information on how to print the screen (Ctrl Alt P).

OK - exits update mode and saves any changes you made to the data.

Cancel - exits update mode and doe not save any changes you made to the data.

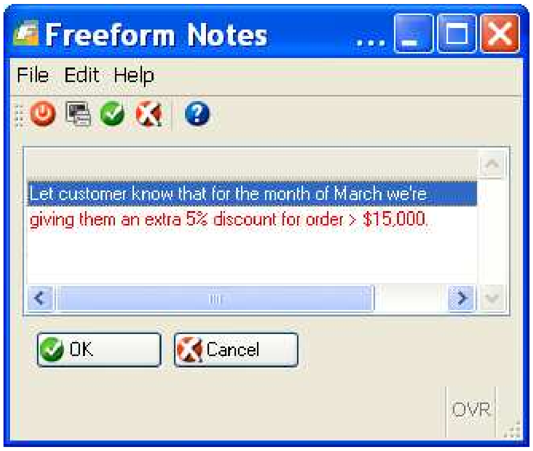
Cut - used to cut text. Same functionality as Ctrl X.

Copy - used to copy text. Same functionality as Ctrl C.

Paste - used to paste text. Same functionality as Ctrl V.

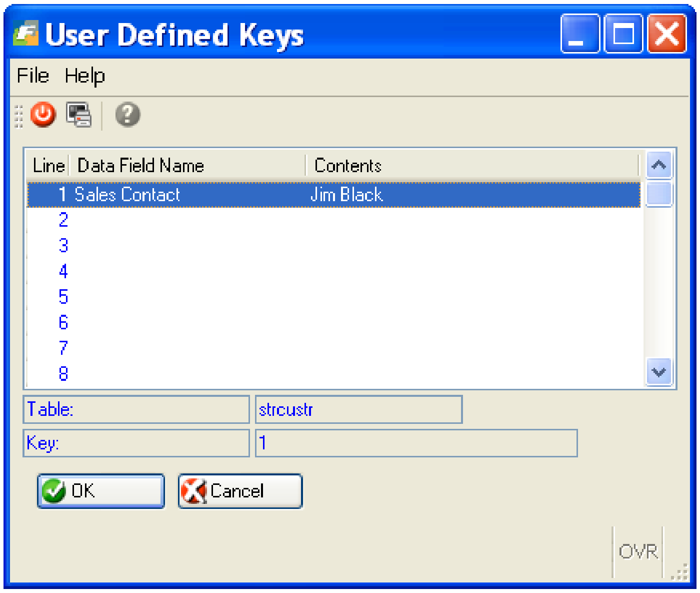
Zoom - if the cursor is in a zoom field (a zoom field is any field that has the magnifying glass icon), selecting this options will display the drill down zoom screen.

Notes - this option displays any notes that have been entered in view mode. If a record does have notes attached to it the word **Notes** will display in the bottom left hand corner of the screen.



Let customer know that for the month of March we're
giving them an extra 5% discount for order > $15,000.

User Defined Fields - this option displays the user defined fields that you have defined in update mode.



Line	Data Field Name	Contents
1	Sales Contact	Jim Black
2		
3		
4		
5		
6		
7		
8		

Table: strcustr

Key: 1

Personal To Do List - this option displays your personal to do list in view mode.

View Detail - allows you to navigate to the detail section of the screen in view only mode.

Next Page - moves the cursor down a page of rows at a time.

Previous Page - moves the cursor down a page of rows at a time.

Insert Detail Row - inserts a detail row if the program allows insertion of a row (F1 also does this).

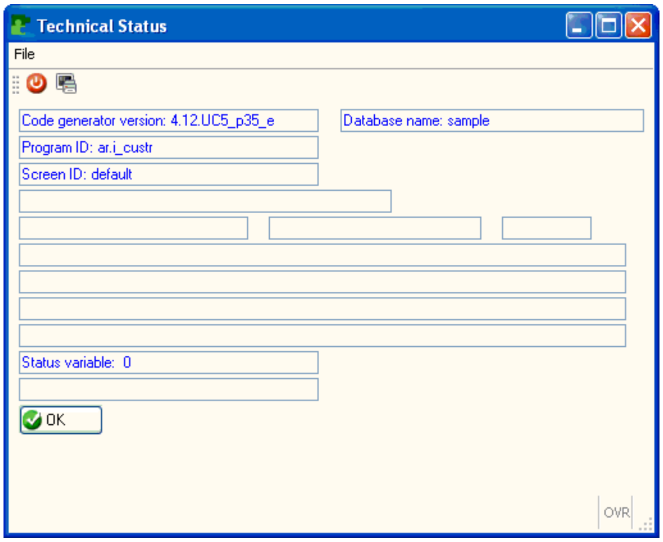
Delete Detail Row - deletes a detail row if the program allows deletion of a row (F2 also does this).

Help - this option displays information about the program you are using and, when in update mode, about the field on the screen you are in.

Note

The text can be edited to fit your specific business rules by clicking edit, and then update on the toolbar on this screen, or pressing U.

Technical Status - this option displays technical status of the program such as the program name. This is useful if you are troubleshooting a program and need the program name.



Code generator version: 4.12.UC5_p35_e

Database name: sample

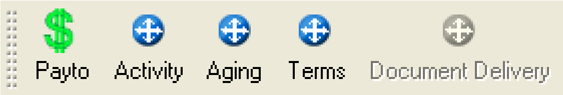
Program ID: ar.i_custr

Screen ID: default

Status variable: 0

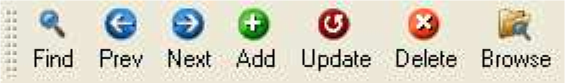
Custom Toolbar

The Custom toolbar is an optional third toolbar on the screen. It will only be found in programs where custom actions have been added to the main screen to launch navigation events. Setting up this toolbar is covered later on in this class. In the example below there are four navigation events set up in the Update Vendor Information screen program so that you can access Vendor Pay-to information, Vendor Invoice and Payment Activity, run an AP Aging, enter new Payment Terms, and when in update mode set the method of delivery for Vendor Purchase Orders.



Ring Menu Toolbar

The last toolbar on the menu is the Ring Menu toolbar, and is used to search for rows in the associated table, and navigate or work with rows. The options are selected by either clicking on them with the mouse or pressing the first letter of the option name (Example: F for Find).



Find - In order to view, modify, or delete a record, you must first retrieve it. Use the Find command to retrieve one or more records.

After selecting Find from the menu, a Query-by-Example (QBE) screen displays, and your cursor moves into the first field in the screen. A QBE screen lets you query (search) the records in the database by entering data in a template that resembles the data-entry screen. Once you have filled in the QBE screen with the data you want to use for your search as specified below, click OK, or press Enter, to execute the search. The system will retrieve all of the records or transactions that match the data you entered.

There are three different ways to find:

All of the records - simply click OK, or press Enter, without entering any data into the QBE screen. The system will retrieve all of the records for that program.

A particular record, enter a piece of information which is unique to the record you are looking for. For example, if you are searching for a particular invoice, you might enter the customer code, or the customer's name and the invoice number. The system will retrieve the record which contains this unique information.

A group of records, enter search criteria into the QBE screen. Search criteria is any information which you want the records to match. For example, if you want to find all customers whose names begin with the letter A, enter "A*" in the business name field. You can enter search criteria in more than one field to further limit the search if you desire. For instance, if you want to find the employee records for all hourly employees who live in Texas, enter the code for hourly employee in the Employee Type field and Texas in the State field. This will retrieve all records which match both search criteria.

Using a combination of search criteria is a very powerful way to manage large amounts of information because it allows you to retrieve only the records you want to see. If you are executing a large query that you find is taking several minutes, you can press [ESC] to halt the search in progress. The system will display the records that were retrieved up to the point at which you halted the search.

If your search finds one record, that record is displayed on the screen. If your search results in more than one record, the first record is displayed. Notice that a message appears at the bottom of the screen like 1 of n. The first number indicates the position in the stack of the records you are viewing; n is the total number of records retrieved by the search. Use the Next, Previous, and Browse commands (discussed below) to display the records on the screen.

Next and Prev (Previous) - Once you have retrieved a group of records using the Find command, use Next and Prev to page through them. Next displays the next record; Prev displays the previous record. When you reach the last record (ie- record 10 of 10), Next will take you to the first record. When you are on the first record, Prev will display the last record. The record added most recently is always the last record so to quickly find the last record added, simply use Find to retrieve all records and then select Prev. You can also press N and P.

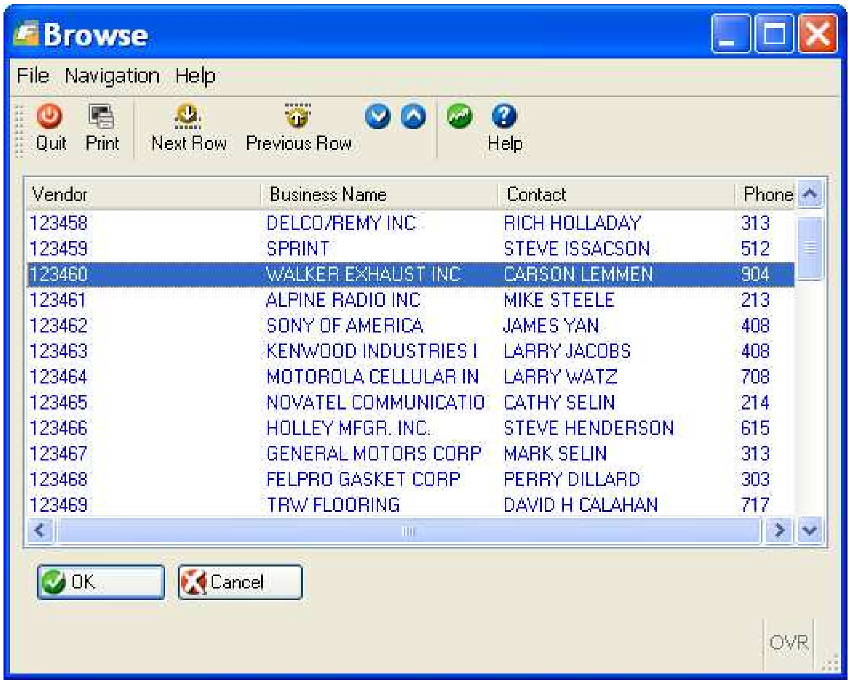
Add - Use Add to add a new record or transaction in the program you are running. A blank data-entry screen appears with the cursor in the first data-entry field. Move through the fields on the screen by pressing [TAB]. For more information on entering data into data-entry screens, see "Features Common to All Screens" Chapter 4.

Once you finish entering information, save the new record or transaction by clicking OK, or pressing Enter. If you decide you do not want to add the record, abandon your changes by pressing [ESC] or clicking cancel.

Update - Update is used to modify data in existing records. Once you have located the document you wish to modify with Find and Next/Prev or Browse, use Update to modify the record. The cursor moves to the first input field on the screen. Move through the fields on the screen by pressing [TAB] making changes or additions to the data as required. Notice how the comment line at the bottom of the screen changes to correspond with the field where the cursor rests. Three keystroke combinations-[CTRL]-[a], [CTRL]-[x], and [CTRL]-[d]- are available for you to insert and delete data while you update the record. Click OK, or Press Enter, to store your changes. Click Cancel or press [ESC] to abandon your changes and restore the document.

Delete - Use Delete to remove records from the system. You must first use Find to retrieve a record in order to delete it. After selecting Delete, you are prompted to verify deletion of the record or transaction to avoid accidentally deleting data.

Browse - Browse lists information about several records at once. The Browse list displays a single line of information for each found record. You can scroll or page through the list and select the desired record to be displayed on the screen. Browse is more efficient than Next and Previous when you have retrieved a large number of records. Each Browse screen is slightly different, to accommodate the information it displays, but they all operate in the same manner.



Vendor	Business Name	Contact	Phone
123458	DELCO/REMY INC	RICH HOLLADAY	313
123459	SPRINT	STEVE ISSACSON	512
123460	WALKER EXHAUST INC	CARSON LEMMEN	904
123461	ALPINE RADIO INC	MIKE STEELE	213
123462	SONY OF AMERICA	JAMES YAN	408
123463	KENWOOD INDUSTRIES I	LARRY JACOBS	408
123464	MOTOROLA CELLULAR IN	LARRY WATZ	708
123465	NOVATEL COMMUNICATIO	CATHY SELIN	214
123466	HOLLEY MFGR. INC.	STEVE HENDERSON	615
123467	GENERAL MOTORS CORP	MARK SELIN	313
123468	FELPRO GASKET CORP	PERRY DILLARD	303
123469	TRW FLOORING	DAVID H CALAHAN	717

The commands on the Browse menu are:

Next and Prev - move the highlight to the next or previous record in the array. The array is not continuous; that is, you cannot move backwards past the first record or forward beyond the last record.

Up and Down - page backward and forward through the records one browse page at a time using the up and down arrow icons.

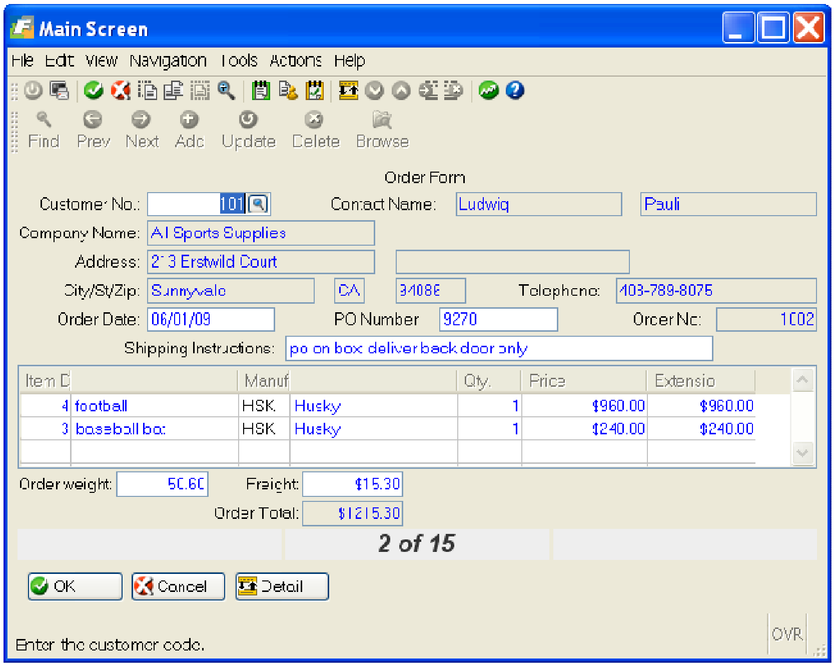
Options - with the Options icon, you access extra options which may be available in certain programs. When you select the Options command, a drop down list displays additional options available for that particular program. Setting up additional options is covered later in this class.

Accessing Zooms

Zooms help the user enter data. When entering values in fields, the user can sometimes Zoom into a list of valid values for that field and select one. Users invoke Zooms by pressing [CTRL]-[z] in a field or clicking the magnifying glass icon. Not all fields have Zooms attached to them.

Zoom help users enter valid values.

In this example, the user initiates a Zoom from the Customer No. field.



Order Form

Customer No.: 101 | Contact Name: Ludwig | Pauli

Company Name: A1 Sports Supplies

Address: 213 Erstwild Court

City/St/Zip: Sunnyvale | CA | 94086 | Telephone: 403-789-8075

Order Date: 06/01/09 | PO Number: 9270 | Order No: 1002

Shipping Instructions: po on box deliver back door only

Item D		Manuf		Qty.	Price	Extensio
4	football	HSK	Husky	1	$960.00	$960.00
3	baseball bat	HSK	Husky	1	$240.00	$240.00

Order weight: 50.60 | Freight: $15.30

Order Total: $1215.30

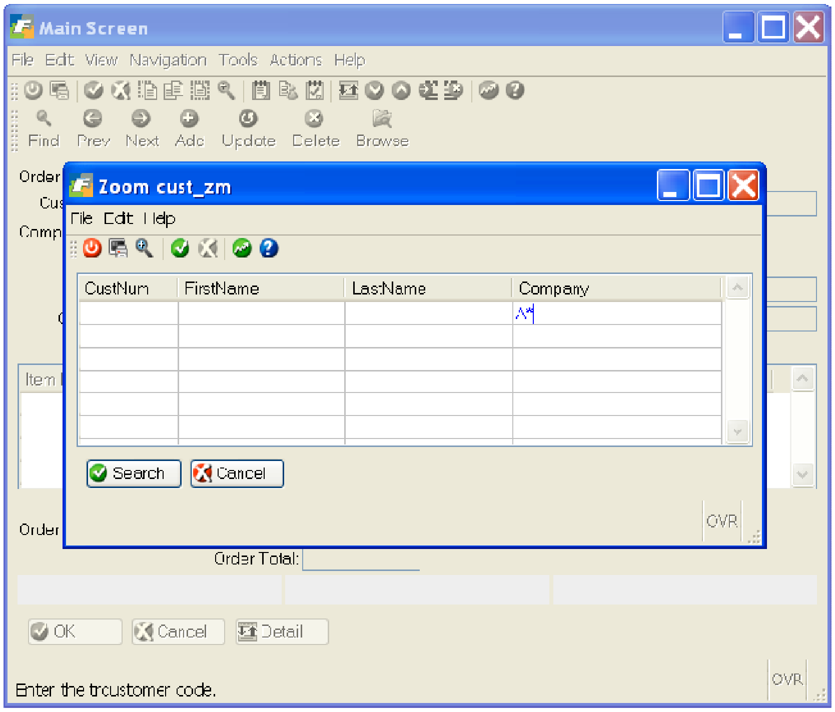
2 of 15

Enter the customer code.

Zooms also use filters before returning values. If there are many valid values that can go into a field, Zooms, by default, may first display a selection criteria screen. The selection criteria screen allows users to limit which values the Zoom returns.

Zooms can filter values before they are returned.

In this example, the user wants to see a list of companies that begin with the letter A.



CustNum	FirstName	LastName	Company
			A*

Order Total:

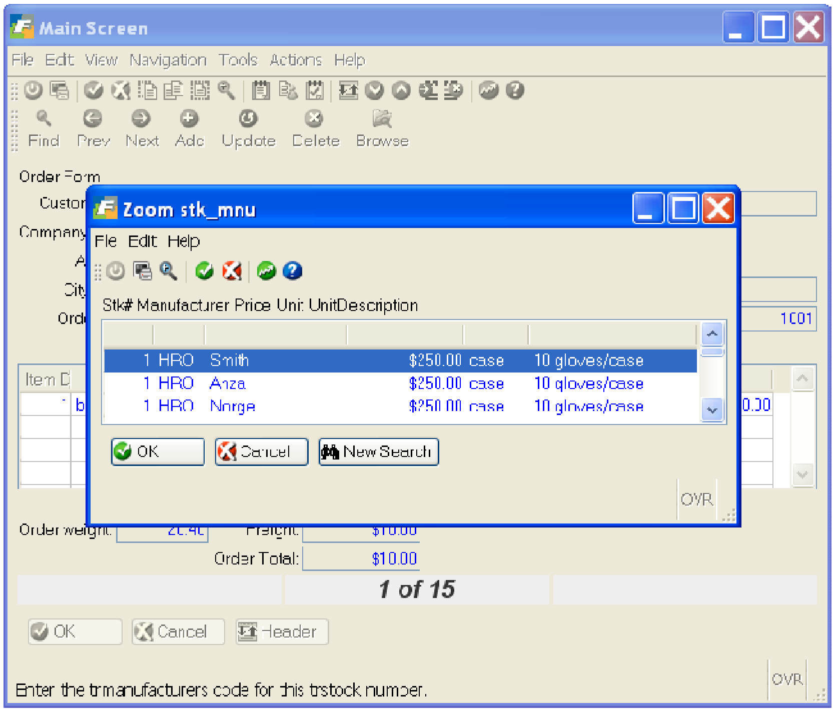
Enter the trcustomer code.

Using AutoZooms

You can also invoke a Zoom without pressing [CTRL]-[z] or clicking the magnifying glass icon. If you place an asterisk in a field and press [TAB], the Zoom is performed for you. You can combine the asterisk with letters to filter the Zoom.

AutoZooms let you enter selection criteria directly into a field.

In this example, the AutoZoom returns values that begin with H.



Stk# Manufacturer Price Unit UnitDescription

Stk#	Manufacturer		Price	Unit	UnitDescription
1	HRO	Smith	$250.00	case	10 gloves/case
1	HRO	Anza	$250.00	case	10 gloves/case
1	HRO	Norge	$250.00	case	10 gloves/case

Order weight: 20.40 Freight: $10.00

Order Total: $10.00

1 of 15

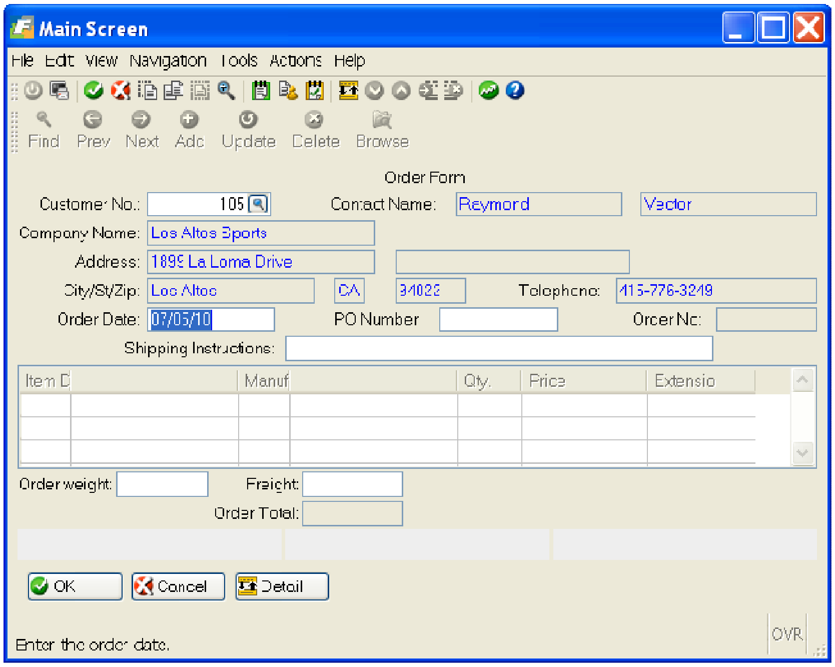
Enter the trmanufacturers code for this trstock number.

Using Lookups

When a user enters a value in a field, Lookups can be defined to display related data into adjacent fields. Lookups also ensure that the user only enters valid values into a field.

Lookups display related data into adjacent fields.

In this example, the Lookup displayed values for the Contact Name, Company Name, Address, City/St/Zip, and Telephone fields. This Lookup is based on the Customer No. field.


Main Screen
File Edit View Navigation Tools Actions Help
Find Prev Next Add Update Delete Browse
OK
Cancel
Detail
OVR

Order Form

Customer No.:	105	Contact Name:	Raymond	Victor	
Company Name:	Los Altos Sports				
Address:	1899 La Loma Drive				
City/St/Zip:	Los Altos	CA	94022	Telephone:	415-776-3249
Order Date:	07/05/10	PO Number		Order No:	
Shipping Instructions:					

Item D	Manuf	Qty.	Price	Extensio

Order weight:

Freight:

Order Total:

Enter the order date.

Section Summary

- In order to create and run *Input* programs, you must set certain LINUX environment variables and export them. These variables are set for you via the startup script.
- Using *Fitrix VDT Application Code Generator*, you can create sophisticated input programs.
- The topmost portion of an input program is known as the Toolbar Area. The Toolbar area can contain as many as 4 Toolbars – Top Menu, Main, Custom (optional or custom) and Ring Menu Toolbars.
- Input programs may use zoom screens to assist in data entry. Zooms perform data selection and validation tasks.
- You can access a Zoom by pressing [CTRL]-[z] or clicking the magnifying glass icon.
- AutoZooms let you place selection criteria directly into an input field.
- Lookups diplay related data into adjacent fields. For example, when a user enters a number into the Customer No. field, the Contact Name, Company Name, Address, City/St./Zip, and Telephone fields get filled automatically.

Exercise 1A

Objective: To set up your development environment.

Genero Desktop Client

For all the exercises in this book, you should be using the Genero Desktop Client (GDC). You do this by logging into Fitrix by clicking the Fitrix Icon on your desktop:

Once you are logged in, you are presented with the application menus. To get to the Linux prompt, click the terminal screen icon 🖥 on Toolbar at the top of the window.

Check Your Current Environment Variable Settings

The `env` command displays current environment variable settings.

- **At the LINUX prompt, enter:**

```
env
```

Use the env command to see what values the following environment variables contain:

```
fg
INFORMIXDIR
PATH
DBPATH
FGLDBPATH
```

To show the value in a single environment variable, you can use the `echo` command.

- **At the LINUX prompt, type:**

```
echo $fg
```

Set Your Environment

Each of the environment variables shown on the previous page must point to a specific directory, depending upon how your system is set up. Here is a rundown of the correct variable settings:

`fg`

This variable should point to the directory where the Fitrix application is installed. For example: /fitrix/fx_dev.

`fgtooldir`

This variable should point to the directory where the VDT Application Code Generator product is installed. For example, `$fg=/fitrix/fx_tools.`

`INFORMIXDIR`

This variable should point to the directory where your Informix product is installed. For example, `$INFORMIXDIR=/usr/informix.`

`PATH`

This variable should contain both the `$fg/bin` and `$INFORMIXDIR/bin` directories.

`DBPATH`

This variable should contain `$fgtooldir/lib/forms` and `$fgtooldir/codegen`/data.

`FGLDBPATH`

This variable should point to where the generro schema definitions are kept. For example, /ftirix/fx_dev/data.

`DBNAME`

The DBNAME variable is used by the fg.screen and fg.form development scripts. The DBNAME variable, while not required if you are using the ‘standard’ database. it is very useful if you are not. The variable should contain the name of the database you are using for development.

Note

The dollar sign ($) before the environment variable indicates that you want to display the value contained within the variable.

You must issue two commands to set an environment variable. First enter the variable name followed by an equals sign and the value the variable should contain. Second, "export" the variable. Please substitute the name of your database that your instruction gives you below for ‘student1’ . To set the $DBNAME variable.

At the Linux prompt, type:

```
DBNAME=student1      ; export DBNAME
```

Use the echo command again to check the variable:

At the prompt, type:

```
echo $DBNAME
```

You need to set the DBNAME variable to the name of your student database for all of the exercises in this workbook.

Exercise 1 B

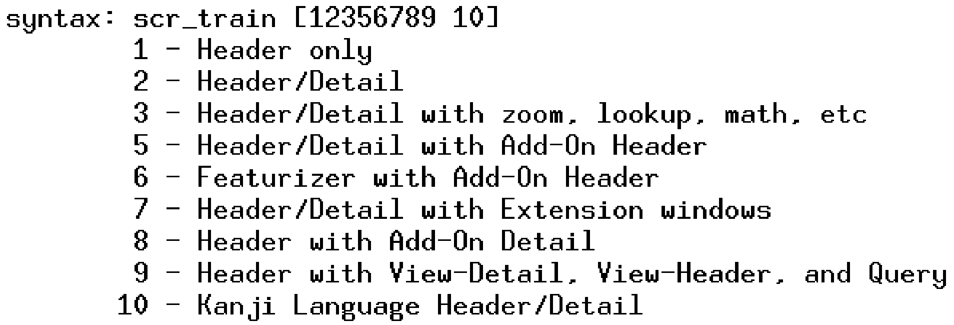
Objective: To become familiar with the VDT Application Code Generator demo programs.

List the VDT Application Code Generator Training Programs

At the LINUX prompt, type:

```
scr_train
```

The following list appears:



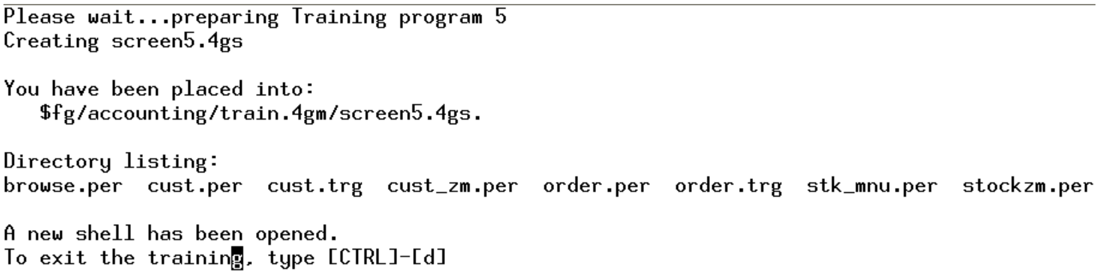
```
syntax: scr_train [12356789 10]
         1 - Header only
         2 - Header/Detail
         3 - Header/Detail with zoom, lookup, math, etc
         5 - Header/Detail with Add-On Header
         6 - Featurizer with Add-On Header
         7 - Header/Detail with Extension windows
         8 - Header with Add-On Detail
         9 - Header with View-Detail, View-Header, and Query
        10 - Kanji Language Header/Detail
```

Start scr_train 5

At the LINUX prompt, type:

```
scr_train 5
```

When you start `scr_train 5`, the following message appears:



```
Please wait...preparing Training program 5
Creating screen5.4gs

You have been placed into:
   $fg/accounting/train.4gm/screen5.4gs.

Directory listing:
browse.per  cust.per  cust.trg  cust_zm.per  order.per  order.trg  stk_mnu.per  stockzm.per

A new shell has been opened.
To exit the training, type [CTRL]-[d]
```

In addition, your prompt changes to reflect the demo program:

```
Training 5 ->
```

The VDT training program (scr_train) gives you a fresh set of form specification (*.per), trigger (*.trg) extension (*.ext), and feature set (*.set) files each time you run it. These files supply the *VDT Application Code Generator* with instructions for building an input program.

Note

Some *VDT* Training programs contain all of these files while others only contain form specification (*.per) files. At this point, you do not have to know or understand what these files do. Just realize that they are used by *VDT Application Code Generator* to create an input program.

Also realize that each time you run a training program, you receive a fresh set of files. Because of this fact, do not be afraid to "break" the program. If a file is corrupted, just start over.

Once you receive the Training prompt, you can use the VDT Application Code Generator to build and run an input program. In general, the following steps are required:

1. **Run the *VDT Application Code Generator* to create source code.**

2. **Execute the `fg.make` command, the compilation program to compile the source code and build a runnable program file.**

3. **Execute `fglrun`, to run the resulting program file.**

Exercise 1C

Objective: To convert the initial `scr_train 5` files into a program.

Start the VDT Application Code Generator

- **At the Training prompt, enter:**

```
fg.screen
```

This command starts the *VDT Application Code Generator,* which reads the form specification (*.per) files in the training directory and creates 4GL source code based on these files. As the *VDT Application Code Generator* works, multiple lines of code scroll past your screen.

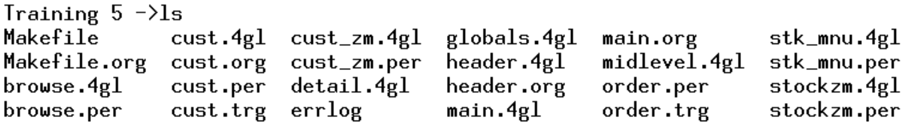
List the Generated Files

When the *VDT Application Code Generator* finishes creating code, the Training prompt reappears. You can use the `1s` command to see a listing of the files the *VDT Application Code Generator* creates.

- **At the Screen Demo prompt, type:**

```
1s
```

The following list of files appears:



```
Training 5 ->ls
Makefile      cust.4gl  cust_zm.4gl  globals.4gl  main.org      stk_mnu.4gl
Makefile.org  cust.org  cust_zm.per  header.4gl   midlevel.4gl  stk_mnu.per
browse.4gl    cust.per  detail.4gl   header.org   order.per     stockzm.4gl
browse.per    cust.trg  errlog       main.4gl     order.trg     stockzm.per
```

As you can see the *Screen* Code Generator creates several source code (*.4g1) files. From these files, the compilation utility (`fg.make`) builds a runnable program file.

After you use the *VDT Application Code Generator* to create 4GL source code, you can use the `fg.make` command to compile the source code and build a runnable program file.

Start the Compilation Utility

- **At the Training prompt, enter:**

```
fg.make
```

This command performs several tasks, most of which are described in later chapters. For now, you should simply realize that it builds a runnable program file.

List Your Program File

When the `fg.make` command finishes, the Training prompt reappears. Again, you can use the `ls` command to display the files created by `fg.make`. You should see screen5.42r program file.

- **At the Screen Demo prompt, type:**

```
ls
```

In the file listing, you should have a `screen5.42r` file.

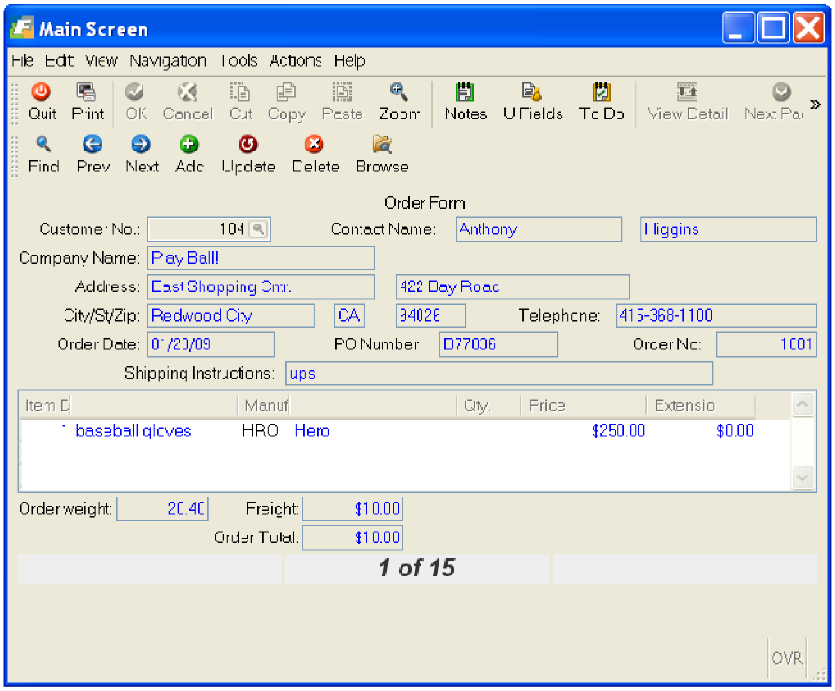
Start the Input Program

Start the program:

```
fglrun screen5.42r
```

The screen 5 input program begins:

This figure shows the main screen of scr_train 5.



Order Form

Customer No.: 104 Contact Name: Anthony Higgins

Company Name: Pay Ball!

Address: East Shopping Cntr. 422 Bay Road

City/St/Zip: Redwood City CA 94026 Telephone: 415-368-1100

Order Date: 07/23/09 PO Number D77036 Order No: 1001

Shipping Instructions: ups

Item D		Manuf		Qty.	Price	Extensio
1	baseball gloves	HRO	Hero		$250.00	$0.00

Order weight: 20.40 Freight: $10.00

Order Total: $10.00

1 of 15

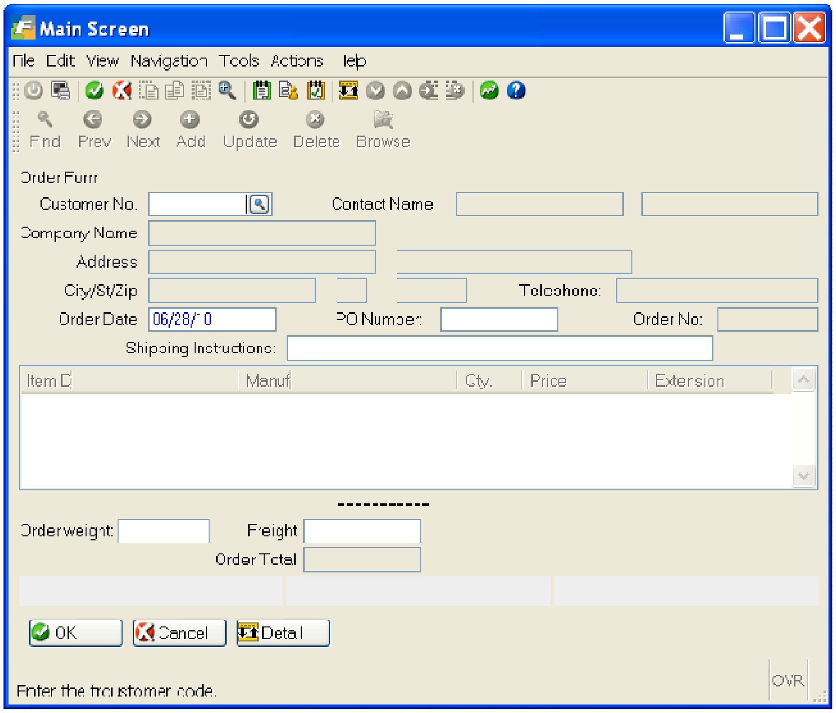
Exercise 1 D

Objective: To become familiar with the input program functionality.

Add a Record

1. **Click the Add Button on the Ring Menu Toolbar.** Add

A screen to enter a new record is displayed and focus moves to the first field:



Order Date 06/28/'0

Enter the trcustomer code.

On some fields, the magnifying glass appears. The magnifying glass indicates that a reference table exists for the field. You can press [CTRL]-[z] to open the reference table or click the magnifying glass. Then select a value for the field.

Fill in the input fields in the header portion of the window, pressing [TAB] to move from field to field.

To move to the detail portion of the window, click the Detail **button or press [CTRL-TAB]. Use the [TAB] key to move from field to field in the detail portion of the window.**

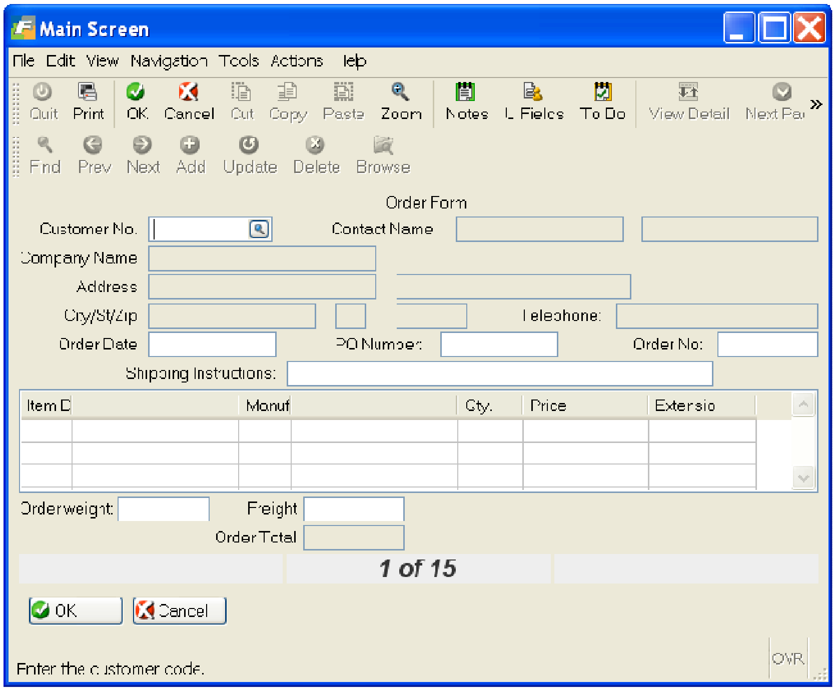
Press [Enter] to store the record.

Find a Record

The Find command lets you select a single record, a group of related records, or all the available records.

2. **Click the Find button** Find **on the Ring Menu Toolbar or press F**.

A blank record appears and the focus moves to the first field:



1 of 15

Enter the customer code.

3. **Press [Enter].**

   All the records in your database table get returned. The first record appears in your main window, and a count of the total records selected displays at the bottom of the form on the status line. You can use Next and Prev buttons to scroll through the entire list.

To limit a Find to a single record or a group of related records, you can enter selection criteria in the fields. This ability is known as Query-By-Example (QBE). For instance, to select all the records that have order dates greater than 04/1/09:

4. **Click the Find button.** Find

5. **Click in the Order Date field and enter:**

```
> 04/01/86
```

6. **Press [Enter].**

All the records older than 04/01/09 are returned. Again you can use Next and Prey buttons to scroll through the list of records.

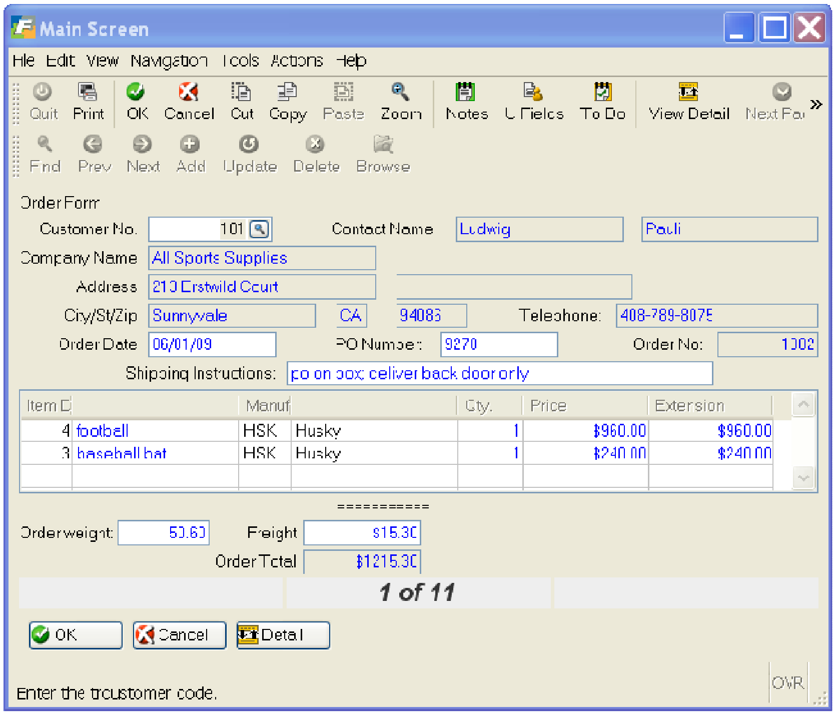
Update a Record

The Update command lets you alter the values in a record.

1. **Use Find to select the record you want to update.**

2. **Select Update by clicking** Update **or by pressing the 'u' key.**

Your focus moves to the first input field, Customer No.



Order Form

Customer No. 101
Contact Name Ludwig Pauli
Company Name All Sports Supplies
Address 213 Crstwild Court
City/St/Zip Sunnyvale CA 94083
Telephone: 408-789-8075
Order Date 06/01/09
PO Number: 9270
Order No: 1002
Shipping Instructions: po on box; deliver back door only

Item ID		Manuf		Qty.	Price	Extension
4	football	HSK	Husky	1	$960.00	$960.00
3	baseball bat	HSK	Husky	1	$240.00	$240.00

===========

Orderweight: 50.60
Freight $15.30
Order Total $1215.30

1 of 11

Enter the customer code.

3. **Move to the field that you want to change and change its value.**

4. **Press [Enter] to store your change.**

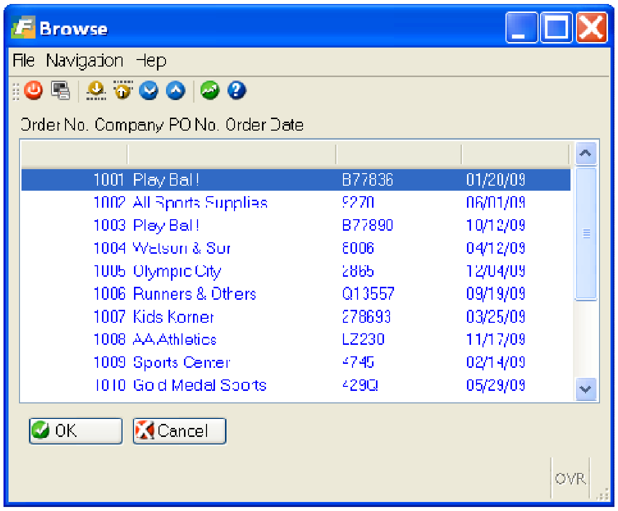
Browse a List of Records

The Browse function lets you view a list of selected records in line-by-line format.

1. **Use the Find button to select a group of records.**

2. **Click the Browse button Browse from the Action Toolbar or press the 'b' key.**

A secondary window appears showing the selected records in a line-by-line format:



Order No.	Company	PO No.	Order Date
1001	Play Ball!	B77836	01/20/09
1002	All Sports Supplies	5270	06/01/09
1003	Play Ball!	B77890	10/12/09
1004	Watson & Son	8006	04/12/09
1005	Olympic City	2865	12/04/09
1006	Runners & Others	Q13557	09/19/09
1007	Kids Korner	278693	03/25/09
1008	AA Athletics	LZ230	11/17/09
1009	Sports Center	4745	02/14/09
1010	Gold Medal Sports	4290	05/29/09

3. **Use buttons on the browse window toolbar to select a record. Select a record by double clicking it or pressing [Ener].**

Quit the Input Program and Training Program.

When you are finished exploring the input program, Click the Quit button Quit on the Main Toolbar. The Quit returns you to the Training prompt.

1. **At the Training prompt, type [CTRL]-[d] or enter:**

   ```
   exit
   ```

2. **Type [CTRL]-[d] or enter:**

   ```
   exit
   ```

You are returned to Fitrix visual menus.

Chapter 2

Getting Started with the VDT Form Designer

Main topics:

- VDT Form Designer Overview
- Starting the VDT Form Designer
- Using the VDT Form Designer Pull-Down Menus
- Creating a Form Image
- Converting Forms into Input Programs

VDT Form Designer Overview

The VDT Form Designer lets you develop complete data-entry programs written in FourJ's Genero Business Development Language. FourJ's Genero Business Development Language is a fourth generation language and referred to as 4GL in the rest of this workbook.

The VDT Form Designer is an interactive utility featuring a full screen editor, a database administration facility, and a screen enhancement builder. The VDT Form Designer acts as the control center for running the VDT Application Code Generator and compilation utility. From within the VDT Form Designer you can:

- Paint a form image, which can be directly converted into an input program.
- Access the database to add, delete, and update tables and columns.
- Store form image information in ASCII files (form specification *.per files), which are compliant with FourJ's Perform format and easily moved to other systems.
- Create custom program events that are called from logical trigger points within the generated code.
- Copy and move any element of the form image.
- Store form image blocks on a Clipboard.
- Define data-entry areas and how they join with other data entry areas.
- Define how forms work with other forms.
- Specify the order in which input fields are processed on the form.
- Generate default form images with the AutoForm feature.
- Access other programs and tools on the system without leaving the VDT Form Designer.

Starting the VDT Form Designer

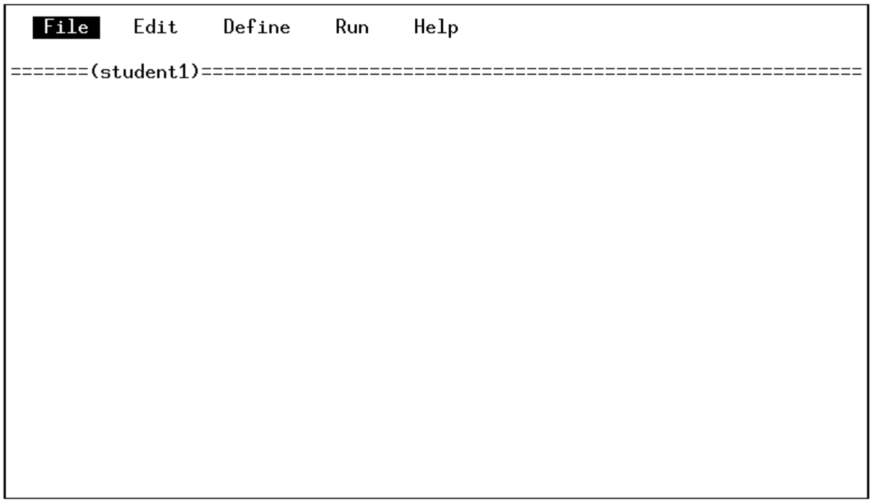
You can start the VDT Form Designer using the fg.form command. This command has the following syntax:

```
fg.form -dbname database
```

Where *database* is the name of the database you want to use.

After you type this command, the VDT Form Designer loads and displays the following window to your screen:

The VDT Form Designer consists of two sections: the pull-down menus and the Form Editor.


File
Edit
Define
Run
Help
=======(student1)==

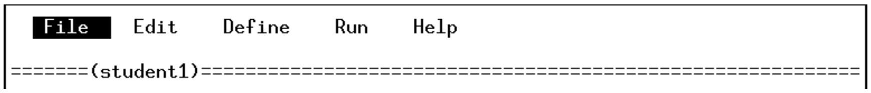
You should always navigate to the directory in which you want to read and write form specification (*.per) files, before starting the VDT Form Designer.

The VDT Form Designer consists of two sections: the pull-down menus and the Form Editor.

Using the VDT Form Designer Pull-Down Menus

The VDT Form Designer contains five pull-down menus.

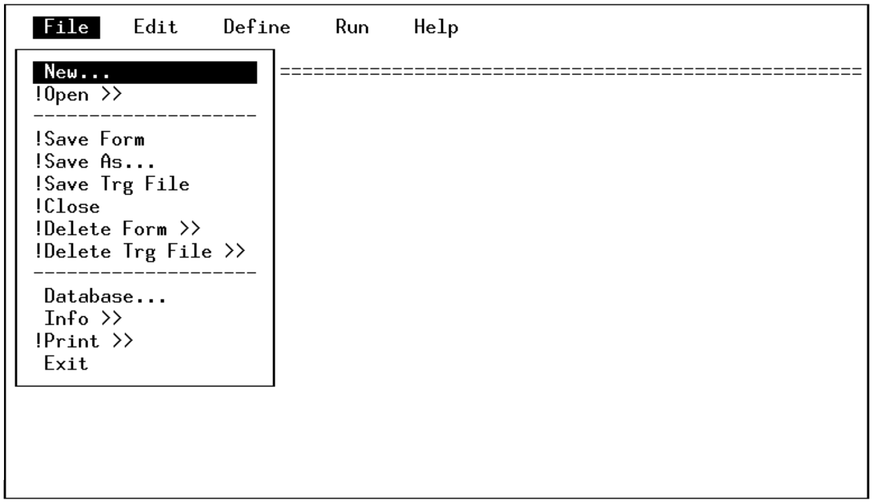
The VDT Form Designer contains five pull-down



```
 File   Edit   Define   Run   Help

=======(student1)==========================================================
```

You can open a pull-down menu by highlighting it and pressing [ENTER). You can also open a pull-down menu by typing the first character of the menu name (e.g., type F to open the File pull-down menu). Each pull-down menu contains a number of menu options. You select a menu option by highlighting it and pressing [ENTER]

You can open a pull-down menu by highlighting it and pressing [ENTER]. You can also open a pull-down menu by typing the first character of the menu name (e.g., type F to open the File pull-down menu).



```
 File   Edit   Define   Run   Help
 New...              =====================================================
!Open >>
--------------------
!Save Form
!Save As...
!Save Trg File
!Close
!Delete Form >>
!Delete Trg File >>
--------------------
 Database...
 Info >>
!Print >>
 Exit
```

Options preceded by an exclamation point (!) are not available. Options followed by greater-than signs (») open another menu with additional options. Options followed by an ellipsis (...) open a subsequent window.

Creating a Form Image

The VDT Form Designer lets you paint form images. You can use the VDT Form Designer to create a new form image or you can open existing form images. A form image graphically represents how your form will look and work once it is built. You paint and edit form images from within the Form Editor. The general steps for creating a new form image are as follows:

1. **Select New from the File pull-down menu.**

2. **Enter a name for the new form.**

3. **Select the screen type you want to use.**

In all there are ten screen types you can build. Your main screen is either a header or header/detail screen. The other screens act as secondary screens, some of which you can connect to the main screen (see “Using Different Screen Types” on page 182).

Screen Type	**Function**
`header`	Writes to a single database table.
`header/detail`	Writes to a header table and a detail table.
`add-on header`	Writes to a peripheral table from the main screen.
`add-on detail`	Writes to an additional scrolling detail table from the main screen.
`extension`	Writes to additional columns within the main header table.
`zoom`	Selects valid values for an input field.
`browse`	Lists documents in a line-by-line format.
`query`	Generates a selection prompt for use with report programs.
`view-header`	Allows you to view data from a peripheral header table.
`view-detail`	Allows you to view data from a subsequent scrolling detail table.

Painting the Form Image

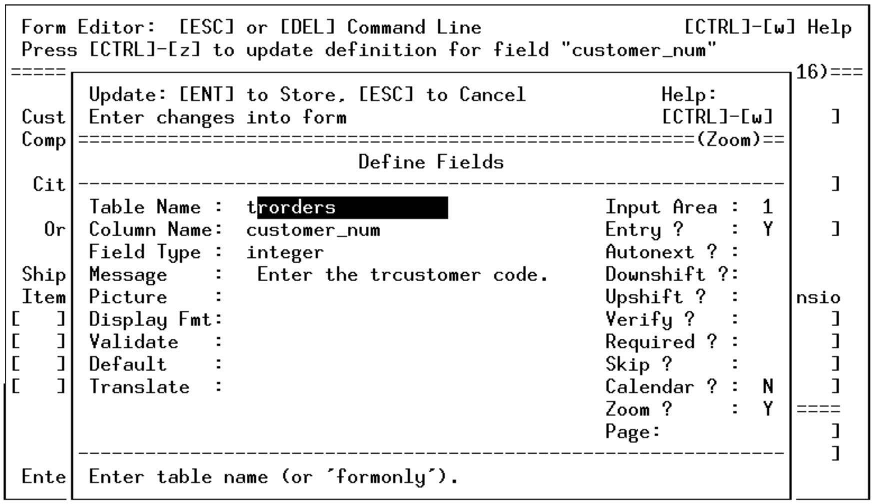
Once you load a form into the Form Editor, you can start painting the form image. Form images contain both text and input field definitions. The Form Editor provides several editing keys.

Keystroke	**Use**
[CTRL]-[a]	Toggles between insert and overstrike mode.
[CTRL]-[x]	Deletes a character.
[CTRL]-[d]	Deletes to the end of a line.
[CTRL]-[u]	Undoes an edit.
[CTRL]-[v]	Marks and cuts a text block to the Clipboard (see “Using the Clipboard” on page 42).
[CTRL]-[t]	Cuts a text block and places it on the Clipboard.
[CTRL]-[p]	Pastes a text block.
[F1]	Inserts a blank line above current line.
[F2]	Deletes current line.
[ENTER]	Moves cursor to start of next line.
[HOME]	Moves cursor to top left comer of form.
[	Defines a new field.
]	Lengthens an existing field.
[ESC]	Toggles between pull-down menus and Form Editor:
[DEL]	Returns to pull-down menus.

Defining Fields

When painting the form image, you enter field labels and field attributes. You define a field in the Form Editor by pressing the left bracket ([) key. This causes the Define Fields window to appear.

You define fields and set field attributes in the Define Fields window. When you press the left bracket key from within the Form Editor, the Define Fields window appears.



```
 Form Editor:  [ESC] or [DEL] Command Line               [CTRL]-[w] Help
 Press [CTRL]-[z] to update definition for field "customer_num"
=====                                                           16)===
      Update: [ENT] to Store, [ESC] to Cancel          Help:
 Cust Enter changes into form                          [CTRL]-[w]     ]
 Comp ===================================================(Zoom)==
                          Define Fields
  Cit ---------------------------------------------------------       ]
      Table Name :  trorders                Input Area :  1
   Or Column Name:  customer_num            Entry ?    :  Y           ]
      Field Type :  integer                 Autonext ? :
 Ship Message    :   Enter the trcustomer code.    Downshift ?:
 Item Picture    :                          Upshift ?  :           nsio
[   ] Display Fmt:                          Verify ?   :              ]
[   ] Validate   :                          Required ? :              ]
[   ] Default    :                          Skip ?     :              ]
[   ] Translate  :                          Calendar ? :  N           ]
                                            Zoom ?     :  Y        ====
                                            Page:                     ]
      ---------------------------------------------------------       ]
 Ente Enter table name (or 'formonly').
```

In the Define Fields window you specify the attributes of the field. The attributes are arranged in the window so that the most important and least modified values are supplied first.

Most important are the Table Name and Column Name fields. You can enter values into these two fields directly or use Zoom to select from a list of available values.

The Field Type column is automatically filled in when you enter a valid column name in the Column Name field. You cannot modify the Field Type field because it relates to the column as defined in the database. If you specify Table Name as formonly, you are able to specify a value in the Field Type column.

The Input Area field specifies whether the field is on the header (1) or detail (2) part of the form.

The Entry? field is a YIN field that determines whether the field is for display purposes only or if it accepts input from the user.

The Message field stores a descriptive line that is displayed when the user positions the cursor in the field.

In the Picture field, you can add a character pattern for displaying the data. For example, area code and phone number fields might display use (###) ###-#### as their character pattern.

The Display Fmt field serves as a hybrid attribute for FORMAT and DISPLAY LIKE attributes, which are mutually exclusive. Refer to your Genero Business Development manuals for more information on these attributes.

The Validate field is similar to Display Fmt. It covers the INCLUDE and VALIDATE LIKE attributes. These attributes are also mutually exclusive. Again, refer to your Genero Business Development manuals for more information on these attributes.

The Default field lets you set a default value to appear in the field. The user can change default field values.

The Translate field lets you indicate which language you want to use to display data for this field. If specified, translation logic is generated for this field.

The Calendar field lets you specify that you want a calendar to be accessible when entering data for a date field

The Zoom field lets you specify that a zoom window will be accessible for this data entry field. This causes the magnifying glass to display to the right if the entry field at runtime.

The remaining fields are Y IN fields. You can experiment with these fields to see how they affect your input field.

Marking, Copying, and Pasting

When painting your form image, you can cut and paste fields and text. Copying consists of marking a block of text using the arrow keys and selecting the Copy option from the Edit pull-down menu. Once copied, you can paste the text block anywhere in your form image.

To mark and copy a text block:

1. **Position the cursor at one comer of the block of text you want to cut.**

2. **Press [CTRL]-[v] to start the Mark feature.**

3. **Use the arrow keys to highlight the entire block of text you want to mark.**

   As you move the cursor, the text you mark appears in reverse video.

4. **When you finish marking the entire block, press [CTRL]-[v] to copy the text block to the Clipboard.**

To paste a text block back onto your form image:

1. **Position the cursor on the form image where you want the block to appear.**

2. **Press [CTRL]-[p] to paste the block from the Clipboard to the form image.**

3. **Use the arrow keys to adjust where you want the block to stick.**

You can move the entire text block to any location on your form image before you stick it to the image.

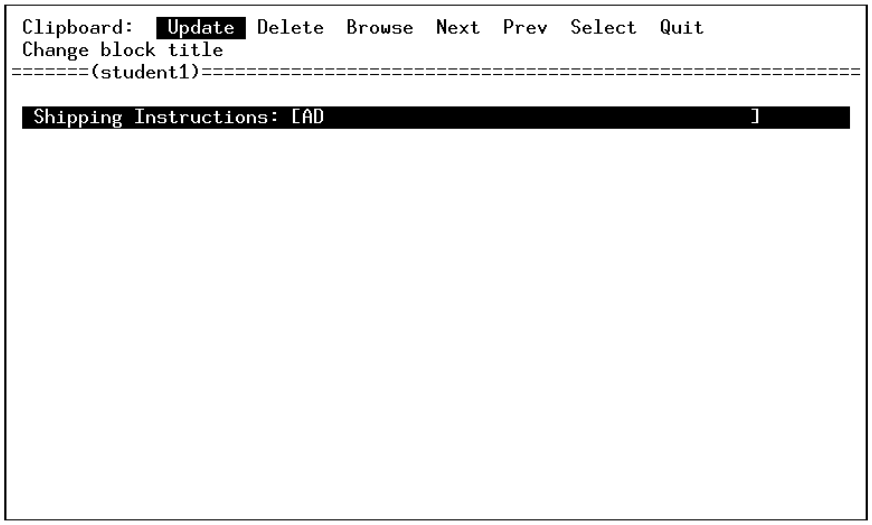
4. **Press [ESC] to stick the block to your form image.**

In a similar fashion, you can cut a block of text from your form image. Mark the block you want to cut as described above. Once you mark the text block, press [CTRL]-[t] to cut it. You can also paste a cut block back onto your form image in the same manner as described above.

Using the Clipboard

The Clipboard acts as a temporary storage place for text blocks. You can place anything onto the Clipboard and retrieve it. All the text you cut or copy gets placed on the Clipboard. Any text that you overwrite when you paste a block onto your form image gets stored to the Clipboard. You can access the Clipboard from the Edit pull-down menu.

The Clipboard acts as a temporary storage place for text blocks.

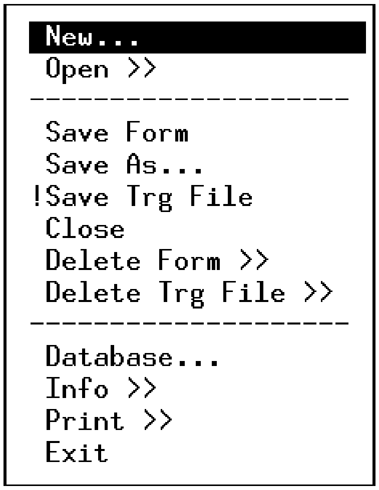


```
 Clipboard:  Update  Delete  Browse  Next  Prev  Select  Quit
 Change block title
=======(student1)=========================================================

  Shipping Instructions: [AD                                      ]
```

Saving a Form Image

After you paint your form image, you must save it with the Save Form option on the File pull-down menu.

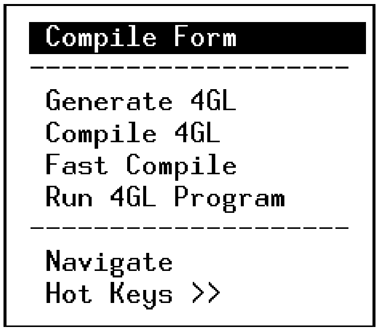
Use the Save Form option to save a form image.



Converting Forms into Input Programs

Once you create a form image and save it, you can run the *VDT Application* Code Generator and compilation utility from within the VDT Form Designer. The Run pull-down menu contains all the options necessary to convert your form into an input program.

The Run pull-down menu contains all the options necessary to convert your form into an input program.



In general, you can use the following Run pull-down menu options to convert your form into an input program:

1. **Generate 4GL -this option creates the 4GL source code.**

2. **Compile 4GL -this option compiles the 4GL code and links in library functions.**

3. **Run 4GL Program -this option runs the input program in the same manner a user would see it.**

Note

You can also run the *VTD Application* Code Generator and compilation utility from outside the VDT Form Designer (see “Starting the Tools from the Command Line” on page 137).

Section Summary

- The VDT Form Designer is an interactive utility that lets you develop complete data-entry programs written in 4GL.
- There are two commands that start the VDT Form Designer: fg.start and fg.form.
- The VDT Form Designer contains five pull-down menus. You can open a pull-down menu by highlighting it and pressing [ENTER].
- The VDT Form Designer lets you paint form images. You can use the VDT Form Designer to create a new form image or modify an existing form image.
- Once you load a form into the Form Editor, you can start painting the form image. Form images contain both text and input field definitions.
- When painting the form image, you enter field labels and field attributes. You define a field in the Form Editor by pressing the left bracket ([) key.
- When painting your form image, you can cut and paste fields and text. Copying consists of marking a block of text using the arrow keys and selecting the Copy option from the Edit pull-down menu.
- The Clipboard acts as a temporary storage place for text blocks.
- After you paint your form image, you must save it with the Save Form option on the File pull-down menu.
- Once you create a form image and save it, you can run the *VTD Application* Code Generator and compilation utility from within the VDT Form Designer.

Exercise 2A

Objective: To create a practice directory in which you will build your own input program.

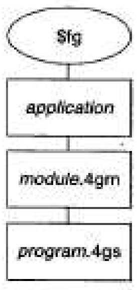
Create a Practice Directory Structure

In Exercise 1, you used the `scr_train 5` to build an input program. The `scr_train` command created a new shell for you to work in and placed you in the screen training "program" directory. When you create input programs--without using the screen training scrip--you must create your own directory structure.

VDT generated input programs use a four-tiered directory structure. The first tier is your fitrix directory or the directory specified by the `$fg` variable. For example:

```
$fg=/fitrix/fx_dev
```

The second tier is the application directory followed by the module directory and finally the program directory. The module and program directories use special naming extensions: *.4gm for the module directory and *.4gs for the program directory.



Before you build an input program with the VDT Form Designer, it helps to duplicate this directory structure.

1. **Move to your home directory:**

```
cd $HOME
```

2. **Create an application directory called labs:**

```
mkdir labs
```

3. **Move to your labs directory and create a module directory called aw. 4gm for Application Workbench:**

```
cd labs; mkdir aw.4gm
```

The semicolon delimits two LINUX commands.

4. **Move to your aw. 4gm directory and create a program directory called i_cust.4gs:**

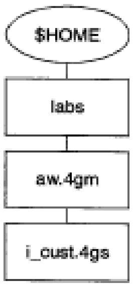
```
cd aw.4gm ; mkdir i_cust.4gs
```

Program directories reflect the type of programs they contain. Input program directories start with i-l which stands for input.

5. **Finally, move to the i_cust.4gs directory:**

```
cd i_cust.4gs
```

Once complete, you should be in the `i_cust.4gs` directory and have the following directory structure:



Exercise 2B

Objective: To start and become familiar with the VDT Form Designer.

Start the VDT Form Designer

From within the `i_cust.4gs` directory, you can use the VDT Form Designer to build an input program.

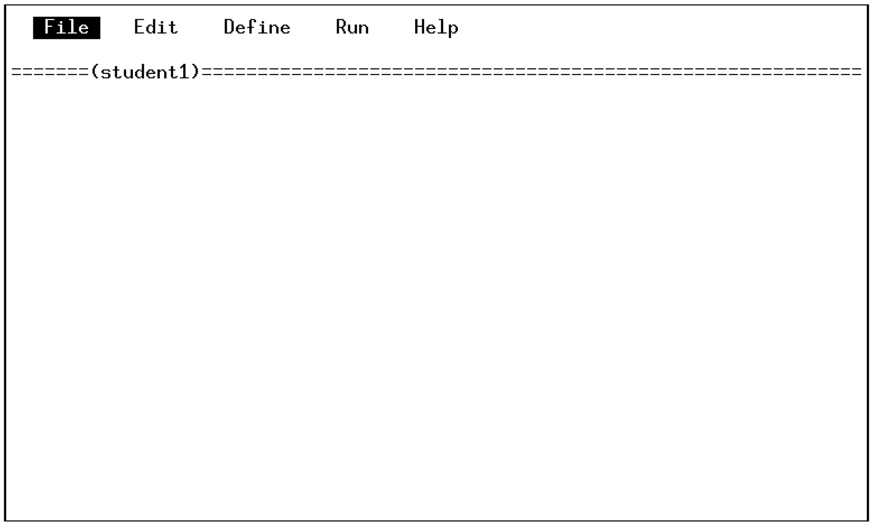
To start the VDT Form Designer, enter:

```
fg.form –dbname student1
```

Note

The `-dbname` flag specifies the database you want to use with the VDT Form Designer. If you have been set up to use a different database, specify it in place of `student1`. You may also set the DBNAME variable to the name of the database you wish to use.

After you enter the fg.form command, the VDT Form Designer appears:


File
Edit
Define
Run
Help
=======(student1)==

The VDT Form Designer lets you design input forms. In the next section you will build a Customer Entry program.

Exercise 2C

Objective: To use the VDT Form Designer to design a Customer Entry form.

There are several steps involved in designing a Customer Entry form. In general you should use the following sequence:

1. **Create a new form.**
2. **Add field labels.**
3. **Define which table and columns are used.**
4. **Save the form.**

Create Your New Form

The New option on the File pull-down menu lets you create a new form. For this exercise, you will make a header form called `cust`.

1. **Select New from the File pull-down menu.**

   The Define a New Form box appears.

2. **Enter oust into the Form Name field.**

   The "Select the screen type" box appears.

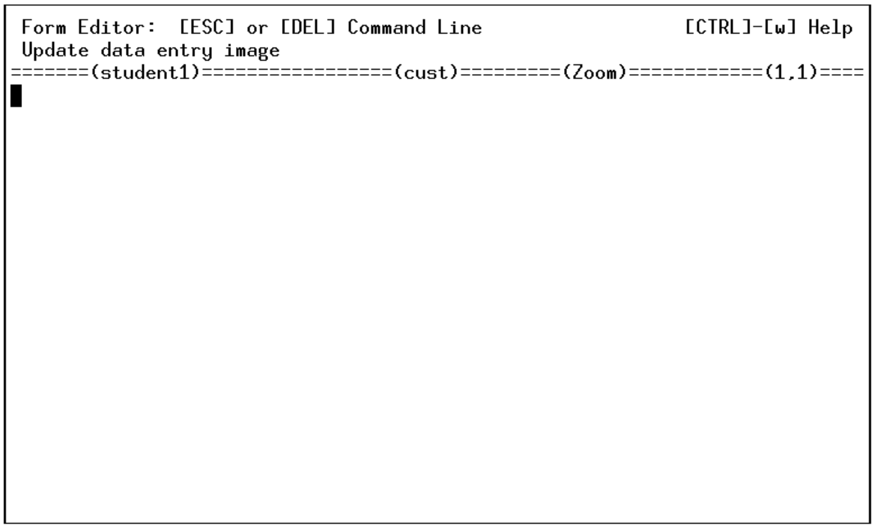
3. Choose header from the "Select the screen type" box.

A new form is created and the cursor is placed on the upper left comer of the form (at this point, the form is empty).

Add Field Labels

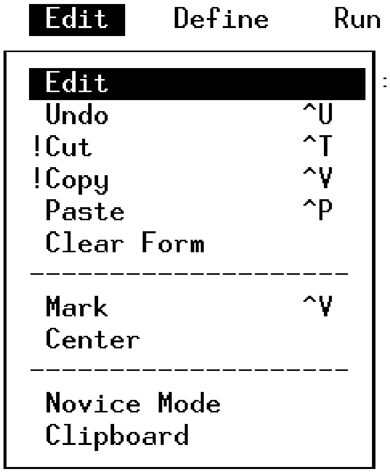
Once you create a new form, you can use the Form Editor to add input field labels. If you have just created a new form, your cursor is placed within the Form Editor automatically. The Form Editor lets you enter text and define input fields:

When you create a new form, your cursor is placed within the Form Editor automatically.



```
 Form Editor:  [ESC] or [DEL] Command Line                [CTRL]-[w] Help
 Update data entry image
=======(student1)=================(cust)=========(Zoom)============(1,1)====
█
```

The [ESC] key lets you toggle between the Form Editor and the pull-down menus. You can also move to the Form Editor by selecting Edit from the Edit pull-down menu.

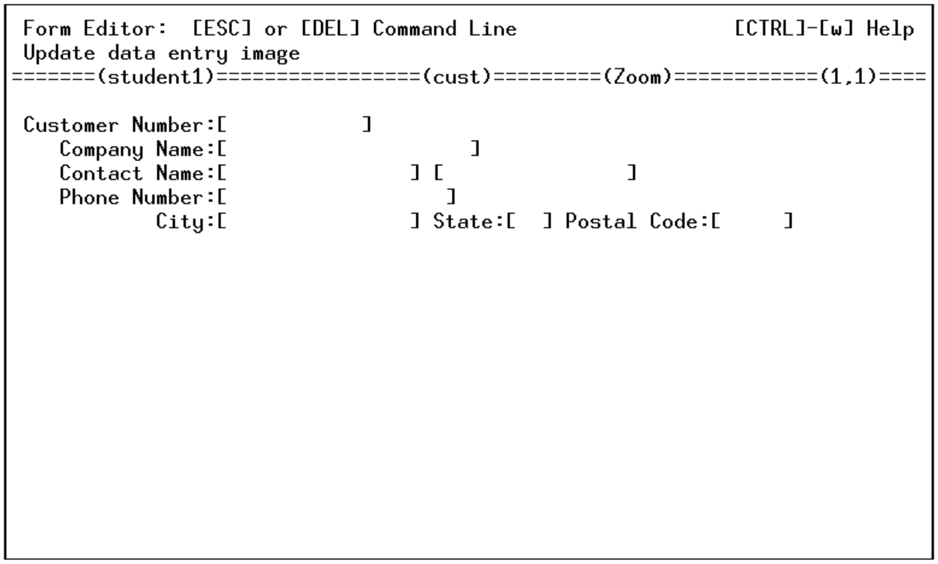


The Form Editor provides a number of useful editing keys and keystrokes to help you design your input form. The following list contains a few of them:

[F1]	Inserts a line.

[F2]	Deletes a line.
[ENTER]	Moves cursor to the start of the next line.
[HOME]	Moves cursor to the upper left corner.
[CTRL]-[a]	Toggles between insert and overstrike mode.
[CTRL]-[x]	Deletes a character.
[CTRL]-[d]	Deletes to the end of a line.
[CTRL]-[U]	Undoes an edit.

For this exercise, use the Form Editor to add input field labels that resemble the following form:



```
 Form Editor:  [ESC] or [DEL] Command Line                 [CTRL]-[w] Help
 Update data entry image
=======(student1)================(cust)=========(Zoom)============(1,1)====

 Customer Number:[           ]
    Company Name:[                    ]
    Contact Name:[               ] [               ]
    Phone Number:[                  ]
            City:[               ] State:[  ] Postal Code:[     ]
```

After you create all the labels, you can define the actual input fields themselves.

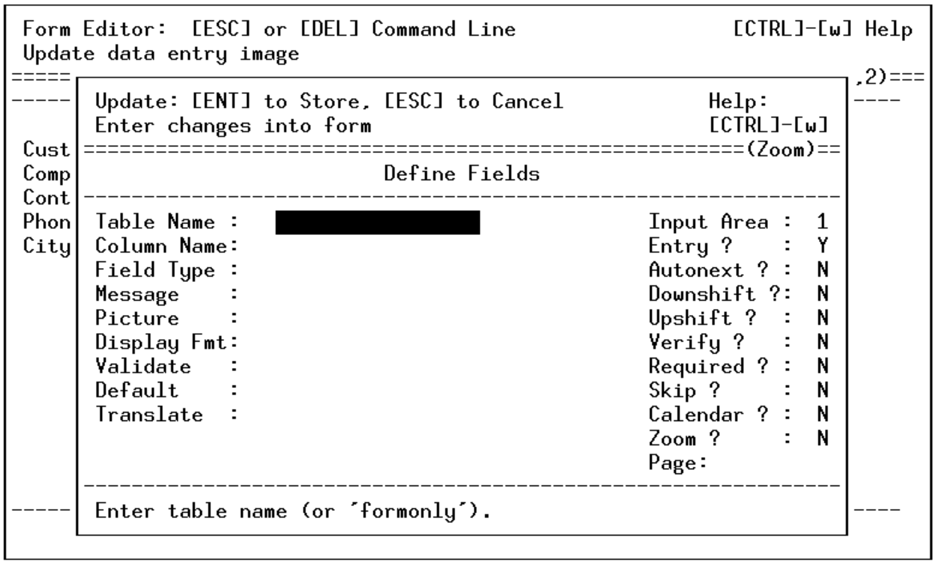
Define Input Fields

At this point, you need to define a corresponding field for each field label on your form. The Form Editor gives you a special key, the left bracket ([) key, for defining input fields.

Position your cursor to the right of the Customer Number field label you created.

1. **Press the left bracket ([) key.**

The Define Fields dialog window appears.



```
 Form Editor:  [ESC] or [DEL] Command Line                  [CTRL]-[w] Help
 Update data entry image
=====                                                                 ,2)===
-----| Update: [ENT] to Store, [ESC] to Cancel          Help:        |----
     | Enter changes into form                          [CTRL]-[w]   |
 Cust|=====================================================(Zoom)==|
 Comp|                        Define Fields                          |
 Cont|---------------------------------------------------------------|
 Phon| Table Name :                           Input Area :  1        |
 City| Column Name:                           Entry ?    :  Y        |
     | Field Type :                           Autonext ? :  N        |
     | Message    :                           Downshift ?:  N        |
     | Picture    :                           Upshift ?  :  N        |
     | Display Fmt:                           Verify ?   :  N        |
     | Validate   :                           Required ? :  N        |
     | Default    :                           Skip ?     :  N        |
     | Translate  :                           Calendar ? :  N        |
     |                                        Zoom ?     :  N        |
     |                                        Page:                  |
     |---------------------------------------------------------------|
-----| Enter table name (or 'formonly').                             |----
```

Input fields are associated with columns in a database table. They accept data from the user and insert it into a column. In this exercise, each field that you define will correspond to a column in the `customer` table.

Note

If you see a simplified version of this window, you are in "Novice mode." For all exercises in this training material, you must be in "Expert mode." The Edit pull-down menu contains an option that toggles between Expert and Novice mode. When Novice Mode is showing, it means you are in Expert mode and vice versa.

2. **Enter trcustomer in the Table Name field.**

3. **In the Column Name field, press [CTRL]-[z].**

   A list of all the columns in the trcustomer table appears.

4. **Highlight `customer_num` and press [ESC] to select it.**

   Data entered by the user into the Customer Number input field will go directly into this colunm in the customer database.

5. **Press [ENTER] to move to the Input Area field.**

   Notice that when you press [ENTER] the Field Type field gets filled in automatically with a serial not null value.

6. **Verify that the Input Area field contains a 1 and press [ENTER].**

   For now, all fields will have an Input Area of 1 (see "Input Areas Overview" on page 104). Place a 1 in this field.

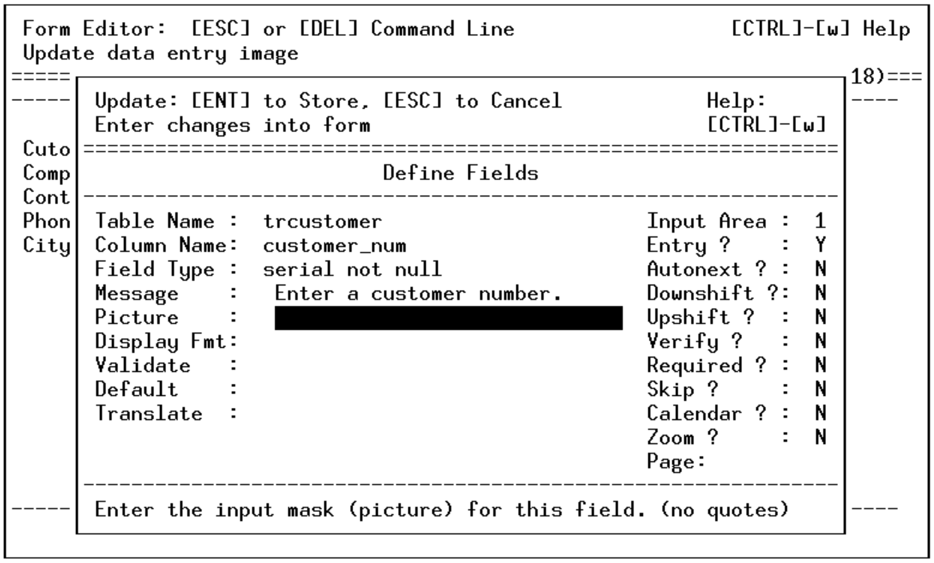
7. **Accept the Y value for the Entry? field and press [ENTER].**

   A Y value lets the user enter data into this field. An N specifies a no-entry field (i.e., a field in which the user cannot enter data).

8. **Type a message in the Message field and press [ESC].**

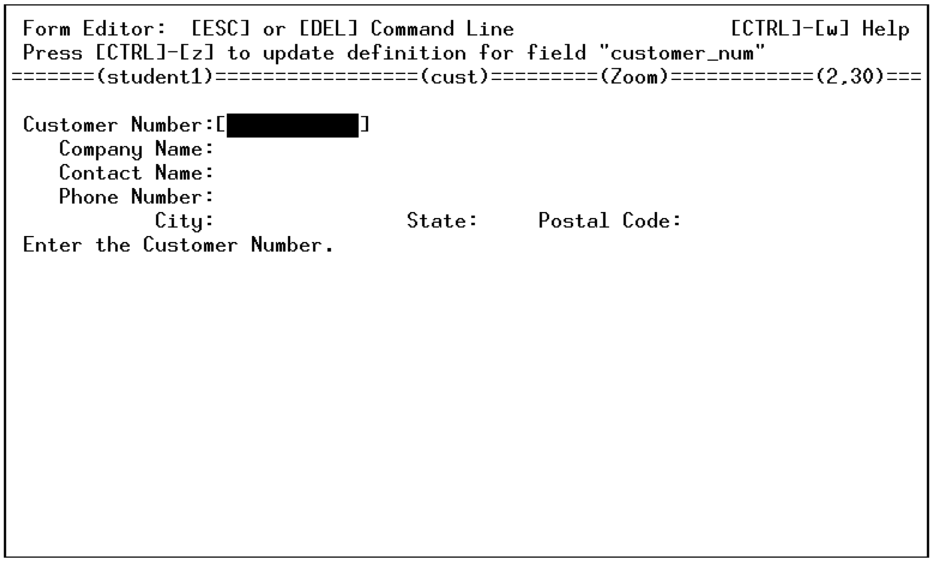
   This message will appear at the bottom of the form when the cursor is in the Customer Number field.

   For now, you can leave the other fields on the Define Fields window as is. The finished window should appear as follows:



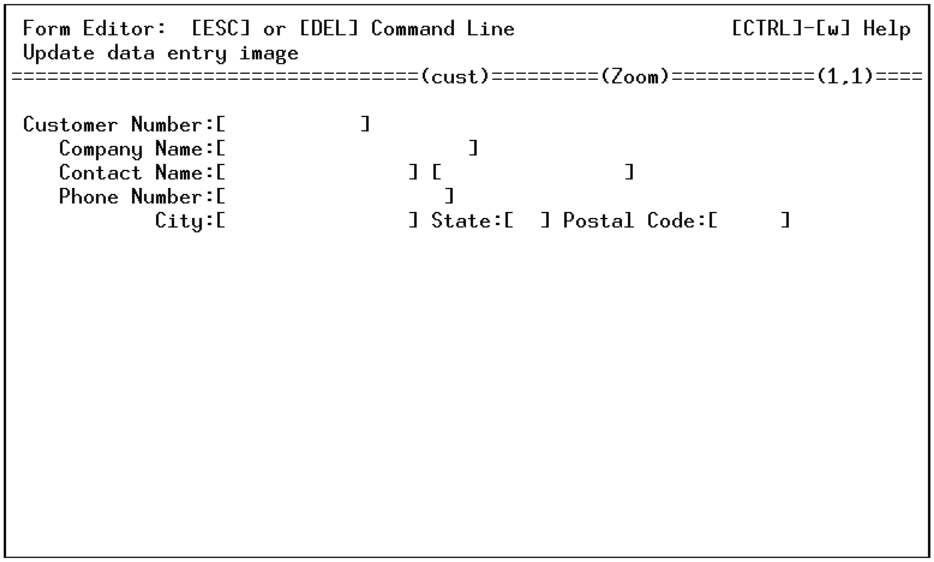
```
 Form Editor:  [ESC] or [DEL] Command Line             [CTRL]-[w] Help
 Update data entry image
=====                                                           18)===
-----   Update: [ENT] to Store, [ESC] to Cancel          Help:    ----
        Enter changes into form                          [CTRL]-[w]
 Cuto  ===========================================================
 Comp                         Define Fields
 Cont  -----------------------------------------------------------
 Phon   Table Name :  trcustomer                       Input Area :  1
 City   Column Name:  customer_num                     Entry ?    :  Y
        Field Type :  serial not null                  Autonext ? :  N
        Message    :   Enter a customer number.        Downshift ?:  N
        Picture    :                                   Upshift ?  :  N
        Display Fmt:                                   Verify ?   :  N
        Validate   :                                   Required ? :  N
        Default    :                                   Skip ?     :  N
        Translate  :                                   Calendar ? :  N
                                                       Zoom ?     :  N
                                                       Page:
       -----------------------------------------------------------
-----   Enter the input mask (picture) for this field. (no quotes)   ----
```

Once you save the Customer Number field definition, the field appears in the Form Editor as two brackets with a highlight between them. Notice also how the field is automatically sized and the field message appears at the bottom of the screen:



```
 Form Editor:  [ESC] or [DEL] Command Line             [CTRL]-[w] Help
 Press [CTRL]-[z] to update definition for field "customer_num"
=======(student1)================(cust)=========(Zoom)============(2,30)===

 Customer Number:[           ]
    Company Name:
    Contact Name:
    Phone Number:
            City:                State:     Postal Code:
 Enter the Customer Number.
```

Follow the same sequence of steps to define the rest of the input fields on your form. For the Contact Name field, define two fields (`fname` and `Iname`). When you finish, your form should look as follows:



```
 Form Editor:  [ESC] or [DEL] Command Line                [CTRL]-[w] Help
 Update data entry image
==================================(cust)=========(Zoom)============(1,1)====

 Customer Number:[           ]
    Company Name:[                    ]
    Contact Name:[               ] [               ]
    Phone Number:[                  ]
            City:[               ] State:[  ] Postal Code:[     ]
```

After you create a field definition, you might need to re-edit it at some point.

To re-edit a field definition:

1. **Place your cursor in the field and press [CTRL]-[z].**

   A pop-up menu appears.

2. **Select Field from the pop-up menu.**

   The Define Fields window appears.

3. **Edit the field definition using the Define Fields window and press [ESC] to save your changes.**

Save the Form

When you are satisfied with your input form, save it using the Save Form option under the File pull-down menu.

To save a form:

- **Select Save Form from the File pull-down menu.**

The VDT Form Designer reads your form image and generates instructions in a form specification (*.per) file. This file is used by the VDT Application Code Generator to create source code, which is discussed next.

Exercise 2D

Objective: To use the VDT Form Designer to generate, compile, and run your Customer Entry program.

Recall that you built a demonstration input program from the LINUX command line using `fg.screen`, `fg.make`, and `fglrun`. The VDT Form Designer gives you the same ability, but you simply select these commands from the VDT Form Designer's Run pull-down menu.

Generate Source Code

1. **Select Generate 4GL from the Run pull-down menu.**

   A pop-up menu appears asking you which forms to generate code for.

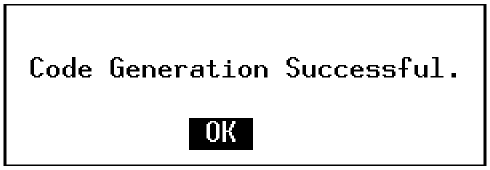
2. **Select All Forms from the pop-up menu.**

   A message box appears asking you if you want to only generate code for local forms.

3. **Select YES on the "Local forms only" message box.**

   The Screen Code Generator is run and code scrolls past your screen as it creates code based on your `cust` form. You might see a message indicating that your `cust` form is not current. If this happens, simply select YES from the message box.

   When the *Screen* Code Generator finishes, the following message appears:

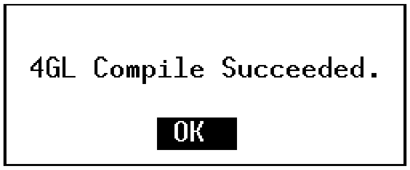


Code Generation Successful.

Compile the Code

- **Select Compile 4G L from the Run pull-down menu.**

The VDT Form Designer calls the compilation utility and creates a program file. When done, the following message appears:

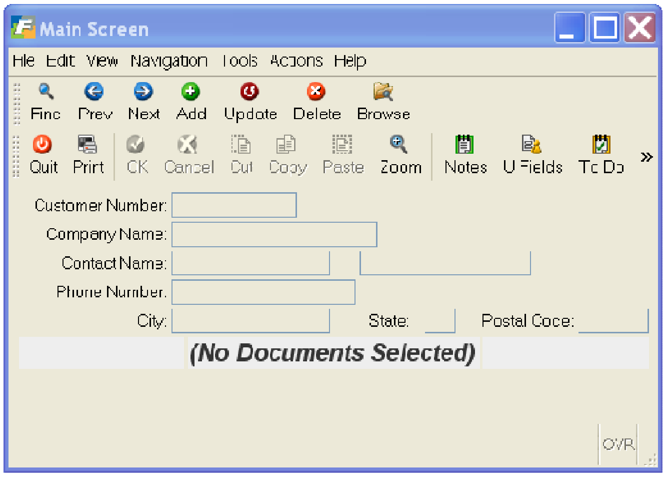


4GL Compile Succeeded.

Run Your Customer Entry Program

1. **Select Run 4GL Program from the Run pull-down menu.**

The VDT Form Designer runs your Customer Entry program.



Main Screen
File Edit View Navigation Tools Actions Help
Find Prev Next Add Update Delete Browse
Quit Print OK Cancel Cut Copy Paste Zoom Notes U Fields To Do
Customer Number:
Company Name:
Contact Name:
Phone Number:
City:
State:
Postal Code:
OVR

(No Documents Selected)

2. **Use the Toolbars to "test drive" your input program. When you finish, select Exit to return to the VDT Form Designer.**

Exercise 2E

Objective: To make a slight change to your Customer Entry program and then rebuild it.

At times, you may want to make changes to your form and incorporate those changes into your generated-input program. For example you may want to move a field label and definition to a different location. The VDT Form Designer makes this task easy.

In this exercise you will use the VDT Form Designer's Mark, Cut, and Copy options to move the Phone Number field to a new location on the `cust` form. Once finished, you will save `cust` and rebuild an input program from it. The resulting input program will reflect the change you made.

Note

This exercise picks up where the Exercise 2C left off. You should be in the VDT Form Designer and have your cust form visible in the Form Editor. If you are not at this point, use the steps in the previous sections to catch up.

In general there are three steps to moving a portion of your form:

1. Mark the portion you want to move.

2. Cut the marked portion.

3. Paste the cut portion back onto the form in the appropriate spot.

Mark the Phone Number Field

Before you can move a portion of your `cust` form, you must mark it. You can mark anything that appears on your form: field labels, field definitions, or both.

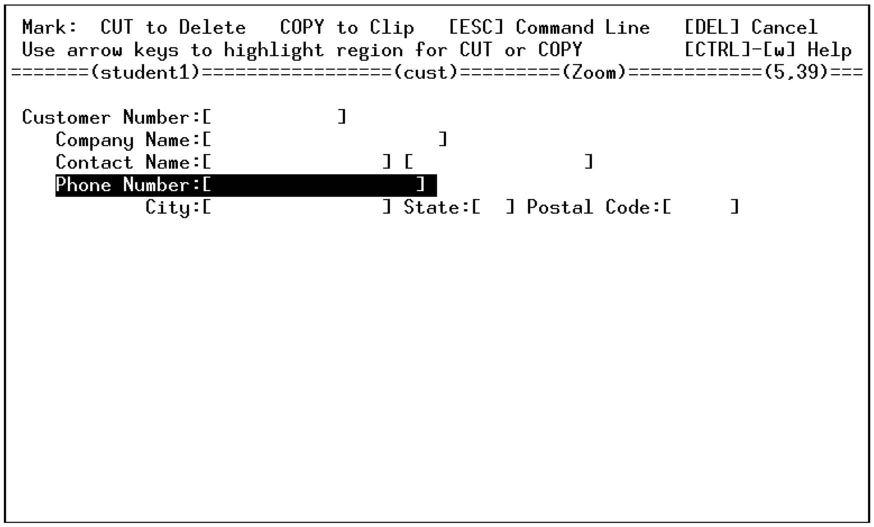
1. **If you are not in it already, move to the Form Editor: Select the Edit option under the Edit pull-down menu.**

2. **Move the cursor to the start of the Phone Number field.**

3. **Press [CTRL]-[v].**

   This keystroke places you into "Mark" mode.

4. **Use the arrow keys to highlight the Phone Number field.**



```
 Mark:  CUT to Delete   COPY to Clip   [ESC] Command Line   [DEL] Cancel
 Use arrow keys to highlight region for CUT or COPY         [CTRL]-[w] Help
=======(student1)=================(cust)=========(Zoom)============(5,39)===

 Customer Number:[            ]
    Company Name:[                    ]
    Contact Name:[               ] [               ]
    Phone Number:[                  ]
            City:[               ] State:[  ] Postal Code:[     ]
```

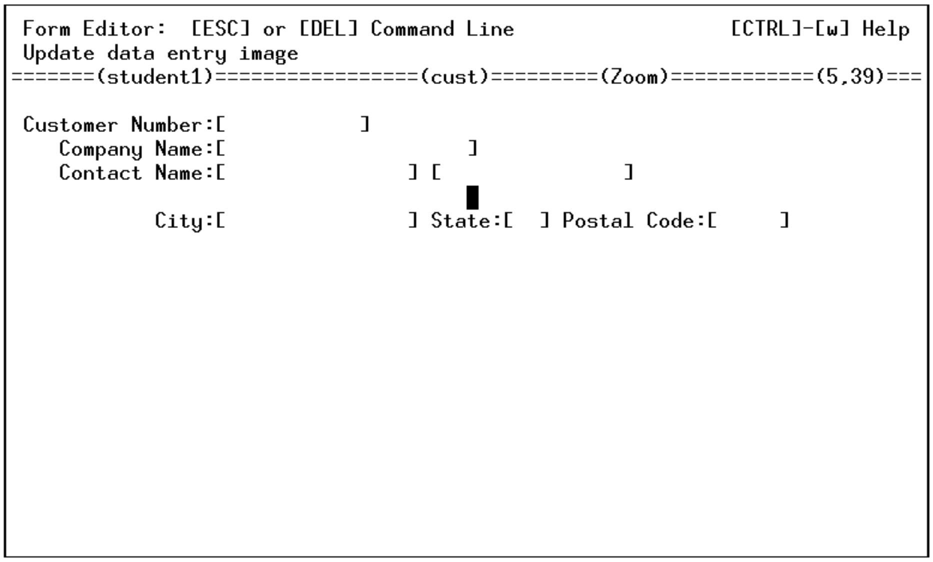
This is the area that you will cut.

Cut the Phone Number Field

Once you mark (ie., highlight) the Phone Number field, you can cut it from your form (once cut, you can paste it back into your form at any location).

- **Press [CTRL]-[t].**

The Phone Number field disappears.

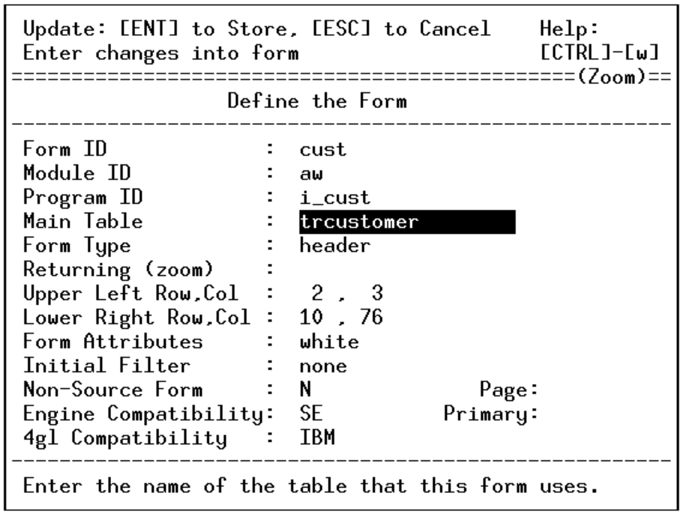


```
 Form Editor:  [ESC] or [DEL] Command Line                  [CTRL]-[w] Help
 Update data entry image
=======(student1)=================(cust)=========(Zoom)============(5,39)===

 Customer Number:[           ]
    Company Name:[                    ]
    Contact Name:[               ] [               ]

            City:[               ] State:[  ] Postal Code:[     ]
```

Now you can use the Paste option to place this field below the City-State, and Postal Code line.

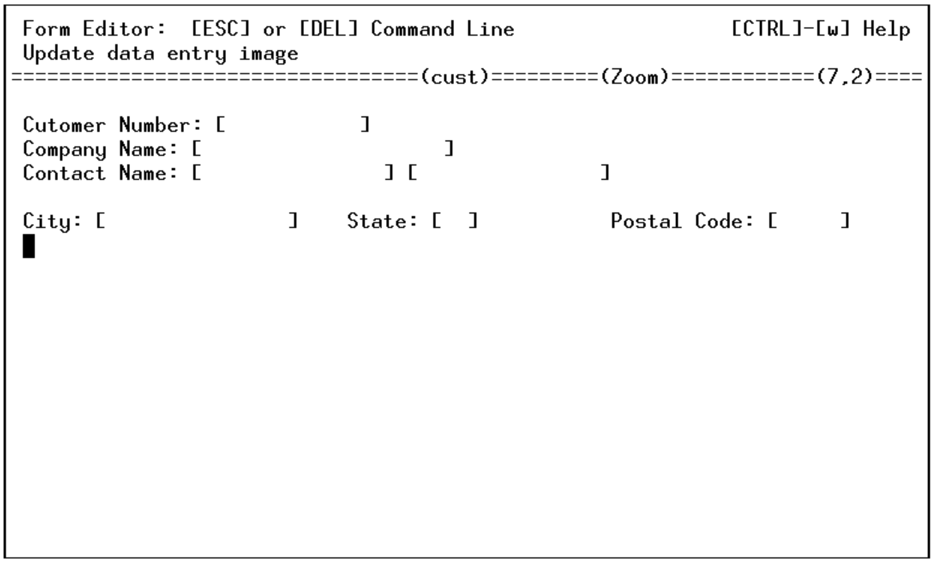
Paste the Phone Number Field Back into Your Form

1. **Move your cursor below the City field. If you cannot move the cursor below the City field, you need to increase the size of your form. To increase the size of your form, you exit Edit mode by pressing [ESC]. Choose Define Defaults option under the Define pull-down menu. The Define the Form window appears:**



```
 Update: [ENT] to Store, [ESC] to Cancel    Help:
 Enter changes into form                    [CTRL]-[w]
==============================================(Zoom)==
                  Define the Form
------------------------------------------------------
 Form ID             :  cust
 Module ID           :  aw
 Program ID          :  i_cust
 Main Table          :  trcustomer
 Form Type           :  header
 Returning (zoom)    :
 Upper Left Row,Col  :   2 ,  3
 Lower Right Row,Col :  10 , 76
 Form Attributes     :  white
 Initial Filter      :  none
 Non-Source Form     :  N            Page:
 Engine Compatibility:  SE        Primary:
 4gl Compatibility   :  IBM
------------------------------------------------------
 Enter the name of the table that this form uses.
```

Move to the Lower Right Row, Col field and increase the Row to 11 to add one more line. Press [ESC] to save your changes. Choose the Edit option from the Edit pull-down menu. Move your cursor to below the City field.



```
 Form Editor:  [ESC] or [DEL] Command Line                [CTRL]-[w] Help
 Update data entry image
==================================(cust)=========(Zoom)============(7,2)====

 Cutomer Number: [           ]
 Company Name: [                    ]
 Contact Name: [               ] [               ]

 City: [               ]    State: [  ]          Postal Code: [      ]
 █
```

2. **Press [CTRL]-[p].**

The Phone Number field reappears. You can use the arrow keys to "slide" the field around, but for now, leave it where it is.

Note

Be sure to never paste on top of existing form objects. You must paste on a blank space or line.

3. **Press [ESC] to "stick" the field to the form.**

Use the Clipboard

If you make a mistake during cutting and pasting, you can select the Clipboard option from the Edit pull-down menu. Everything you cut gets placed on its own page in the Clipboard. You can use the Clipboard's ring menu to scroll through all the objects you have cut and select the one you want.

When you select an object from the Clipboard, it gets pasted into your form (just like the Phone Number field). Once again, you can reposition the object with the arrow keys before pressing [ESC] to "stick" it to the form.

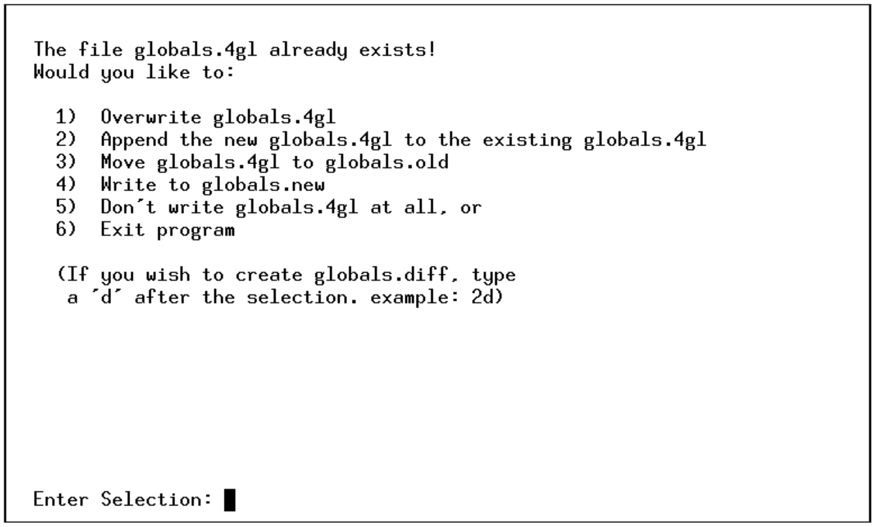
Save Your Changes

Now that you have moved the Phone Number field, you can save your form and rebuild it. Once rebuilt, the resulting input program will reflect the new location of the Phone Number field.

1. **Save your `cust` form with the Save Form option under the File pull-down menu.**

2. **Select Generate 4GL from the Run pull-down menu.**

   During code generation, the "Overwrite" message might appear:



```
The file globals.4gl already exists!
Would you like to:

  1)  Overwrite globals.4gl
  2)  Append the new globals.4gl to the existing globals.4gl
  3)  Move globals.4gl to globals.old
  4)  Write to globals.new
  5)  Don´t write globals.4gl at all, or
  6)  Exit program

  (If you wish to create globals.diff, type
   a ´d´ after the selection. example: 2d)









Enter Selection: █
```

   This message lets you know that you are creating a "new" source code file on top of a file that already exists in your `i_cust.4gs` directory. For this exercise – and in most cases for that matter – you want to overwrite this file. Depending on the number of changes you have made, you might see this message several times.

   When it appears, simply select option one to overwrite the file.

3. **Select Compile 4GL from the Run pull-down menu.**

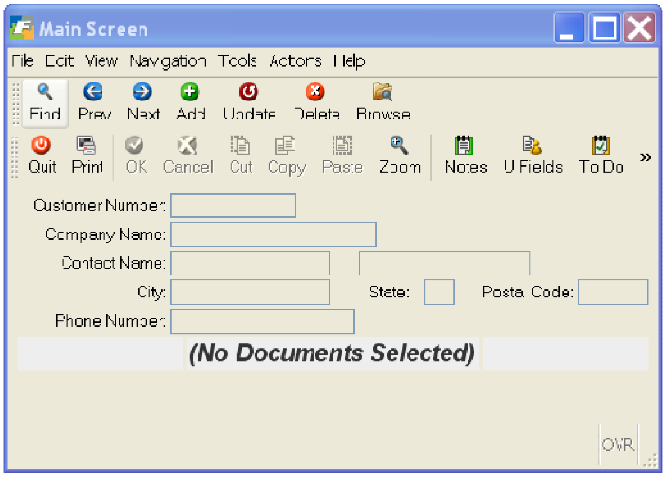
   Once compiled, your program is ready for you to run.

Run the Customer Entry Program Again

Now you can see your changes in the resulting input program.

- **Select Run 4GL Program from the Run pull-down menu.**

  The VDT Form Designer initiates your Customer Entry program.


Main Screen
File Edit View Navigation Tools Actions Help
Find Prev Next Add Update Delete Browse
Quit Print OK Cancel Cut Copy Paste Zoom Notes U Fields To Do
Customer Number:
Company Name:
Contact Name:
City:
State:
Postal Code:
Phone Number:
OVR

(No Documents Selected)

Notice that the Phone Number field appears in its new location. Once again, spend some time experimenting with this program. Add a new document and see if the cursor path through your input fields has changed.

When you are done, select Exit from the ring menu to return to the VDT Form Designer. Exit out of the VDT Form Designer as well (select Exit from the File pull-down menu).

Chapter 3

Working with the Database

Main topics:

- Displaying the Table Information Window
- Changing Database Values
- Updating Genero Schema
- Using the AutoForm option

Displaying the Table Information Window

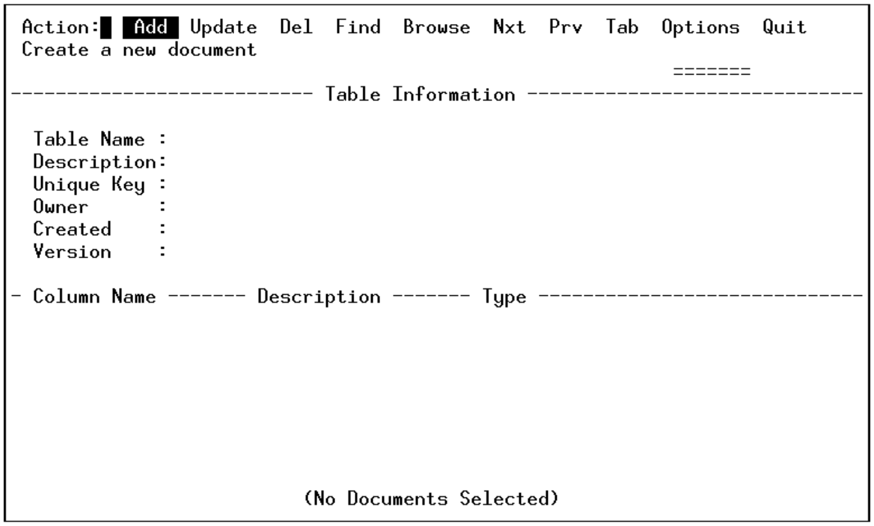
The VDT Form Designer gives you direct access to the database through the Table Information window. This window lets you manage tables and columns in the database.

To initiate the Table Information window:

- **Select Database from the File pull-down menu.**

  The Table Information window appears.

Use the Database option on the File pull-down menu to initiate the Table Information window.



```
Action:  Add  Update  Del  Find  Browse  Nxt  Prv  Tab  Options  Quit
Create a new document
                                                        =======
-------------------------- Table Information -----------------------------

 Table Name :
 Description:
 Unique Key :
 Owner      :
 Created    :
 Version    :

- Column Name ------- Description ------- Type ----------------------------




                         (No Documents Selected)
```

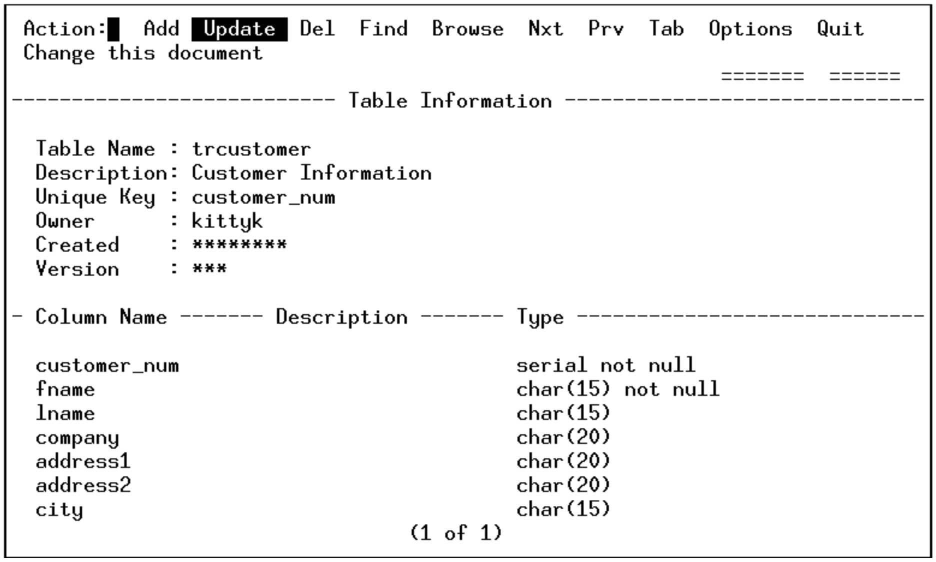
The Table Information window lets you do the following:

- Alter the structure of your database
- Add and drop database tables
- Add, modify, and drop columns from tables

Changing Database Values

The Table Information window uses a ring menu interface. The commands in the ring menu at the top of the window correspond to the toolbar commands you are familiar with.

The Table Information window uses a ring menu interface.



```
Action:█  Add  Update  Del  Find  Browse  Nxt  Prv  Tab  Options  Quit
Change this document
                                                               =======  ======
--------------------------- Table Information ------------------------------

  Table Name : trcustomer
  Description: Customer Information
  Unique Key : customer_num
  Owner      : kittyk
  Created    : ********
  Version    : ***

- Column Name ------- Description ------- Type -----------------------------

  customer_num                            serial not null
  fname                                   char(15) not null
  lname                                   char(15)
  company                                 char(20)
  address1                                char(20)
  address2                                char(20)
  city                                    char(15)
                                 (1 of 1)
```

You can add tables to the database and give them descriptive names. It is very important to fill in the Unique Key field. This field identifies to an input program the columns that uniquely identify a row.

When you use the Table Information window to alter a table (for example, you delete a column), a pop-up window appears and displays the SQL statement that it will run on the table.

The VDT Form Designer stores all the changes you make in a file called `dbadmin.sql`. All changes are also time stamped, and this file remains in your local program directory.

Saving Database Changes

It is good practice to save all custom database changes to a central directory. The convention is to save such changes to the $fg/data/sql.4gc directory. You save the following types of changes to the $fg/data/sql.4gc directory.

- create statements for any new tables created
- alter statements for any tables altered

- Any rows added to system tables. These tables are discussed in later exercises.
    - stxerorh/sterord (error tables)
    - stxmssgr (message table)
    - stxhotkd/stxnvgtd/stxactnr (hot key and event tables)
    - cgsmnitm/cgsmncmd (menu items)

Updating the Genero Schema File

Once you have altered a table, added a table or dropped a table in a database, you must update the Genero schema file for that database. The Genero schema file is used by the Genero compiler.   If you do not update the Genero schema file, you receive errors when you compile you programs. Both the fglform and fg.make commands depend on an up-to-date Genero schema file.

The Genero schema file must exist in the directory(s) listed in `$FGLDBPATH`. The convention is to set `FGLDBPATH=$fg/data.`

You update the Genero schema file by changing directory to where they are kept ($FGLDBPATH=$fg/data). Then you run the sch.sh command on the database:

```
   sch.sh student1
```

When you run `sch.sh <database>` a `<database>.sch` file is created. You must run the `sch.sh` command for all the databases you are using. You will see a <database>.sch file in the $fg/data directory for each database.

Warning: You can make a schema file (example, standard.sch) in your local program directory and that Genero schema file takes precedent. But when you run other programs, they will not find the updated schema file because they use the schema file in $fg/data. Remember to update the Genero schema file in $fg/data.

Using the AutoForm Option

The Table Information window also lets you generate a default form image from a table; in other words you can create an AutoForm. The AutoForm command is located under the Options command on the ring menu. When you create an AutoForm, the AutoForm image gets stored to the Clipboard. You can then quit from the Table Information window and paste the AutoForm image into your form image using the Form Painter.

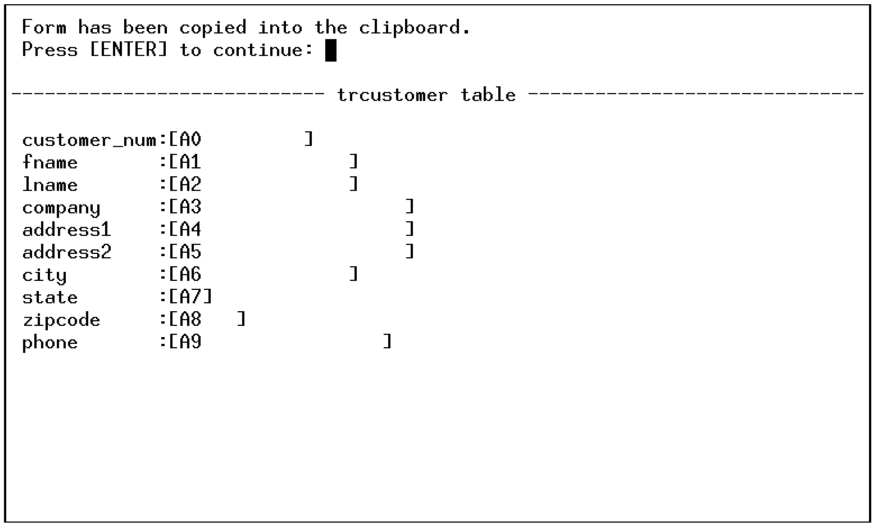
To create an AutoForm:

1. **Use Find to select the table you want to generate an AutoForm from.**

2. **Select the Options command then AutoForm.**

   An AutoForm gets built and its image is stored to the Clipboard.

Use the AutoForm command to generate a default image of a table.



```
 Form has been copied into the clipboard.
 Press [ENTER] to continue: █

---------------------------- trcustomer table ------------------------------

 customer_num:[A0         ]
 fname       :[A1             ]
 lname       :[A2             ]
 company     :[A3                  ]
 address1    :[A4                  ]
 address2    :[A5                  ]
 city        :[A6             ]
 state       :[A7]
 zipcode     :[A8   ]
 phone       :[A9                ]
```

Once you create an AutoForm, you can go back to the VDT Form Designer and retrieve the AutoForm from the Clipboard. Once retrieved, the AutoForm is placed into the Form Editor, and you can edit it any way you want.

Section Summary

- The VDT Form Designer gives you direct access to the database through the Table Information window. This window lets you manage tables and columns in the database.
- With the Table Information window you can alter the structure of your database; add and drop database tables; and add, modify, and drop columns from tables.
- It is good practice to save all custom database changes to a central directory ($fg/data/sql.4gc).
- Use the sch.sh <database> command to update the Genero schema files in $fg/data each time you change the structure of a database.
- The Table Information window also lets you generate a default form image from a table; in other words, you can create an AutoForm.

Exercise 3

Objective: To create a credit table that holds credit codes, descriptions, and amounts. Such a table could hold the following values:

Credit Code	Credit Description	Credit Amount
AAA	Excellent	10,000
BBB	Good	5,000
CCC	Fair	1,000
DDD	Poor	250

Start the VDT Form Designer

1. **Move to the $HOME/labs/ aw.4gm directory:**

   cd $HOME/labs/aw.4gm

2. **Create a new directory to hold a credit entry program.**

   ```
   mkdir i_cred.4gs
   ```

3. **Move to the i_cred.4gs directory:**

   ```
   cd i_cred.4gs
   ```

4. **Start the VDT Form Designer.**

Create a New Form

1. **From the File pull-down menu, select New.**

   The Define a New Form box appears.

2. **Name the form credit**

   The Select the Screen Type box appears.

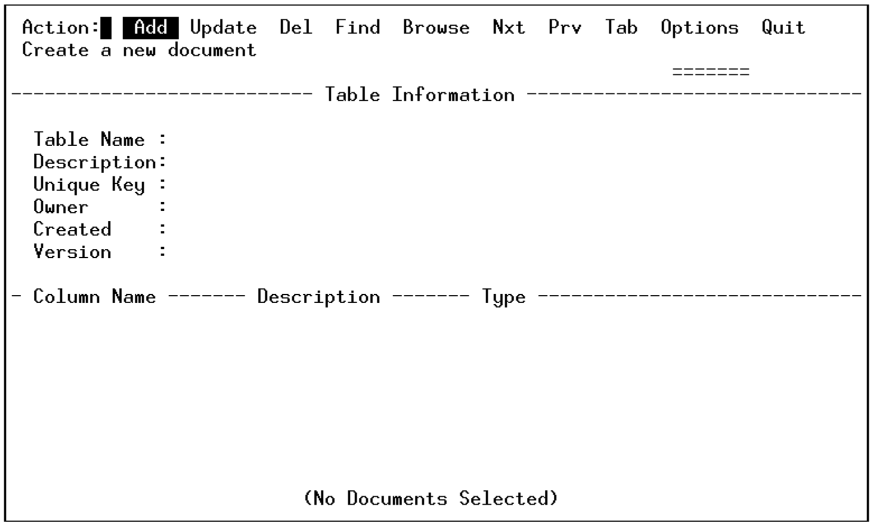
3. **Choose header as the screen type.**

4. **Type [ESC] to exit from edit mode.**

Open the Database Option

1. **From the File pull-down menu, select Database.**

   The Table Information window appears.



```
Action:█ Add  Update  Del  Find  Browse  Nxt  Prv  Tab  Options  Quit
Create a new document
                                                          =======
--------------------------- Table Information ------------------------------

  Table Name :
  Description:
  Unique Key :
  Owner      :
  Created    :
  Version    :

- Column Name ------- Description ------- Type -----------------------------






                          (No Documents Selected)
```

The Database option is a data-entry program that allows you to change the structure of your database. You can add, delete, and alter tables by adding, deleting, and re-arranging columns, and changing column types. You can change the structure of your database much like using Informix dbaccess to do so.

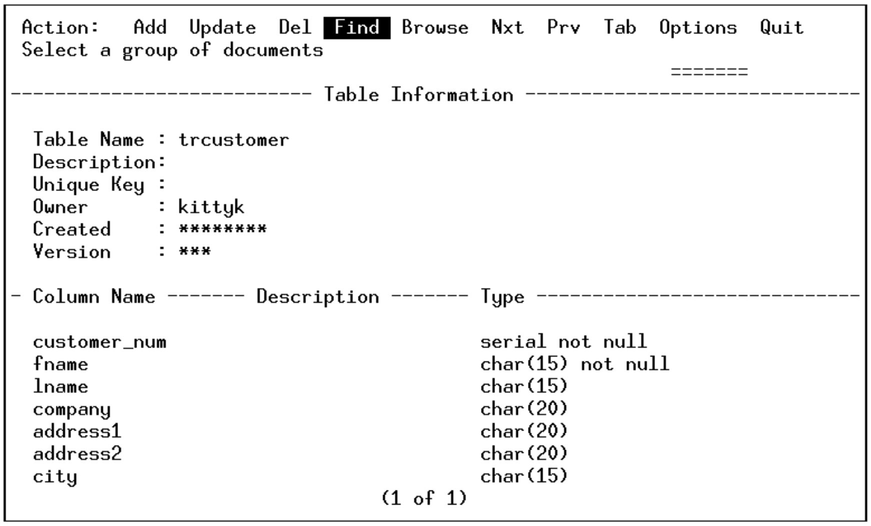
2. **Select the Find ring menu option.**

   Your cursor moves to the Table Name field.

3. **Type trcustomer in the Table Name field.**

4. **Press [ESC].**

   Information about the customer table appears.



```
 Action:   Add  Update  Del  Find  Browse  Nxt  Prv  Tab  Options  Quit
 Select a group of documents
                                                             =======
--------------------------- Table Information ------------------------------

  Table Name : trcustomer
  Description:
  Unique Key :
  Owner      : kittyk
  Created    : ********
  Version    : ***

- Column Name ------- Description ------- Type -----------------------------

  customer_num                        serial not null
  fname                               char(15) not null
  lname                               char(15)
  company                             char(20)
  address1                            char(20)
  address2                            char(20)
  city                                char(15)
                               (1 of 1)
```

Notice how the upper half of the screen (the "header") portion contains information about the `trcustomer` table. The lower half of the screen (the "detail" portion) displays all of the columns that make up the `trcustomer` table.

Add the credit Table

1. **Select the Add ring menu option.**

2. **Add a table to hold credit information.**

   Name your new table credit and add a descriptive name for the table. Do not enter a Unique Key value yet.

3. **Press [TAB] to move to the detail portion of the screen and add the following columns as detail rows:**

Column Name	Description	Type
credit_code	Credit Code	Char(3)
credit_desc	Credit Description	Char(10)
credit_amt	Credit Amount	Decimal(10,2)

4. **Press [TAB] to move back to the header portion of the screen and fill in the Unique Key as credit_code.**

   All tables must have a unique key (i.e., a column that uniquely identifies a row).

5. **Press [ESC] to store your new table.**

Create an AutoForm from the credit Table

1. **Select the Options ring menu and then choose AutoForm.**

   This builds a default data-entry form based on your credit table. It then copies this form to the Clipboard. Once on the Clipboard, you can paste it into a new form.

2. **Press (ENTER].**

3. **Select Quit from the ring menu to return to the Form Painter.**

Use the Clipboard

Instead of creating fields individually, you can copy the AutoForm you created and stored on the Clipboard.

1. **Select Clipboard from the Edit pull-down menu.**

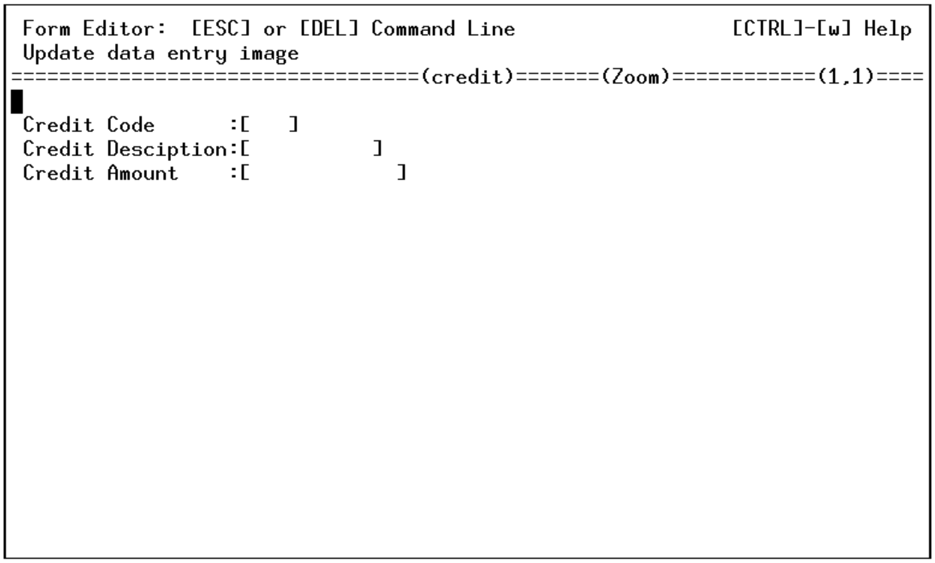
   Find the AutoForm for the credit table.

2. **Once you find the credit table AutoForm, choose Select.**

   The Select option pastes the AutoForm into the Form Editor. You can use the arrow keys to position it.

3. **Press [ESC] to stick it down.**

   Remove the extra heading line that came with the AutoForm.



```
 Form Editor:  [ESC] or [DEL] Command Line                  [CTRL]-[w] Help
 Update data entry image
===================================(credit)=======(Zoom)============(1,1)====
█
 Credit Code      :[   ]
 Credit Desciption:[          ]
 Credit Amount    :[            ]
```

Save, Generate, and Compile

1. **Save your newly-created form.**

   Use the Save Form option under the File pull-down menu.

2. **Exit out of the VDT Form Designer. You must update the Genero schema used by the Genero compiler. Choose Exit from the File pull-down menu.**

3. **Move to the schema directory.**

   ```
   cd $fg/data
   ```

4. **Generate the Genero schema file (substitute your database name).**

   ```
   sch.sh student1
   ```

5. **Save the trcredit create statement to the central database changes directory.**

   ```
   cd $fg/data
   mkdir sql.4gc
   cd sql.4gc
   cp $HOME/labs/aw.4gm/i_cred.4gs/dbadmin.sql trcedit.sql
   ```

6. **Restart the VDT Form Designer in your new program directory (substitute your database name).**

   ```
   cd $HOME/labs/aw.4gm/i_cred.4gs
   fg.form  -dbname student1
   ```

7. **Open the form you saved by selecting Open from the File pull-down menu. Select [ESC] to exit edit mode.**

8. **Select Generate 4GL from the Run pull-down menu.**

   When it is finished, the Code Generation Successful message appears.

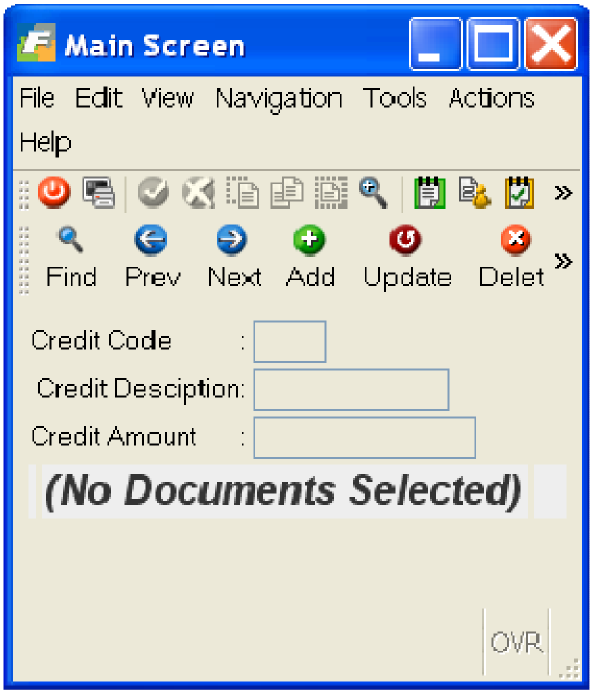
9. **Select Compile 4GL from the Run pull-down menu.**

   When it is finished, the 4GL Compile Succeeded message appears.

Run Your Credit Entry Program

1. **Select Run 4GL Program from the Run pull-down menu.**

   The Credit Entry program starts.


Main Screen
File Edit View Navigation Tools Actions
Help
Find Prev Next Add Update Delet
OVR

Credit Code :

Credit Desciption:

Credit Amount :

(No Documents Selected)

2. **Enter at least four new credit codes.**

   You can use the sample codes shown on page 69.

3. **When finished, exit the program and the Form Painter.**

Chapter 4

Creating Zooms

Main topics:

- Zoom Screen Overview
- Painting a Zoom Image
- Attaching the Zoom Screen

Zoom Screen Overview

A Zoom is a data validation feature that shows the user a list of valid values for an input field Zooms are created from zoom screen types (see ”Using Different Screen Types” on page 182) When users initiate a Zoom, they can enter selection criteria on the fields in the Zoom. The Zoom then takes the selection criteria and returns all valid values that meet the criteria Users can select the value they want to use from the values the Zoom returns.

Zooms make the data-entry process much more accurate and efficient. Field values are validated before they are saved. In general, creating Zooms is a two step process:

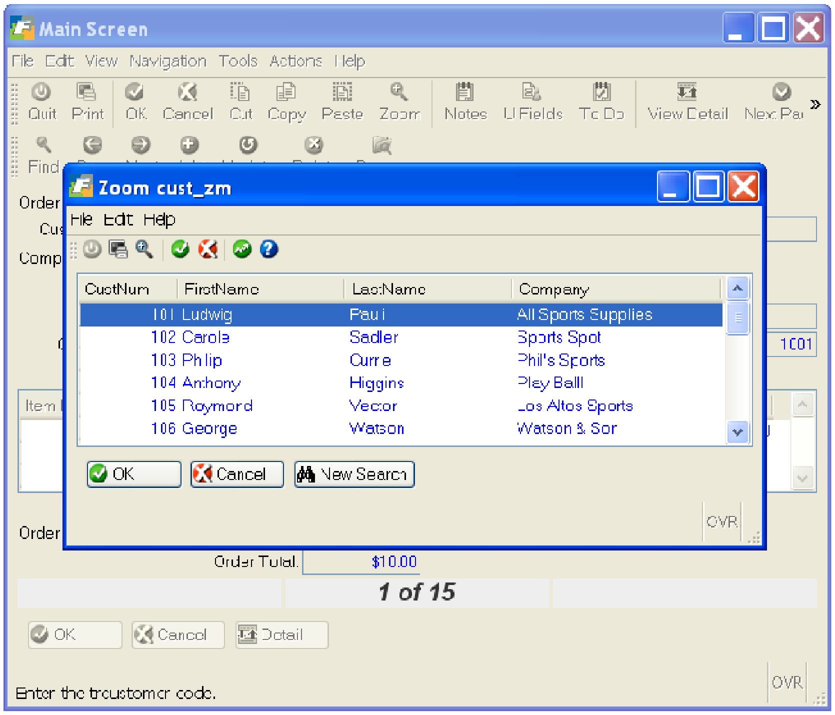
1. **Paint and define the zoom screen image.**

2. **Attach the zoom screen to a field on your main input screen**

Painting a Zoom Image

You define Zooms by using the VDT Form Designer to paint their image. Once you paint the image of the zoom screen, you must also specify from which field on your main input form the zoom screen will be activated. For example, the following application has a zoom screen attached to the Customer No. field.

Zoom screens are attached to input fields. Users initiate this Zoom from the Customer No. field.


Main Screen
File Edit View Navigation Tools Actions Help
Quit Print OK Cancel Cut Copy Paste Zoom Notes UFields To Do View Detail Next Pa
Find
Order
Cu
Comp
Item
Order
Zoom cust_zm
File Edit Help
OK Cancel New Search
OVR
OK Cancel Detail
OVR

CustNum	FirstName	LastName	Company
101	Ludwig	Paul	All Sports Supplies
102	Carole	Sadler	Sports Spot
103	Philip	Currie	Phil's Sports
104	Anthony	Higgins	Play Ball!
105	Raymond	Vector	Los Altos Sports
106	George	Watson	Watson & Son

1001

Order Total $10.00

1 of 15

Enter the trcustomer code.

To define a Zoom:

1. **Select New from the File pull-down menu.**

2. **Specify a name for the zoom screen.**

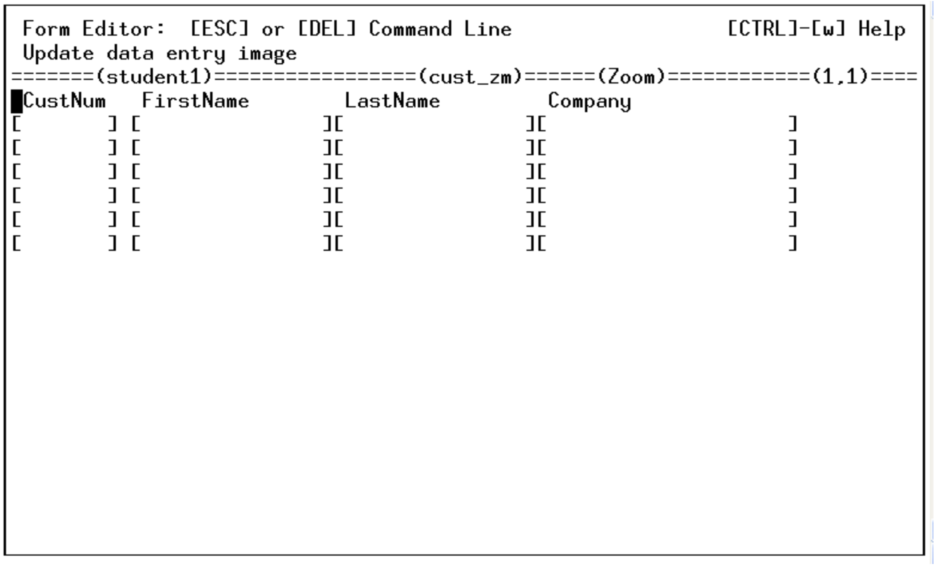
   It is a convention to give zoom windows a name that includes the letters **`zm`**, such as `cust_zm`, `stockzm`, etc

3. **Select zoom as the screen type.**

4. **Use the VDT Form Designer to paint and save the zoom image (see “Creating a Form Image” on page 38).**

   Because zoom screens usually contain several rows of duplicate field definitions, use mark, copy, and paste to speed your creation of the zoom image (see “**Marking, Copying, and Pasting**” on page 41)

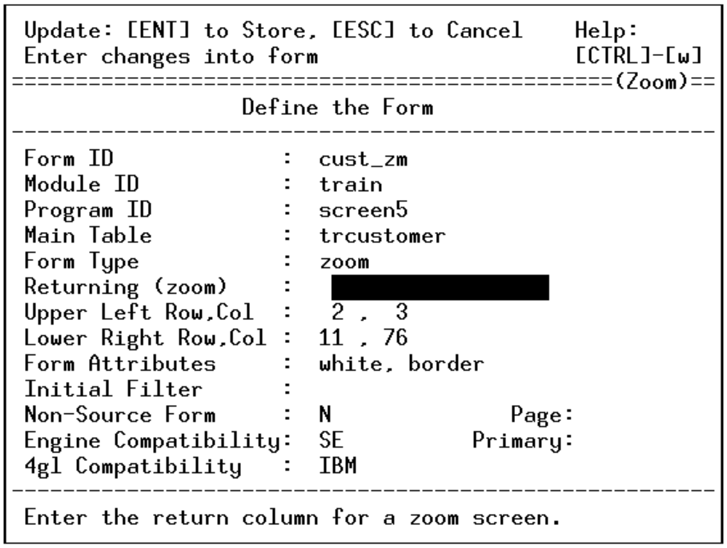
Zooms, such as this one, usually contain several rows of duplicate field definitions.



```
 Form Editor:  [ESC] or [DEL] Command Line                [CTRL]-[w] Help
 Update data entry image
=======(student1)=================(cust_zm)======(Zoom)===========(1,1)====
CustNum   FirstName        LastName         Company
[       ] [               ][               ][                    ]
[       ] [               ][               ][                    ]
[       ] [               ][               ][                    ]
[       ] [               ][               ][                    ]
[       ] [               ][               ][                    ]
[       ] [               ][               ][                    ]
```

After you paint and save your zoom image, you need to use the Form Defaults option on the Define pull-down menu. The Form Defaults option opens the Define the Form window. This window lets you specify from which field the zoom screen can be activated.

After you paint and save your zoom image, you need to use the Define the Form window to set your Zoom attributes



```
 Update: [ENT] to Store, [ESC] to Cancel    Help:
 Enter changes into form                    [CTRL]-[w]
===============================================(Zoom)==
                  Define the Form
-------------------------------------------------------
 Form ID             :  cust_zm
 Module ID           :  train
 Program ID          :  screen5
 Main Table          :  trcustomer
 Form Type           :  zoom
 Returning (zoom)    :
 Upper Left Row,Col  :   2 ,  3
 Lower Right Row,Col :  11 , 76
 Form Attributes     :  white, border
 Initial Filter      :
 Non-Source Form     :  N            Page:
 Engine Compatibility:  SE         Primary:
 4gl Compatibility   :  IBM
-------------------------------------------------------
 Enter the return column for a zoom screen.
```

Make sure to specify a value in the Returning (zoom) field. This field specifies where the returning value gets placed. In most cases, this is the field in which you attach the Zoom. If you are not sure of the field, press [CTRL]-[z] while to see a list of available fields

Attaching the Zoom Screen

You can attach a zoom screen to the main screen of your program using the VDT Form Designer.

To attach a zoom screen to an input field:

1. **Open the form that contains the field that you want to attach the zoom screen to.**

   In most cases, you attach zoom screens to header or header/detail screens, but this is not necessarily the case

2. **Highlight the field you want to attach the zoom screen to.**

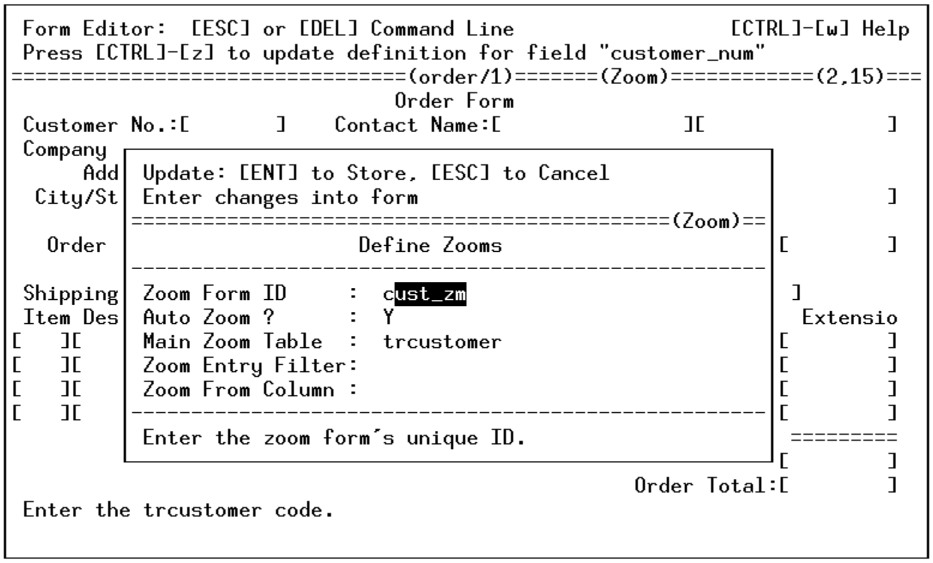
3. **Press [CTRL]-[z]**

   Note the irony here You activate a Form Painter Zoom in order to define a Zoom for your input program When you press [CTRL][z] a pop-up menu appears that contains all the items available for you to attach to the input field

4. **Select Zoom... from the list.**

   The Define Zooms window appears

The Define Zooms window lets you attach a zoom screen to an input field

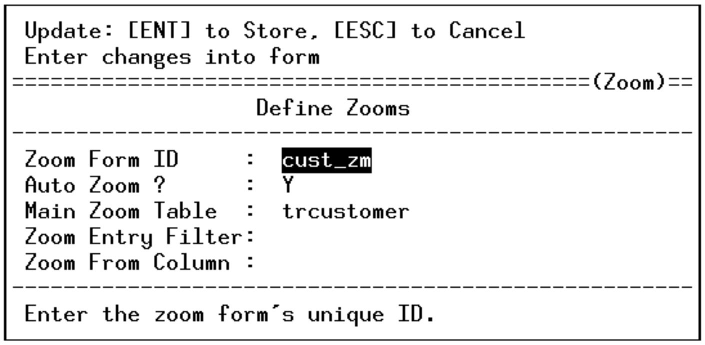


```
 Form Editor:  [ESC] or [DEL] Command Line                 [CTRL]-[w] Help
 Press [CTRL]-[z] to update definition for field "customer_num"
================================(order/1)=======(Zoom)============(2,15)===
                                Order Form
 Customer No.:[       ]    Contact Name:[               ][              ]
 Company |
     Add| Update: [ENT] to Store, [ESC] to Cancel
  City/St| Enter changes into form                                       ]
         |============================================(Zoom)==
   Order |                     Define Zooms                 | [        ]
         |--------------------------------------------------|
 Shipping| Zoom Form ID     :  cust_zm                      |  ]
 Item Des| Auto Zoom ?      :  Y                            |  Extensio
[    ][  | Main Zoom Table  :  trcustomer                   | [        ]
[    ][  | Zoom Entry Filter:                               | [        ]
[    ][  | Zoom From Column :                               | [        ]
[    ][  |--------------------------------------------------| [        ]
         | Enter the zoom form's unique ID.                 |  =========
         |__________________________________________________| [        ]
                                                  Order Total:[        ]
 Enter the trcustomer code.
```

5. **Fill in the Define Zooms window and press [ESC].**

   The Define Zooms window lets you specify how you want the zoom screen to be attached.

Use the Define Zooms window to specify how you want the Zoom to be attached.



```
Update: [ENT] to Store, [ESC] to Cancel
Enter changes into form
============================================(Zoom)==
                   Define Zooms
----------------------------------------------------
Zoom Form ID     :  cust_zm
Auto Zoom ?      :  Y
Main Zoom Table  :  trcustomer
Zoom Entry Filter:
Zoom From Column :
----------------------------------------------------
Enter the zoom form´s unique ID.
```

The Define Zooms window contains several fields. Perhaps the Zoom Form ID field is most important. In this field, you place the name of your Zoom screen. You should make sure that the Main Zoom Table field contains the correct value. If you want to add AutoZoom capability, specify Y in the AutoZoom field.

The Zoom Entry Filter field lets you assign a selection filter to the Zoom. The last field, Zoom From Column, lets you specify a table and column name for the Zoom if they differ from the column on the main screen.

Section Summary

- A Zoom is a data validation feature that shows the user a list of valid values for an input field. Zooms are invoked by pressing [CTRL]-[z].

- You define Zooms by using the VDT Form Designer to paint their image. Zooms are created from zoom screen types. Once you complete painting a Zoom, you can attach it to a field on your input program.

- To attach a zoom to an input field, you must identify which field the Zoom applies to. You can set all the Zoom attributes in the Define Zooms window.

Exercise 4A

Objective: To add a `credit` field to the `i_cust.4gs` program.

Start the VDT Form Designer

1. **Move to $HOME/labs/ aw. 4gm/i_cust. 4gs.**

2. **Start the VDT Form Designer.**

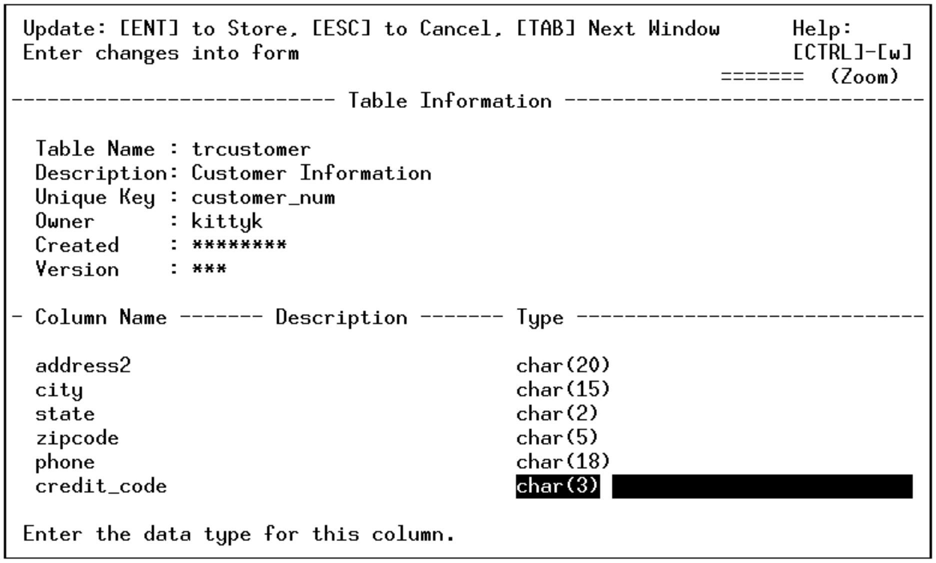
Add the credit_code Column to the customer Table

3. **Select Database from the File pull-down menu.**

   The Table Information window appears.

4. **Select Find from the ring menu, enter trcustomer in the Table Name field, and press [ESC].**

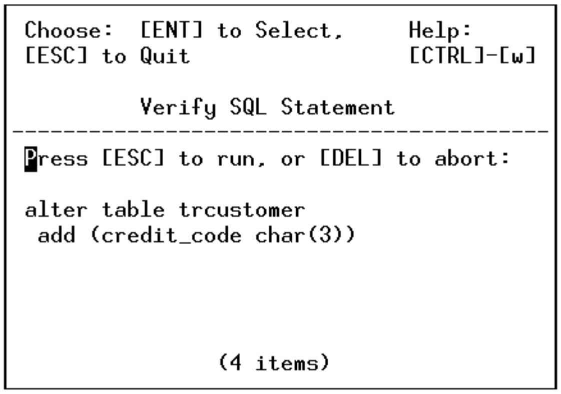
5. **Select Update and add a column named `credit_code` to the `customer` table:**



```
Update: [ENT] to Store, [ESC] to Cancel, [TAB] Next Window      Help:
Enter changes into form                                         [CTRL]-[w]
                                                         =======  (Zoom)
--------------------------- Table Information ------------------------------

  Table Name : trcustomer
  Description: Customer Information
  Unique Key : customer_num
  Owner      : kittyk
  Created    : ********
  Version    : ***

- Column Name ------- Description ------- Type -----------------------------

  address2                                char(20)
  city                                    char(15)
  state                                   char(2)
  zipcode                                 char(5)
  phone                                   char(18)
  credit_code                             char(3)

 Enter the data type for this column.
```

6. **Press [ESC].**

   A Verify SQL Statement box appears.



```
Choose:  [ENT] to Select,      Help:
[ESC] to Quit                  [CTRL]-[w]

          Verify SQL Statement
-----------------------------------------
Press [ESC] to run, or [DEL] to abort:

alter table trcustomer
 add (credit_code char(3))




                (4 items)
```

7. **Press [ESC] again to run the alter table SQL statement.**

8. **Select Quit to return to the VDT Form Desinger.**

9. **Exit the VDT Form Designer by choosing Exit from the File pull-down.**

Update the Genero Schema File

1. **Move to the schema directory**

   ```
   cd $fg/data
   ```

2. **Remake the Genero schema file. Make sure to use your database name.**

   ```
   sch.sh student1
   ```

Update the Central Database Changes Directory

1. Move to the central database changes directory

   cd $fg/data/sql.4gc

2. **Copy the local program dbadmin.sql to $fg/data/sql.4gc**

   cp $HOME/labs/aw.4gm/i_cust.4gs/dbadmin.sql   trcustomer.sql

Add a Credit Code Field to Your Screen

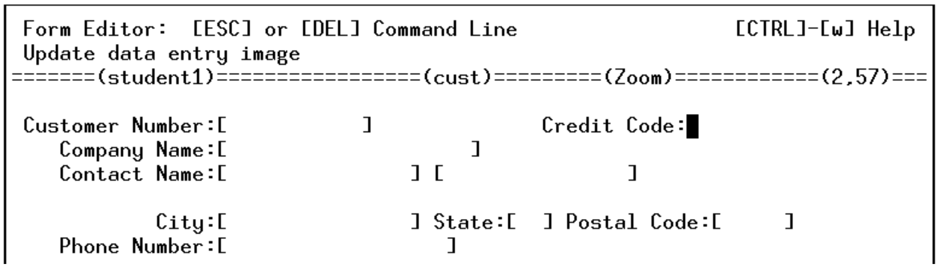
1. **Move to $HOME/labs/aw.4gm/i_cust.4gs**

2. **Start the VDT Form Designer**

3. **Select Open from the File pull-down menu.**

The VDT Form Designer opens your **`cust.per`** file. If you have additional form specification (*.per) files in this directory, you have to select **`cust`** from a list.

4. **Add a Credit Code field label in the upper half of your screen.**

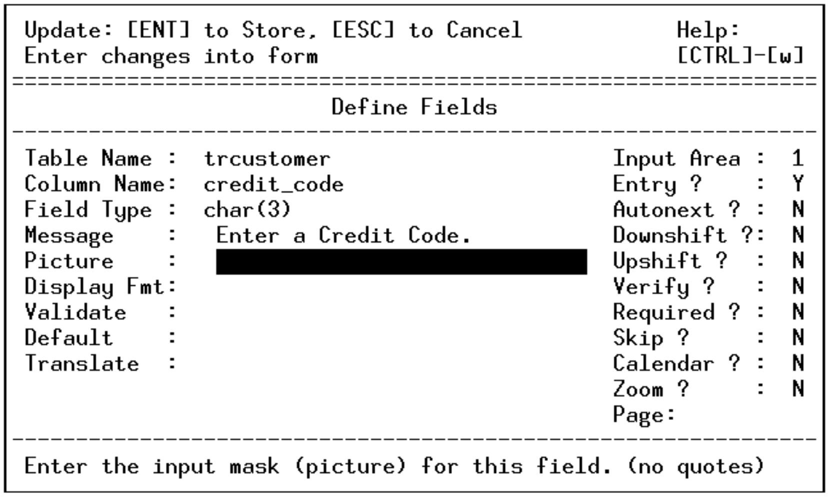


```
Form Editor:  [ESC] or [DEL] Command Line                 [CTRL]-[w] Help
Update data entry image
=======(student1)================(cust)=========(Zoom)============(2,57)===

Customer Number:[            ]              Credit Code:█
   Company Name:[                    ]
   Contact Name:[              ] [               ]

           City:[              ] State:[  ] Postal Code:[     ]
   Phone Number:[                 ]
```

5. **Define the Credit Code field by pressing a left bracket [after the field.**

The Define Fields window appears.

6. **Define the Credit Code field using the values shown below, then press [ESC] to save the definition.**



```
Update: [ENT] to Store, [ESC] to Cancel            Help:
Enter changes into form                            [CTRL]-[w]
===============================================================
                         Define Fields
---------------------------------------------------------------
Table Name :  trcustomer                           Input Area :  1
Column Name:  credit_code                          Entry ?    :  Y
Field Type :  char(3)                              Autonext ? :  N
Message    :   Enter a Credit Code.                Downshift ?:  N
Picture    :   █████████████████████████████       Upshift ?  :  N
Display Fmt:                                       Verify ?   :  N
Validate   :                                       Required ? :  N
Default    :                                       Skip ?     :  N
Translate  :                                       Calendar ? :  N
                                                   Zoom ?     :  N
                                                   Page:
---------------------------------------------------------------
Enter the input mask (picture) for this field. (no quotes)
```

Save, Generate, and Compile

1. **Select Save Form from the File pull-down menu.**

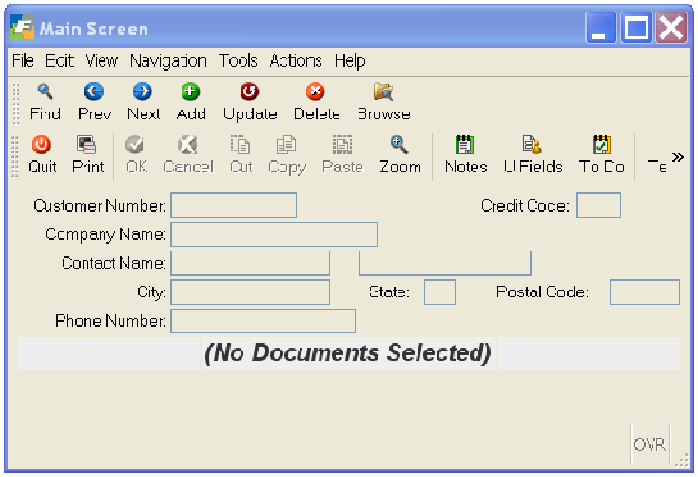
2. **Select Generate 4G L from the Run pull-down menu.**

3. **Select Compile 4G L from the Run pull-down menu.**

Run Your Customer Entry Program

1. **Select Run 4GL Program from the Run pull-down menu.**

2. **Use Find to select an existing customer and add a credit code for that customer.**


Main Screen
File Edit View Navigation Tools Actions Help
Find Prev Next Add Update Delete Browse
Quit Print OK Cancel Cut Copy Paste Zoom Notes U Fields To Do
Customer Number:
Credit Code:
Company Name:
Contact Name:
City:
State:
Postal Code:
Phone Number:
(No Documents Selected)
OVR

3. **When finished, quit the Customer Entry program and the VDT Form Designer.**

Exercise 4B

Objective: To create a zoom screen so users can select from a reference list of credit codes.

Create a Zoom Screen

1. **Start the VDT Form Designer.**

1. **Select New from the File pull-down menu.**

   The Define a New Form box appears.

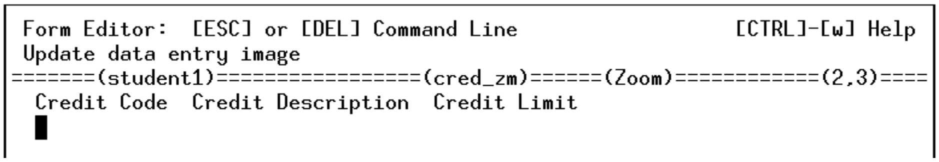
2. **Name the new form cred_zm.**

   The Select a Screen Type box appears.

3. **Use the down arrow to scroll down the screen type list and select zoom as the screen type.**

Create the Column Headings

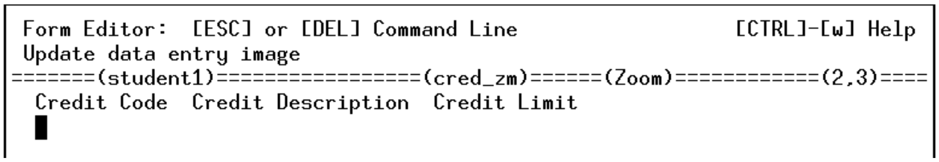
1. **Create the column headings for the Zoom.**



```
 Form Editor:  [ESC] or [DEL] Command Line                [CTRL]-[w] Help
 Update data entry image
=======(student1)================(cred_zm)======(Zoom)===========(2,3)====
  Credit Code  Credit Description  Credit Limit
  █
```

A zoom screen displays data in a row-by-row format.

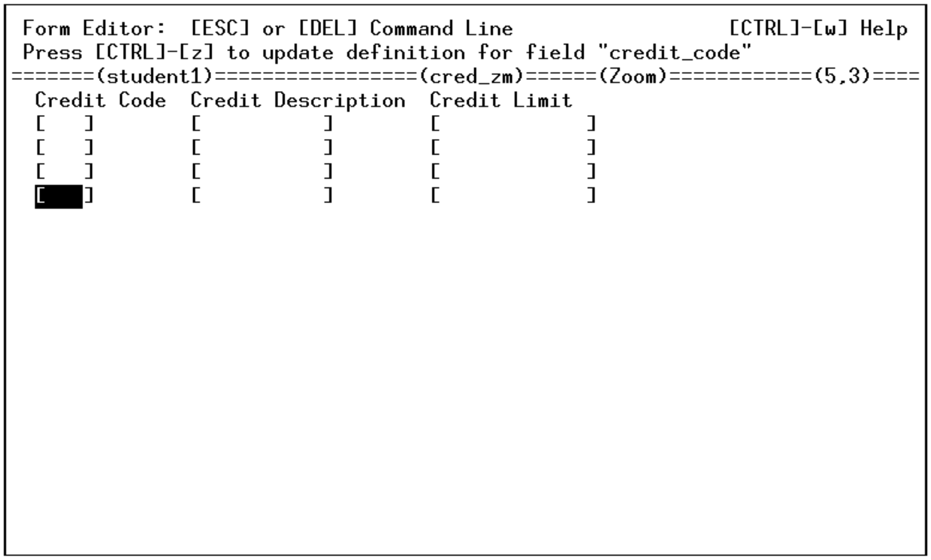
4. **Add field definitions using the columns in your trcredit table. (credit_code, credit_desc, and credit_amt)**



```
 Form Editor:  [ESC] or [DEL] Command Line                [CTRL]-[w] Help
 Update data entry image
=======(student1)================(cred_zm)======(Zoom)===========(2,3)====
  Credit Code  Credit Description  Credit Limit
  █
```

5. **Use the Mark, Copy, and Paste options to add three more rows of field definitions, see “Marking, Copying, and Pasting” on page 41.**

6. **Your finished zoom screen should look as follows:**



```
 Form Editor:  [ESC] or [DEL] Command Line                  [CTRL]-[w] Help
 Press [CTRL]-[z] to update definition for field "credit_code"
=======(student1)=================(cred_zm)======(Zoom)============(5,3)====
  Credit Code  Credit Description  Credit Limit
  [   ]        [          ]        [            ]
  [   ]        [          ]        [            ]
  [   ]        [          ]        [            ]
  [   ]        [          ]        [            ]
```

Specify Form Defaults

1. **Select Form Defaults from the Define pull-down menu.**

   The Form Defaults window appears.

7. **Enter trcredit in the Main Table field.**

   Zooms typically return values to the field from which they were invoked. Since you will be Zooming from the Credit Code field on your Customer Entry program, you must specify from which column the data will be supplied.

8. **Add credit_code in the Returning (zoom) field.**

   You can bypass the other fields on the window.

9. **Select Save Form from the File pull-down menu.**

10. **Select Generate 4GL from the Run pull-down menu.**

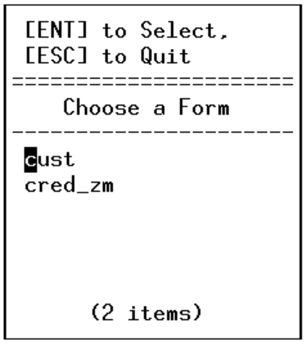
    The Generate 4g1: Enter Selection box appears.

11. **Select `cred_zm`.**

Attach cred_zm to the Credit Code Field

Now you must attach `cred_zm` to the Credit Code field that you credit on the Customer Entry program.

1. **Select Open from the File pull-down menu and open the file that corresponds to your Customer Entry program (cust).**



```
[ENT] to Select,
[ESC] to Quit
======================
   Choose a Form
----------------------
cust
cred_zm



     (2 items)
```

12. **Place your cursor in the Credit Code field and press [CTRL]-[z].**

    The Define Field pop-up menu appears.

13. **Select Zoom... from the Define Field pop-up menu.**

    The Define Zooms window appears.

14. **Enter `cred_zm` in the Zoom ID field.**

15. **Press [ENTER] in the Auto Zoom? field and enter tr`credit` in the Main Zoom Table field.**

16. **Specify `credit_code` in the Zoom From Column field and press [ESC] to save your zoom definition.**

17. **Place your cursor in the Credit Code field and press [CTRL]-[z].**

    The Define Field pop-up menu appears.

18. **Select Field.. from the Define Field pop-up menu.**

    The Define Fields window appears.

19. **Specify ‘Y’ in the Zoom? column.**

    This will cause the magnifying glass icon to display at runtime.

Save, Generate, and Compile

1. **Use the Save Form option under the File pull-down menu.**

20. **Select Generate 4GL from the Run pull-down menu.**

    The Generate 4GL: Enter Selection box appears.

21. **Select All Forms from this box.**

    The Local Forms Only box appears.

22. **Select YES.**

    As the VDT Application Code Generator runs, it builds code for both your zoom screen and your Customer Entry screen.

23. **Select Compile 4GL from the Run pull-down menu.**

Run Your Customer Entry Program

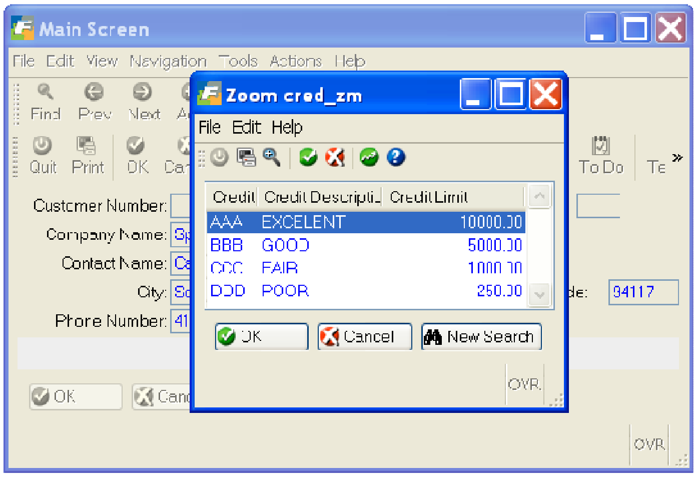
1. **Select Run 4G L Program from the Run pull-down menu.**

   The Customer Entry program starts.

24. **Use Find to select an existing customer and select Update.**

25. **From the Credit Code field, press [CTRL]-[z] and press [ENTER].**

    The Credit Information Zoom appears.



Credit	Credit Descripti	Credit Limit
AAA	EXCELENT	10000.00
BBB	GOOD	5000.00
CCC	FAIR	1000.00
DDD	POOR	250.00

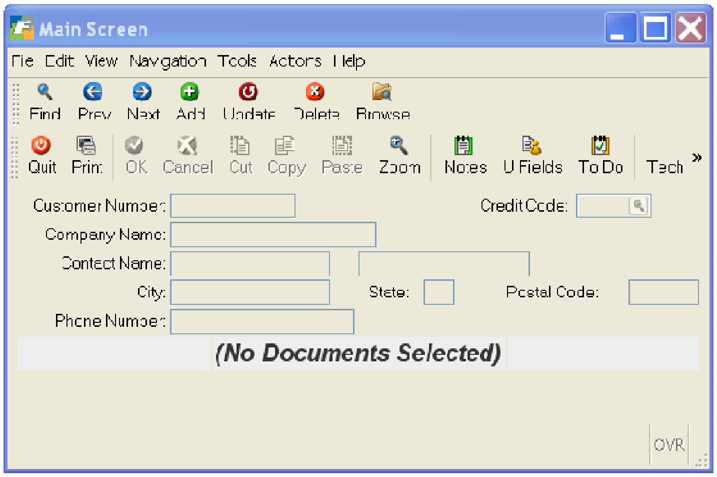
26. **Use the `cred_zm` a few times. When finished, quit out of Customer Entry and the VDT Form Designer.**

The Magnifying Glass (Zoom Indicator) for the Credit Code Field

1. **When you indicate ‘Y’ on the ‘Zoom?’ prompt in the ‘Define Fields’ window, it tells the painter to add the following highlighted lines to the .per file when it is saved:**

```
     Phone Number: [A7                ]
}
END -- GRID
END -- VBOX
END -- LAYOUT

TABLES
    trcustomer

ATTRIBUTES
A0 = trcustomer.customer_num,
     comments = " Enter a customer number.";
buttonedit A8 = trcustomer.credit_code,
      image="gn_zoomf.png", action=ac_zoom,
      comments = " Enter the Credit Code.";
A1 = trcustomer.company, comments = "";
A2 = trcustomer.fname, comments = "";
A3 = trcustomer.lname, comments = "";
A4 = trcustomer.city, comments = "";
```

2. When you run the program, the Credit Code column is displayed with a magnifying glass to the right of the data.


Main Screen
File Edit View Navigation Tools Actions Help
Find Prev Next Add Update Delete Browse
Quit Print OK Cancel Cut Copy Paste Zoom Notes U Fields To Do Tech »
Customer Number:
Credit Code:
Company Name:
Contact Name:
City:
State:
Postal Code:
Phone Number:
(No Documents Selected)
OVR

Chapter 5

Creating Lookups

Main topics:

- Lookup Overview
- Attaching a Lookup to a Field

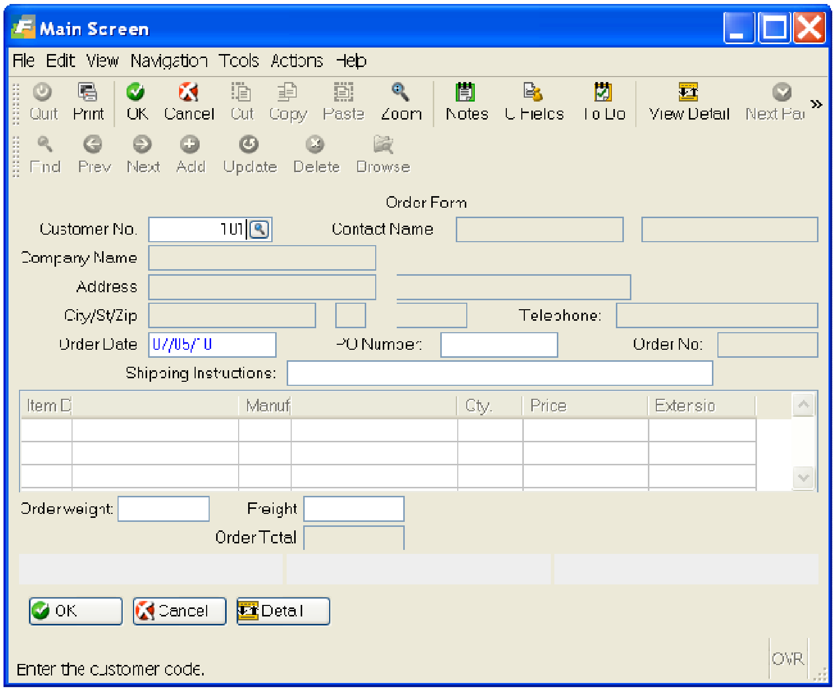
Lookup Overview

A Lookup performs a cross-check between two tables. You provide the lookup with a key value. The generator builds logic to open a cursor and fetch the key value from a reference table. If the key value does not exist in the reference table, an error is returned and the user is placed back in the Lookup field.

Lookups can also return data from the reference table keyed by the value you pass it. For example, if you pass a Lookup the customer number value, it can return a valid customer number, company name, owner name, street address, and other customer information:

A Lookup validates data and returns related data

In this example, a Lookup is defined on the Customer No. field.

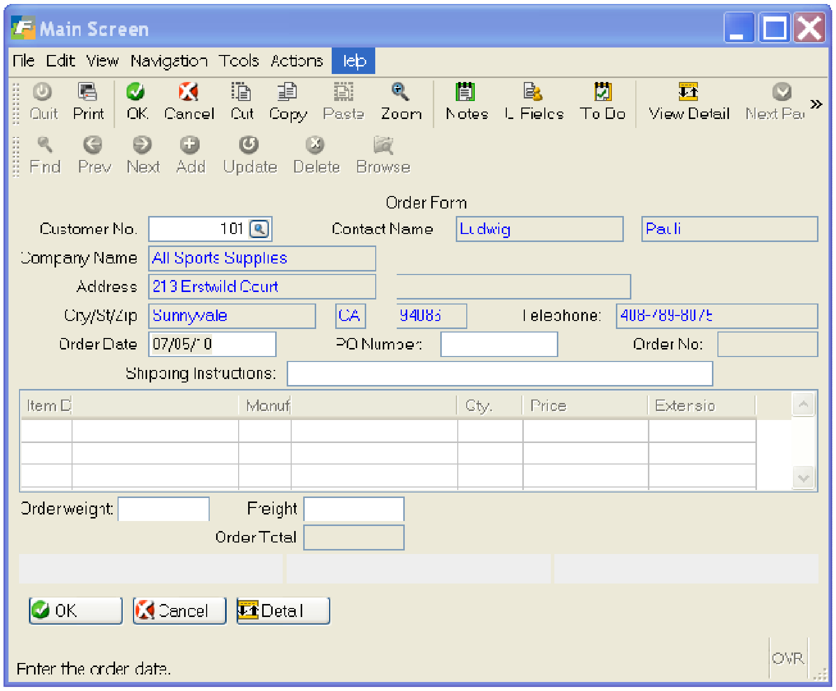


Customer No. 101

Order Date 07/05/10

Enter the customer code.

When the user enters a customer number, data relating to that number fills in the adjacent fields.


Main Screen
File Edit View Navigation Tools Actions Help
Quit Print OK Cancel Cut Copy Paste Zoom Notes L Fields To Do View Detail Next Pa
Find Prev Next Add Update Delete Browse
Item ID Manuf Qty. Price Extensio
OK Cancel Detail
OVR

Order Form

Customer No. 101 Contact Name Ludwig Pauli
Company Name All Sports Supplies
Address 213 Erstwild Court
City/St/Zip Sunnyvale CA 94086 Telephone: 408-789-8075
Order Date 07/05/10 PO Number: Order No:
Shipping Instructions:
Orderweight Freight
Order Total

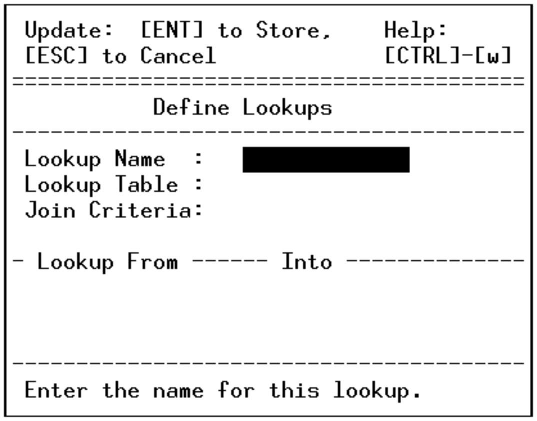
Enter the order date.

Attaching a Lookup to a Field

Like Zooms, you attach Lookups to input fields. Before you create a Lookup, you must know which field you want to attach it to and which fields you want to return values to.

Lookups are defined with the Define Lookups window. This window lets you specify the Lookup name, table, and join criteria. You also specify which fields on your main form the Lookup should fill.

The Define Lookups window lets you specify the Lookup name, table, and join criteria.



```
Update:  [ENT] to Store,     Help:
[ESC] to Cancel              [CTRL]-[w]
=========================================
           Define Lookups
-----------------------------------------
Lookup Name  :
Lookup Table :
Join Criteria:

- Lookup From ------ Into -------------



-----------------------------------------
Enter the name for this lookup.
```

The Lookup Name field holds the name of the Lookup. Uniquely naming Lookups lets you define multiple Lookups on the same field.

The Lookup Table field holds the name of the *looked up* table. In other words, this is the table from which values are being returned.

The Join Criteria field lets you specify the *where* clause of the join statement: you are specifying where the returned value is being put. The Join Criteria field uses the following syntax:

```
Table_name.column_name = $field_name
```

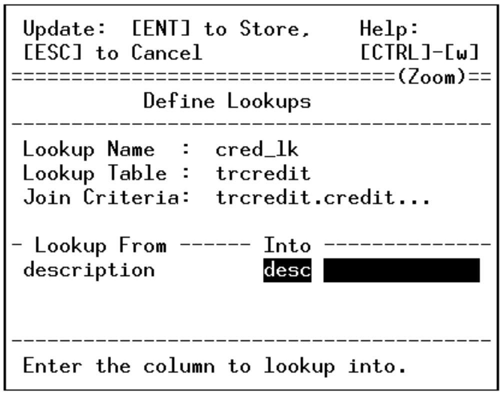
Where *table_name* and *column_name* represent the looked up table and *field_name* represents the column where the value gets returned.

For example, the following join criteria instructs the Lookup to search the `customer_num` column in the `customer` table and verify that the value in the `customer_num` field exists:

```
customer.customer_num = $customer_num
```

The Lookup From and Into fields are optional. These fields let you specify the join criteria when the column and field names differ. For instance, if the column name is `description` and the field name is `desc`, you could define the Lookup as follows:

This example shows how the Lookup From and Into fields are used. You only need to use these fields when the column and field names do not match.



```
 Update:  [ENT] to Store,    Help:
 [ESC] to Cancel             [CTRL]-[w]
==============================(Zoom)==
           Define Lookups
--------------------------------------
 Lookup Name  :  cred_lk
 Lookup Table :  trcredit
 Join Criteria:  trcredit.credit...

- Lookup From ------ Into -------------
 description         desc


--------------------------------------
 Enter the column to lookup into.
```

If the fields and columns have the same name, you do not need to add them to the Lookup From and Into fields. The Screen Generator builds this logic when the field names and column names match.

To define a Lookup:

1. **Using the VDT Form Designer, highlight the field that you want to attach a Lookup to.**

2. **Press [CTRL]-[z] to display the Define Field menu.**

3. Select Lookups... from the Define Field menu.

The Define Lookups window appears. You can also access the Define Lookups window from the. Define pull-down menu by choosing the Lookups... option**.**

4. Fill in the Define Lookups window and press [ESC].

When a user enters an invalid value into a field that has a Lookup attached, an error occurs. The user is not able to leave that field until a valid value has been entered.

Section Summary

- Lookups are placed on fields in a data-entry screen to evaluate the data entered by a user.
- Lookups check a key value against a reference data table. If the key value exists, the Lookup allows the user to continue. If the Lookup doesn't exist, an error occurs and the user is placed back in the Lookup field.
- Another purpose of a Lookup is to return data keyed by the Lookup value. A value entered by a user can cause a cross-referenced value to be looked up in the reference table and displayed on the input form.

Exercise 5A

Objective: To add a lookup on the Credit Code field. A lookup prevents users from entering invalid data.

Check the Credit Code Value

1. **Start the VDT Form Designer and select Run 4GL Program from the Run pull-down menu.**

2. **From your Customer Entry program, use Find to select an existing customer.**

3. **Select Update and enter TTT in the Credit Code field.**

   Recall that TTT is not a record in the credit table. You only created four records in that table (AAA, BBB, CCC, and DDD). Despite this fact, however, the program still accepts TTT, a completely invalid value. You can use lookups to verify data in a field.

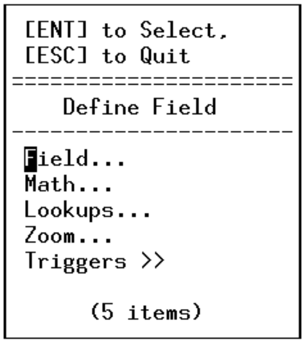
4. **Quit the Customer Entry program.**

5. **From the VDT Form Designer, open the Customer Entry file (cust).**

Define the Lookup

1. **From the VDT Form Designer, place your cursor in the Credit Code field and press [CTRL]-[z].**

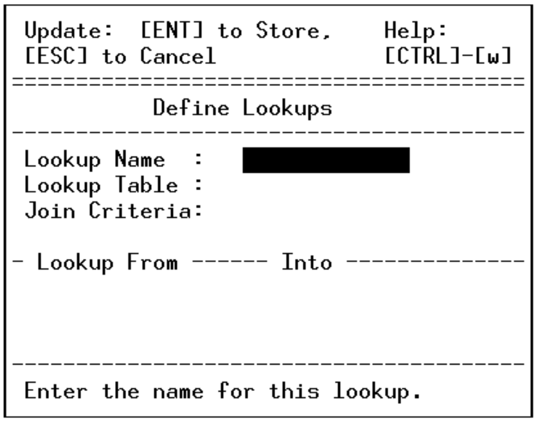
   The Define Field pop-up menu appears.



```
[ENT] to Select,
[ESC] to Quit
======================
    Define Field
----------------------
Field...
Math...
Lookups...
Zoom...
Triggers >>

     (5 items)
```

2. **Select Lookups... from the Define Field pop-up menu.**

   The Define Lookups window appears.

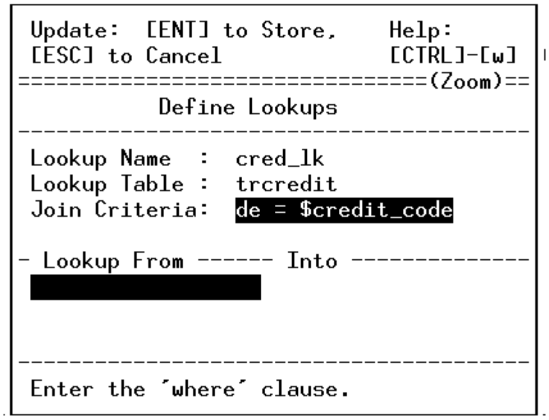


```
 Update:  [ENT] to Store,    Help:
 [ESC] to Cancel             [CTRL]-[w]
========================================
           Define Lookups
----------------------------------------
 Lookup Name  :
 Lookup Table :
 Join Criteria:

- Lookup From ------ Into -------------



----------------------------------------
 Enter the name for this lookup.
```

3. **Enter `cred_lk` in the Lookup Name field.**

4. **Enter trcredit in the Lookup Table field.**

5. **Enter `trcredit.credit_code = $credit_code` in the Join Criteria field.**



```
 Update:  [ENT] to Store,    Help:
 [ESC] to Cancel             [CTRL]-[w]
==============================(Zoom)==
           Define Lookups
----------------------------------------
 Lookup Name  :  cred_lk
 Lookup Table :  trcredit
 Join Criteria:  de = $credit_code

- Lookup From ------ Into -------------



----------------------------------------
 Enter the 'where' clause.
```

6. **Press [ESC] to save your lookup.**

Save, Generate, and Compile

1. **Use the Save Form option under the File pull-down menu.**

2. **Select Generate 4GL from the Run pull-down menu.**

   The Generate 4gl: Enter Selection box appears.

3. **Select All Forms from this box.**

   The Local Forms Only box appears.

4. **Select YES**

5. **Select Compile 4GL from the Run pull-down menu.**

Run Your Customer Entry Program

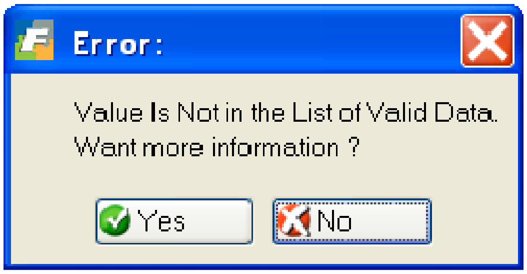
1. **Select Run 4GL Program from the Run pull-down menu.**

   The Customer Entry program starts.

2. **Find a customer and select Update.**

3. **Enter TTT in the Credit Code field.**

   An error message appears:



Error:

Value Is Not in the List of Valid Data.
Want more information ?

4. **Click ✓ Yes to return to the Credit Code field and enter a valid value (AAA).**

   This time the value is accepted and the cursor moves to the next field.

5. **Press [ENTER] to save. Then click ⏻ to return to the VDT Form Designer.**

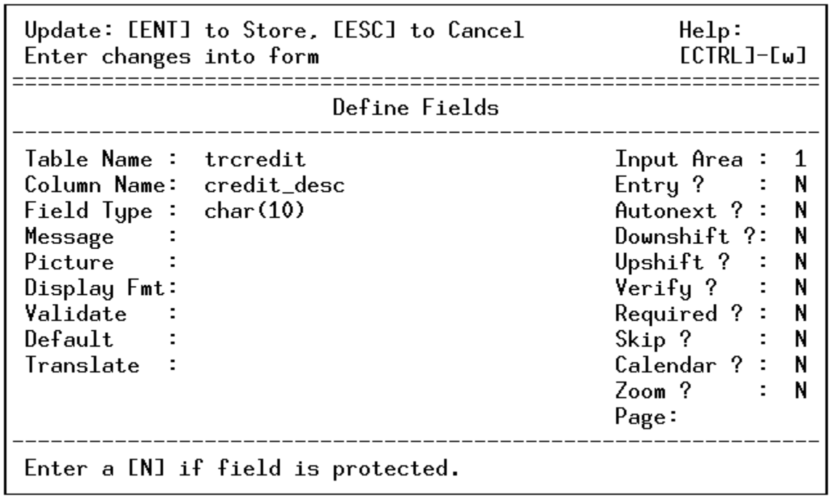
Exercise 5B

Objective: To create a Credit Desc field that is linked to the Credit Code field. When the user specifies a Credit Code, the Credit Desc field will get filled automatically.

1. **Create a new field on the Customer Entry form called Credit Desc.**

   In other words, create a field label and press [Ctrl-z] to define it.

   On the Define Fields window, specify N in the Entry? field.

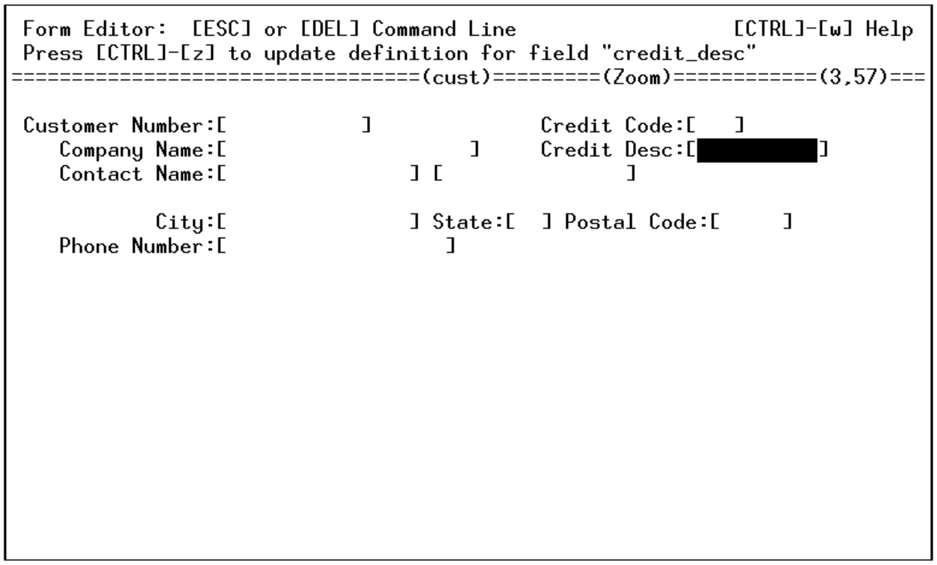


```
Update: [ENT] to Store, [ESC] to Cancel           Help:
Enter changes into form                           [CTRL]-[w]
===============================================================
                          Define Fields
---------------------------------------------------------------
Table Name :  trcredit                       Input Area :  1
Column Name:  credit_desc                    Entry ?    :  N
Field Type :  char(10)                       Autonext ? :  N
Message    :                                 Downshift ?:  N
Picture    :                                 Upshift ?  :  N
Display Fmt:                                 Verify ?   :  N
Validate   :                                 Required ? :  N
Default    :                                 Skip ?     :  N
Translate  :                                 Calendar ? :  N
                                             Zoom ?     :  N
                                             Page:
---------------------------------------------------------------
Enter a [N] if field is protected.
```

   When Entry? is N, the user cannot enter / update the field.

2. **Press [ESC] to save the field definition.**

   You should now have the following fields on you Customer Entry program:



```
 Form Editor:  [ESC] or [DEL] Command Line                 [CTRL]-[w] Help
 Press [CTRL]-[z] to update definition for field "credit_desc"
==================================(cust)=========(Zoom)============(3,57)===

 Customer Number:[           ]              Credit Code:[    ]
    Company Name:[                    ]     Credit Desc:[          ]
    Contact Name:[               ] [               ]

            City:[               ] State:[  ] Postal Code:[     ]
    Phone Number:[                  ]
```

Save, Generate, and Compile

1. **Use the Save Form option under the File pull-down menu.**

2. **Select Generate 4GL from the Run pull-down menu.**

   The Generate 4gl: Enter Selection box appears.

3. **Select All Forms from this box.**

   The Local Forms Only box appears.

4. **Select YES.**

5. **Select Compile 4GL from the Run pull-down menu.**

Run Your Customer Entry Program

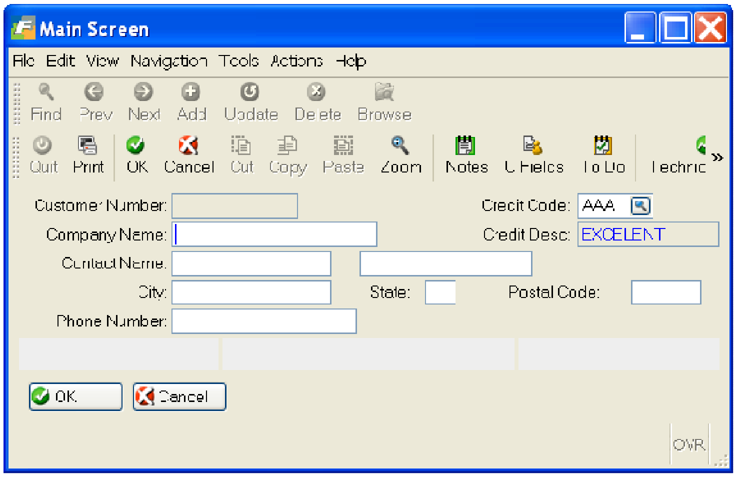
1. **Select Run 4GL Program from the Run pull-down menu.**

   The Customer Entry program starts.

2. **Select Add to create a new customer entry.**

3. **In the Credit Code field, enter BBB.**

   Notice how the Credit Desc field is filled automatically**.**


Main Screen
File Edit View Navigation Tools Actions Help
Find Prev Next Add Update Delete Browse
Quit Print OK Cancel Cut Copy Paste Zoom Notes U Fields To Do Technic
Customer Number:
Company Name:
Contact Name:
City:
State:
Postal Code:
Phone Number:
OK
Cancel
OVR

Credit Code: AAA

Credit Desc: EXCELENT

4. **Quit out of the Customer Entry program to return to the VDT Form Designer.**

Chapter 6

Input Areas and Specification Files

Main topics:

- Input Area Overview
- Creating Form Specification (*.per) Files

Input Areas Overview

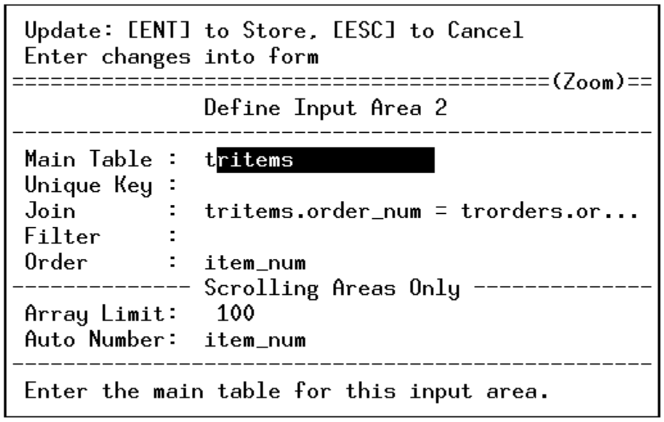
Input areas are where you specify characteristics about the header and/or detail portion of the form. The header portion is always assigned area 1 and the detail portion is assigned area 2.

```
Input Area 1 = Header
Input Area 2 = Detail
```

You set input area characteristics with the Define Input Areas window. You can use the Input Areas option under the Define pull-down menu to access this window.

You set input area characteristics with the Define Input Areas window.

This example shows values for the detail portion of a form



```
 Update: [ENT] to Store, [ESC] to Cancel
 Enter changes into form
========================================(Zoom)==
              Define Input Area 2
------------------------------------------------
 Main Table :  tritems
 Unique Key :
 Join       :  tritems.order_num = trorders.or...
 Filter     :
 Order      :  item_num
-------------- Scrolling Areas Only --------------
 Array Limit:   100
 Auto Number:  item_num
------------------------------------------------
 Enter the main table for this input area.
```

For a header/ detail screen, you must specify characteristics about the detail portion of the form in order for the form to work properly. For instance, you must specify the join that connects the header table to the detail table.

The most important field is the Join field. The Join field specifies how the header and detail tables are related. Use the following syntax to define the join between the two tables:

```
header_table.column = detail_table.column
```

Where *header_table.column* represents the name of the header column and *detail_table.column* represents the name of the detail column.

Another important field is the Unique Key field. This field specifies which columns uniquely define a row in the table.

If you do not specify the input area for the header, the VDT Form Designer puts in default values for you.

When you define the header input area, you cannot enter the Join field. This field is only for detail input areas.

Creating Form Specification (*.per) Files

Every time you save a form image with the VDT Form Designer, a form specification (*.per) file is created. It is helpful for you to become familiar with this file.

The VDT Application Code Generator uses form specification (*.per) files to produce all the 4GL source code necessary to create an input program. In general, form specification files contain the following sections:

Section	**Use**
SCHEMA	Specifies the database that the form is created and compiled against.
LAYOUT	Contains the image of the form. Each input field is identified by a field tag.
TABLES	Identifies the tables that are used by the form.
ATTRIBUTES	Ties each field tag (in the `SCREEN` section) with a column in the table. Fields can also be classified as *formonly*. *Formonly* fields are not associated with columns of any database. They are used to enter or show the values of program variables. This section also contains related characteristics of the field (e.g., comments, required logic, verification logic, and formatting instructions, button assignments).
INSTRUCTIONS	Specifies non-default field delimiters and defines screen arrays and records, such as the `s` record.
FXOOL	Contains specific instructions that are read by the VDT Application Code Generator. The VDT Application Code Generator builds the code logic based on what is specified in the section.

Once you become familiar with VDT CASE Tools, you will learn how to read form specification files. You will learn how to recognize what the VDT Form Designer creates in these files. A typical form specification file looks as follows:

```
SCHEMA student1

LAYOUT (TEXT=%"train.screen5.order")
VBOX nm_vbox_main (TAG="tg_vbox_main")
```

```
GRID
{

 Customer No.:[A0     ]    Contact Name:[A1             ][A2             ]
 Company Name:[A3                  ]
      Address:[A4                  ][A5                  ]
  City/St/Zip:[A6             ][A7] [A8   ] Telephone:[A9                ]

   Order Date:[AA      ]     PO Number:[AB        ]    Order No:[AC      ]

 Shipping Instructions: [AD                                      ]
}
END -- GRID
TABLE nm_table_header_detail (WIDTH=76, TAG="tg_table_header_detail")
{
 Item Description      Manufacturer             Qty.      Price   Extension
[AE ][AF             ][AG ][AH             ]  [AI  ][AJ        ][AK        ]
[AE ][AF             ][AG ][AH             ]  [AI  ][AJ        ][AK        ]
[AE ][AF             ][AG ][AH             ]  [AI  ][AJ        ][AK        ]
[AE ][AF             ][AG ][AH             ]  [AI  ][AJ        ][AK        ]
}
END -- TABLE
GRID
{

===========
                              Order weight:[AL     ]    Freight:[AM        ]
                                                    Order Total:[AN        ]
}
END -- GRID
END -- VBOX
END -- LAYOUT

TABLES
    trorders
    trcustomer
    tritems
    trstock
    trmanufact
ATTRIBUTES
buttonedit A0 = trorders.customer_num,
     image="gn_zoomf.png", action=ac_zoom,
     comments = " Enter the trcustomer code.";
A1 = trcustomer.fname, noentry, comments = "";
A2 = trcustomer.lname, noentry, comments = "";
A3 = trcustomer.company, noentry, comments = "";
A4 = trcustomer.address1, noentry, comments = "";
A5 = trcustomer.address2, noentry, comments = "";
A6 = trcustomer.city, noentry, comments = "";
```

```
A7 = trcustomer.state, noentry, comments = "";
A8 = trcustomer.zipcode, noentry, comments = "";
A9 = trcustomer.phone, noentry, comments = "";
AA = trorders.order_date, format = "mm/dd/yy",
     comments = " Enter the order date.";
AB = trorders.po_num,
     comments = " Enter the trcustomer's purchase order number.";
AC = trorders.order_num, noentry, comments = "";
AD = trorders.ship_instruct,
     comments = " Enter any special shipping instructions to show on the in-
voice.";
AL = trorders.ship_weight,
     comments = " Enter the total shipping weight for this order.";
AM = trorders.ship_charge,
     comments = " Enter the total shipping charge for this order.";
AN = formonly.t_price type money, noentry, comments = "";
buttonedit AE = tritems.stock_num,
     image="gn_zoomf.png", action=ac_zoom,
     comments = " Enter the trstock number for this line item.";
AF = trstock.description, noentry, comments = "";
buttonedit AG = tritems.manu_code,
     image="gn_zoomf.png", action=ac_zoom,
     comments = " Enter the trmanufacturers code for this trstock number.";
buttonedit AH = trmanufact.manu_name, noentry,
     image="gn_zoomf.png", action=ac_zoom, comments = "";
AI = tritems.quantity,
     comments = " Enter the number of units sold for this item.";
AJ = trstock.unit_price, noentry, comments = "";
AK = tritems.total_price, noentry, comments = "";

INSTRUCTIONS
screen record s_order (trorders.customer_num, trcustomer.fname,
    trcustomer.lname, trcustomer.company, trcustomer.address1,
    trcustomer.address2, trcustomer.city, trcustomer.state,
    trcustomer.zipcode, trcustomer.phone, trorders.order_date,
    trorders.po_num, trorders.order_num, trorders.ship_instruct,
    trorders.ship_weight, trorders.ship_charge, formonly.t_price)

screen record s_tritems[4] (tritems.stock_num, trstock.description,
    tritems.manu_code, trmanufact.manu_name, tritems.quantity,
    trstock.unit_price, tritems.total_price)

delimiters "  "


{
######################################################################
FXTOOL
######################################################################
defaults
    module         = train
    type           = header/detail
    init           = order_num > 100
    attributes     = white
    location       = 2, 3
    ifx_lang_ver   = IBM
    ifx_engine_ver = SE

input 1
    table     = trorders
    key       = order_num
    filter    = order_date > "12/31/80"
    order     = order_num
    math      = t_price = sum(total_price) + ship_charge
    lookup    = name=trcustomer, key=customer_num, table=trcustomer,
      filter=customer_num = $customer_num
    zoom      = key=customer_num, screen=cust_zm, table=trcustomer
    default   = order_date = today

input 2
    table     = tritems
    order     = item_num
    join      = tritems.order_num = trorders.order_num
    arr_max   = 100
    autonum   = item_num
    math      = total_price = quantity * unit_price
    lookup    = name=stock_num, key=stock_num, table=trstock,
      into=description, filter=stock_num = $stock_num
    lookup    = name=trstock_manu, key=manu_code, table=trstock,
      into=unit_price,
      filter=stock_num = $stock_num and manu_code = $manu_code
    lookup    = name=trmanufact, key=manu_code, table=trmanufact,
      into=manu_name, filter=manu_code = $manu_code
    zoom      = key=stock_num, screen=stockzm, table=trstock,
      noautozoom
    zoom      = key=manu_code, screen=stk_mnu, table=trstock,
      filter=trstock.stock_num = $stock_num
}
```

Section Summary

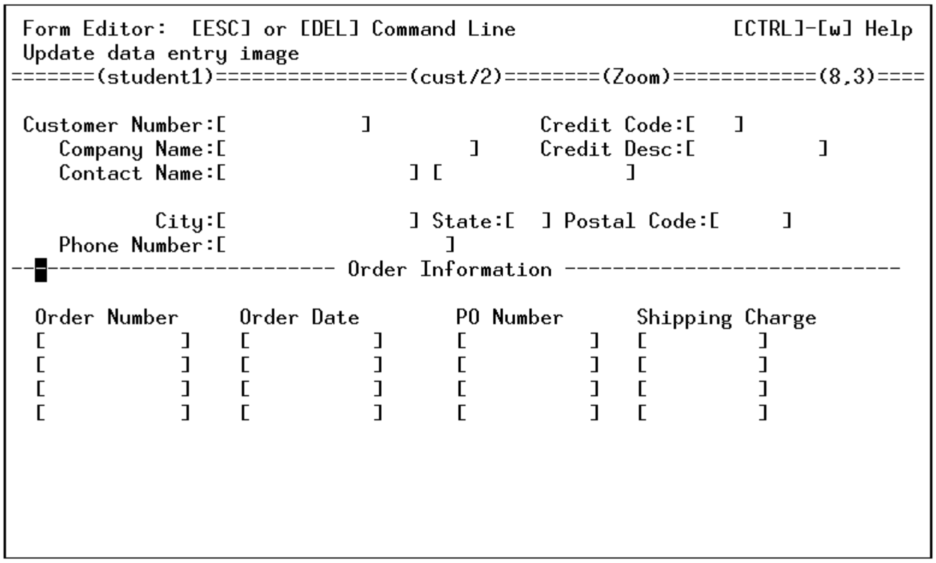
- All forms you create with the VDT Form Designer contain input areas. Input areas correspond to the header and/or detail section of a form. The most important attribute that you set is the table attribute. It specifies which table the header portion of the form writes to and which the detail portion of the form writes to.

- TheVDT Form Designer creates and Informix form specification (*.per) file. As you become familiar with the Firtrix *CASE* Tools, you will learn how to read and alter form specification files.

Exercise 6

Objective: To convert the Customer Entry program from a header screen to a header / detail screen. The detail portion will write to a detail table, which is the "many" table in a one-to-many table relationship.

The detail portion will show data from the `trorders` table. At the end of this exercise, your Customer Entry program will look as follows:



```
 Form Editor:  [ESC] or [DEL] Command Line                 [CTRL]-[w] Help
 Update data entry image
=======(student1)===============(cust/2)========(Zoom)============(8,3)====

 Customer Number:[           ]             Credit Code:[   ]
    Company Name:[                    ]    Credit Desc:[          ]
    Contact Name:[               ] [               ]

            City:[               ] State:[  ] Postal Code:[     ]
    Phone Number:[                ]
---------------------------- Order Information ----------------------------

  Order Number     Order Date        PO Number      Shipping Charge
  [           ]    [          ]      [          ]   [         ]
  [           ]    [          ]      [          ]   [         ]
  [           ]    [          ]      [          ]   [         ]
  [           ]    [          ]      [          ]   [         ]
```

Change the Screen Type to Header/Detail

This exercise assumes you are already running the VDT Form Desoigner with your `cust.per` form open. If this is not the case move to your program directory (`cd $HOME/labs/aw.4gm/i_cust.4gs`), start the Form Painter, and open `cust.per`.

1. **Select Form Defaults from the Define pull-down menu.**

   The Define the Form window appears. As you recall, this window specifies various characteristics about your form, including the screen type (which is set in the Form Type field).

2. **Change the Form Type field from header to header/detail.**

   This converts your form to a header/ detail screen.

3. **Change the Lower Right Row field to 20.**

   This makes the form big enough for your detail section.

4. **Press [ESC] to store your change and close the window.**

Add the Detail Section

Now add a detail section called Order Information to your Customer Entry program.

1. **Creating a detail section title:**

   -------------------Order Information -------------------

2. **Add the following field labels below the title:**

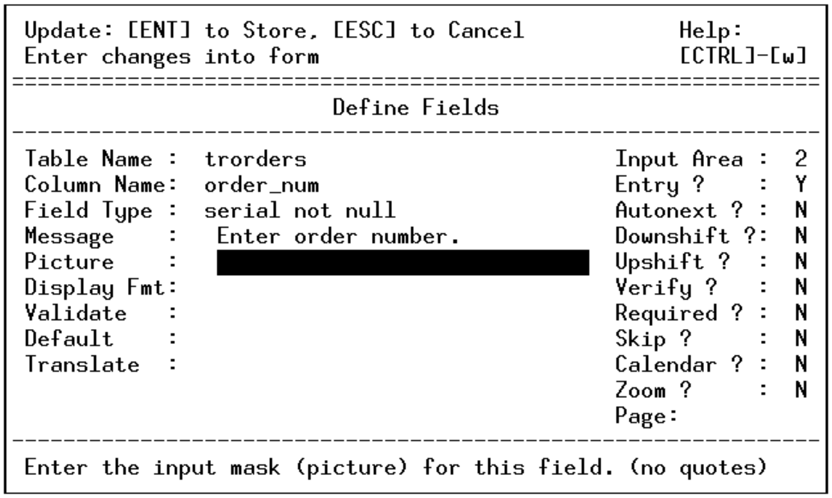
```
Order Number       Order Date          PO Number      Shipping Charge
```

3. **Place your cursor below the 0 in Order Number.**

4. **Press [Ctrl-z].**

   The Define Fields window appears. Fields in this detail section correspond to the **`orders`** table. Remember that a detail section is considered Input Area 2.

5. **Define the Order Number field using the following values.
   (Note the Table Name and Input Area fields):**



```
Update: [ENT] to Store, [ESC] to Cancel          Help:
Enter changes into form                          [CTRL]-[w]
===============================================================
                          Define Fields
---------------------------------------------------------------
Table Name :  trorders                       Input Area :  2
Column Name:  order_num                      Entry ?    :  Y
Field Type :  serial not null                Autonext ? :  N
Message    :   Enter order number.           Downshift ?:  N
Picture    :                                 Upshift ?  :  N
Display Fmt:                                 Verify ?   :  N
Validate   :                                 Required ? :  N
Default    :                                 Skip ?     :  N
Translate  :                                 Calendar ? :  N
                                             Zoom ?     :  N
                                             Page:
---------------------------------------------------------------
Enter the input mask (picture) for this field. (no quotes)
```

6. **Press [ESC] to store these values and define the field.**

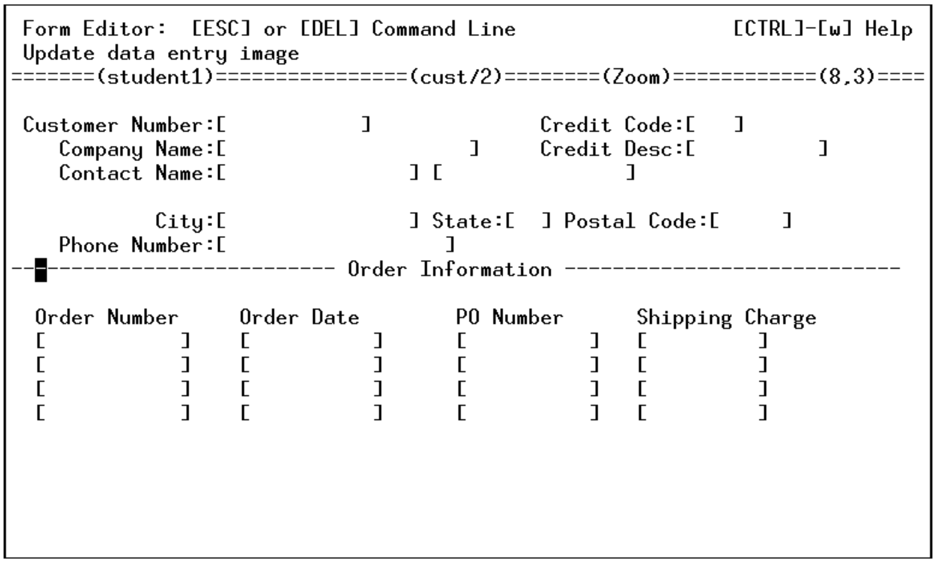
7. **Repeat these steps until you've created a complete row of detail fields.**

   Once you have a complete row, use the Mark, Copy, and Paste options to create three duplicate rows. As you recall, detail sections, much like zooms, display data in a row-by-row format.

   When you are finished you should have four detail lines with fields for the following columns:

```
trorders.order_num
trorders.order_date
trorders.po_num
trorders.ship_charge
```

Your screen should look as follows:



```
 Form Editor:  [ESC] or [DEL] Command Line                [CTRL]-[w] Help
 Update data entry image
=======(student1)================(cust/2)========(Zoom)============(8,3)====

 Customer Number:[           ]             Credit Code:[   ]
    Company Name:[                    ]    Credit Desc:[          ]
    Contact Name:[               ] [               ]

            City:[               ] State:[  ] Postal Code:[     ]
    Phone Number:[                  ]
--------------------------- Order Information ----------------------------

  Order Number     Order Date        PO Number      Shipping Charge
  [           ]    [          ]      [          ]   [         ]
  [           ]    [          ]      [          ]   [         ]
  [           ]    [          ]      [          ]   [         ]
  [           ]    [          ]      [          ]   [         ]
```

Define the Detail Input Area

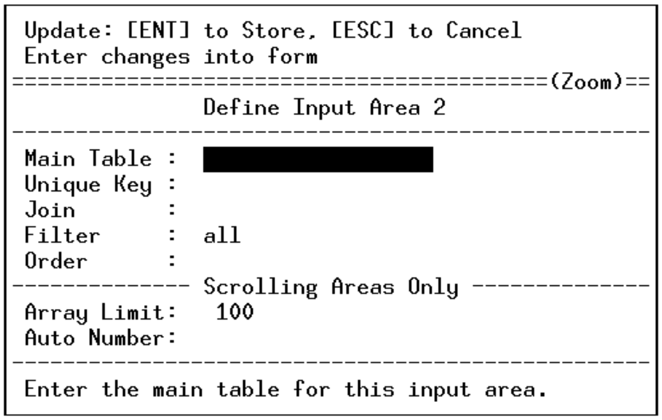
Once the image of the Customer Entry form's detail section is correct, you must define the Input Area.

1. **Select Input Areas from the Define pull-down menu.**

   The Input Area list box appears.

2. **Select Detail from the list box.**

   The Define Input Area 2 box appears.



```
 Update: [ENT] to Store, [ESC] to Cancel
 Enter changes into form
==========================================(Zoom)==
               Define Input Area 2
--------------------------------------------------
 Main Table :
 Unique Key :
 Join       :
 Filter     :  all
 Order      :
-------------- Scrolling Areas Only --------------
 Array Limit:   100
 Auto Number:
--------------------------------------------------
 Enter the main table for this input area.
```

3. **Specify trorders as the Main Table.**

   Based on this value, the Unique Key value is automatically filled with the **`order_num`** value.

4. **In the Join field, enter:**

   ```
   trcustomer.customer_num = trorders.customer_num
   ```

5. **For now, disregard the other fields and press [ESC].**

   The Define Input Area 2 window closes.

Save, Generate, and Compile

1. **Use the Save Form option under the File pull-down menu.**

2. **Select Generate 4GL from the Run pull-down menu.**

   The Generate 4gl: Enter Selection box appears.

3. **Select All Forms from this box.**

   The Local Forms Only box appears.

4. **Select YES.**

5. **Select Compile 4GL from the Run pull-down menu.**

Run Your Customer Entry Program

6. **Select Run 4GL Program from the Run pull-down menu.**

   The Customer Entry program starts.

7. **2. Use Find to select all existing customers.**

8. **Use Nxt and Prv to scroll through the records.**

   As you scroll, notice how values from the orders table populate the detail section of the program. As you can see, some customers have made orders while others have not.

9. **Click the** **button to move to the Detail section. When you are through, remain in your Customer Entry program. The next exercise starts from here.**

Chapter 7

Working with the User Control Libraries

Main topics:

- Main topics:
- User Control Library Overview
- Creating a To-Do List
- Adding Freefrom Notes
- Entering Error Messages
- Adding Help Text
- Defining Navigation Events
- Mapping Hot Keys to Navigation Events

User Control Library Overview

The User Control Libraries are a part of the Enhancement Toolkit. These libraries provide a series of features that give your users more control over generated programs created by VDT Application Code Generator.

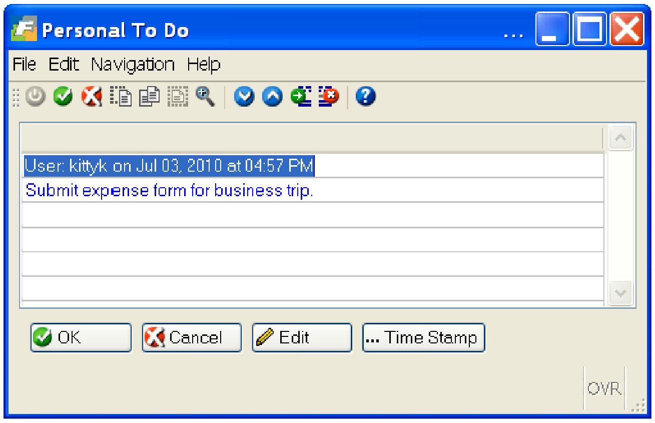
The User Control Libraries provide the following:

- a set of commonly-requested features that appear in programs you create with VDT Application Code Generator..
- a set of features that makes supporting and servicing generated applications easier.

Creating a To-Do List

A To-Do List gives the user a note pad to track the tasks they need to complete. Users can access their To-Do List by pressing [CTRL]-[t] or clicking when they are running an input program. To-Do lists are attached to a user's login ID, so the user's To-Do List is available from every generated input program.

The To-Do List feature gives users a note pad to track the tasks they need to complete.



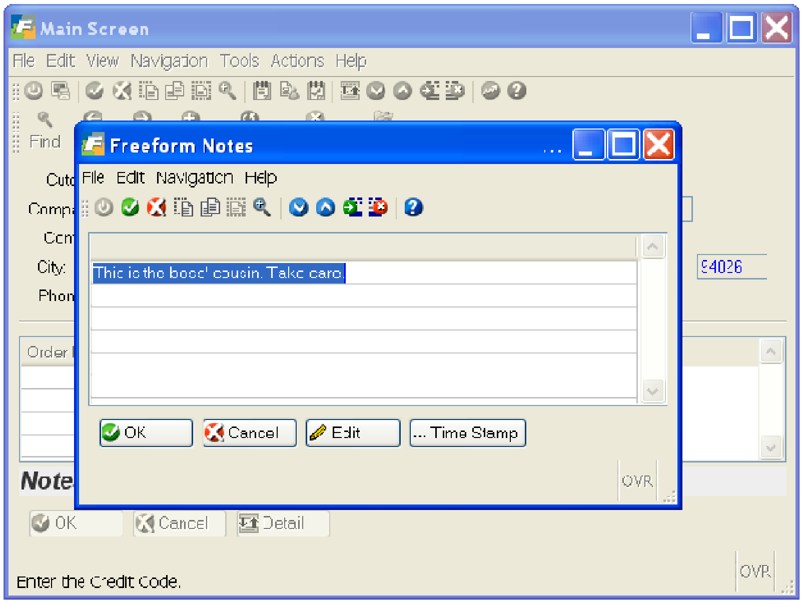
User: kittyk on Jul 03, 2010 at 04:57 PM

Submit expense form for business trip.

Adding Freeform Notes

Freeform Notes let users place notes in a data-entry document. The user presses [CTRL]-[n] and adds the note. The note is bound to the header portion of the input program. When a user defines a note, the note is permanently attached, and other users can view it.

Freeform Notes let users place notes on a data-entry document.



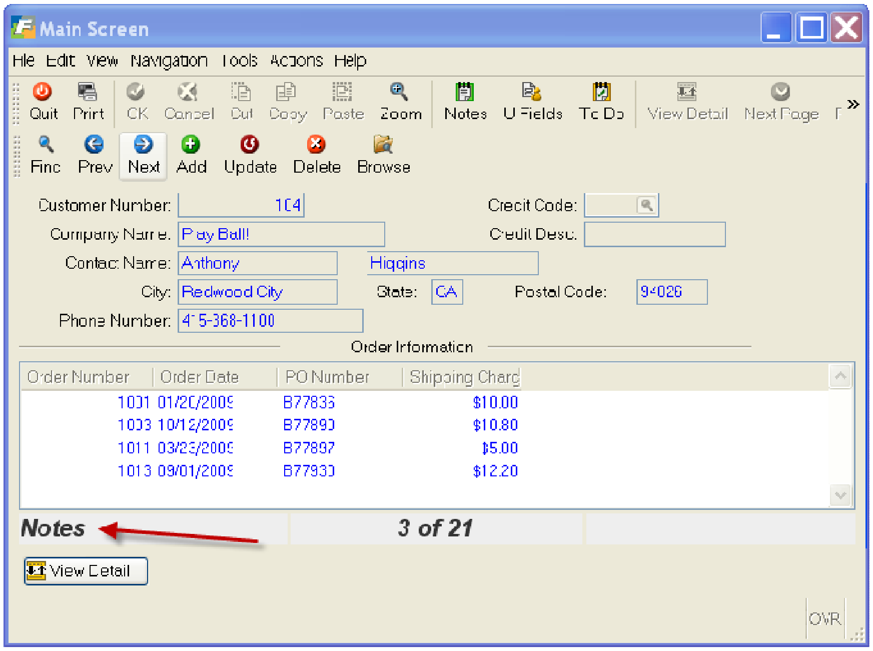
This is the boss' cousin. Take care.

94026

Enter the Credit Code.

When a note is attached to a document, the ***Note*** indicator appears in the lower left portion of the window.

The *Notes* indicator shows a Free-form Note is attached.


Main Screen
File Edit View Navigation Tools Actions Help
Quit Print OK Cancel Cut Copy Paste Zoom Notes U Fields To Do View Detail Next Page
Find Prev Next Add Update Delete Browse
Order Information
Notes
3 of 21
View Detail
OVR

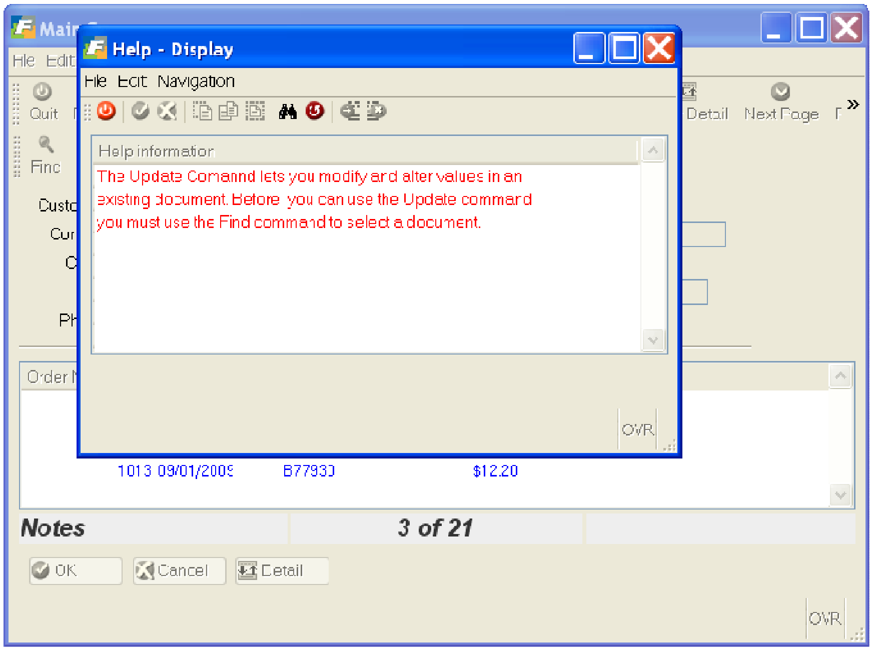
Customer Number:	104	Credit Code:	
Company Name:	Pay Ball!	Credit Desc:	
Contact Name:	Anthony	Higgins	
City:	Redwood City	State: CA	Postal Code: 94026
Phone Number:	415-368-1100		

Order Number	Order Date	PO Number	Shipping Charg
1001	01/20/2009	B77833	$10.00
1003	10/12/2009	B77893	$10.80
1011	03/23/2009	B77897	$5.00
1013	09/01/2009	B77930	$12.20

Adding Help Text

The VDT Application Code Generator also provides a context sensitive help system, which both you and program users can update and modify. When users have questions about input fields, commands, or any program control, they can press [CTRL]-[w] or click ? to see help information. Users can modify the help text by clicking ↺ or choosing Edit/Update from the menu toolbar.

Context sensitive help gives users the ability to access specific help information about input fields, commands, or any program control.



The Update Command lets you modify and alter values in an existing document. Before you can use the Update command you must use the Find command to select a document.

1013 09/01/2009 B77930 $12.20

Setting up Hot Keys

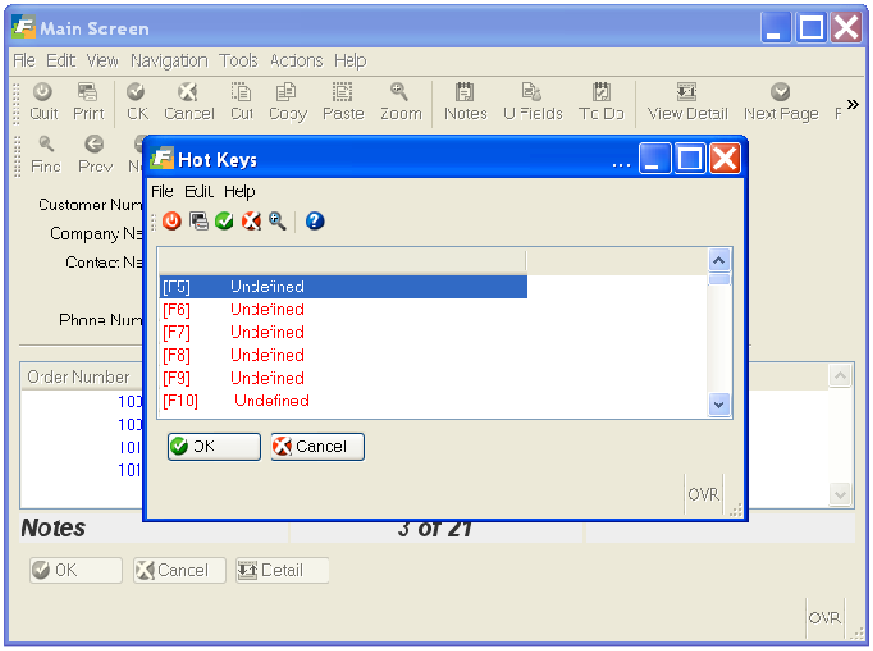
Hot Keys let users map keys on the keyboard to specific program events including custom Navigation events (see "Defining Navigation Events" below). To access the Hot Keys pop-up menu, users can press [CTRL]-[e] or choose Hot Key Definitions from the Tools pull-down menu. The Hot Keys pop-up menu serves three purposes:

1. **It lets users see how their keys are mapped.**

2. **It lets users customize their work environment and change their default Hot Key settings.**

3. **It gives users the ability to assign their own Navigation events to Hot Keys.**

Hot Keys let users map their keyboard to specific program events.



[F5] Undefined
[F6] Undefined
[F7] Undefined
[F8] Undefined
[F9] Undefined
[F10] Undefined

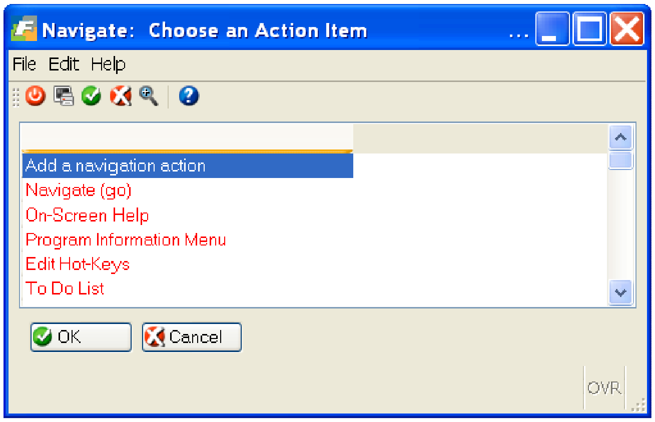
Hot Keys are defined in the Hot Keys window. To access the Hot Keys window, users must highlight the key they want to define on the Hot Keys menu and press [CTRL]-[z].

Defining Navigation Events

Navigation gives users the ability to define custom program events. These events can perform a number of useful tasks, such as suspending one program to jump to another one. Users can define Navigation events to go with an assortment of predefined Navigation events. When users press [CTRL]-[g], the Navigation pop-up menu appears.

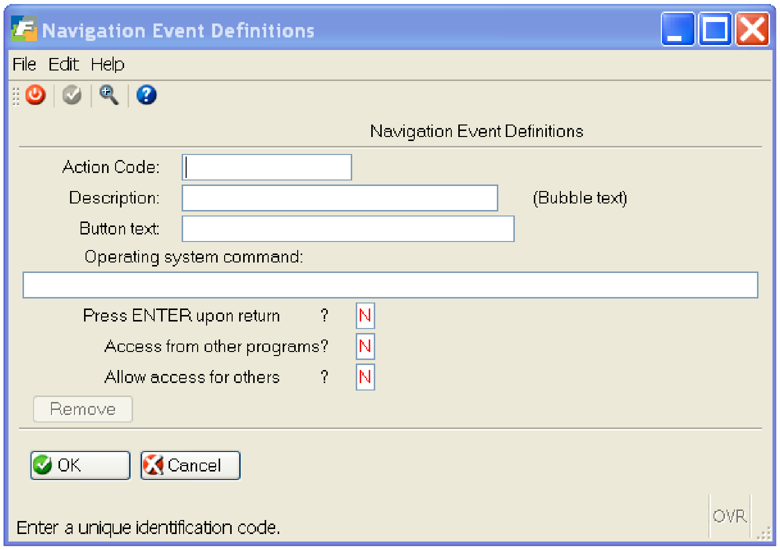
The Navigation pop-up menu lets users select from a list of predefined Navigation events.

Users can use this menu to create Navigation events.



Users can add Navigation events by selecting "Add a navigation action" from the Navigation menu.

The Navigation Commands window lets users define new Navigation events.



Action Code:

Description: (Bubble text)

Button text:

Operating system command:

Press ENTER upon return ? N

Access from other programs? N

Allow access for others ? N

Enter a unique identification code.

You must name your Navigation event in the Action Code field. You also need to describe your event in the Description field. If you are entering an operating system event, enter the operating system command in the "Operating system command" field. For example, if this event starts another program, enter the program command in this field.

The remaining fields on the Navigation Commands window are Y/N fields. "Press ENTER upon return" makes the user press [ENTER] once the event terminates. The "Access from other programs" field specifies whether this event can be run from other programs or not. The final field, "Allow access for others" specifies if others can use this event.

To define a Navigation event:

1. **Press [CTRL-g] to open the Navigation pop-up menu.**

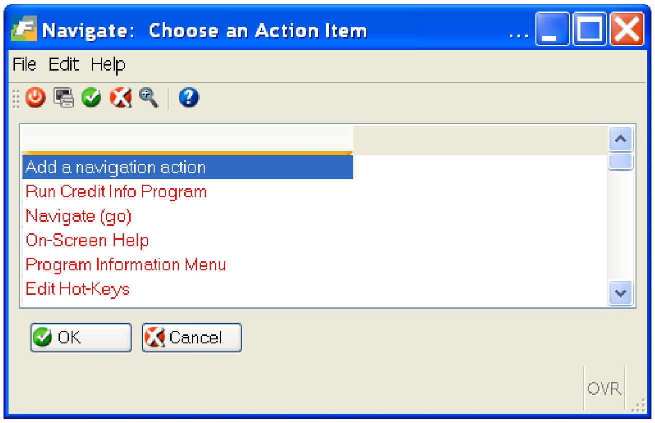
2. **Select "Add a navigation action" from the menu.**

   The Navigation Commands window appears.

3. **Complete the Navigation Commands window and press [ENTER].**

   Once you define a Navigation event, it appears on your Navigation menu.

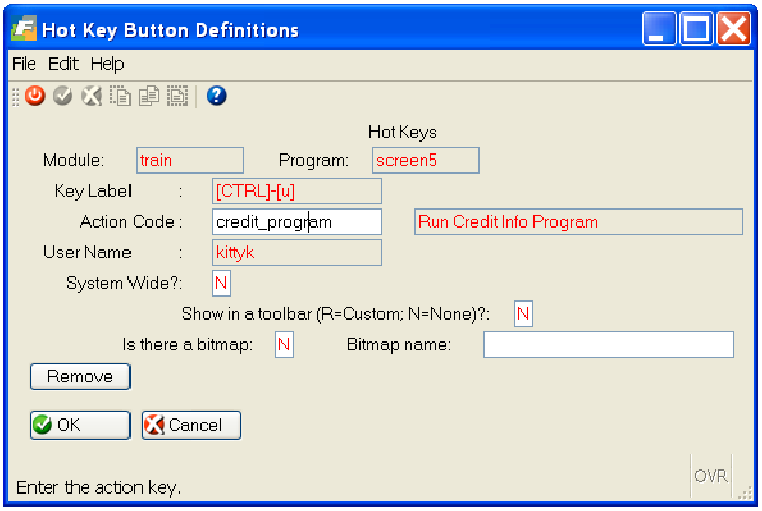
This example shows the Run Credit Info Program event, which is a custom Navigation event defined by the user.



Mapping Hot Keys to Navigation Events

You can also combine the power of Hot Keys and Navigation by defining custom Hot Keys to operate your Navigation events. You set Hot Keys to work with Navigation events in the Hot Keys window.

The Hot Keys window lets you assign Hot Keys to Navigation events.



Module: train
Program: screen5
Key Label : [CTRL]-[u]
Action Code : credit_program Run Credit Info Program
User Name : kittyk
System Wide?: N
Show in a toolbar (R=Custom; N=None)?: N
Is there a bitmap: N
Bitmap name:
Enter the action key.

The most important field is the Action Code field. This field corresponds to the Action Code you gave the event in the Navigation Commands window (page 9). The System Wide? field specifies if the Hot Key is available to all system users.

To map a Hot Key to a Navigation event:

1. **Define your custom Navigation event.**

2. **Press [CTRL]-[e] to open the Hot Keys pop-up menu.**

3. **Highlight an undefined key and press [CTRL]-[z].**

   The Hot Keys window appears.

4. **Complete the Hot Key window and press [Enter].**

Section Summary

- The User Control Libraries are a part of Enhancement Toolkit. These libraries provide a series of features that give your users more control over generated programs.
- A To-Do List gives users a note pad to track the tasks they need to complete.
- Freeform Notes let users place notes on a data-entry document.
- Users can use the Error Message window to check error information. In addition, users can log the errors they encounter and add more information describing the error to the Error window.
- The *VDT* Application Code Generator also provides a context sensitive help system, which both you and program users can update and modify.
- Hot Keys let users map their keyboard to specific program events including custom Navigation events.
- Navigation gives users the ability to define custom program events. These events perform a number of useful tasks, such as suspending one program to jump to another one.
- You can combine the power of Hot Keys and Navigation by defining custom Hot Keys to operate your Navigation events.

Exercise 7A

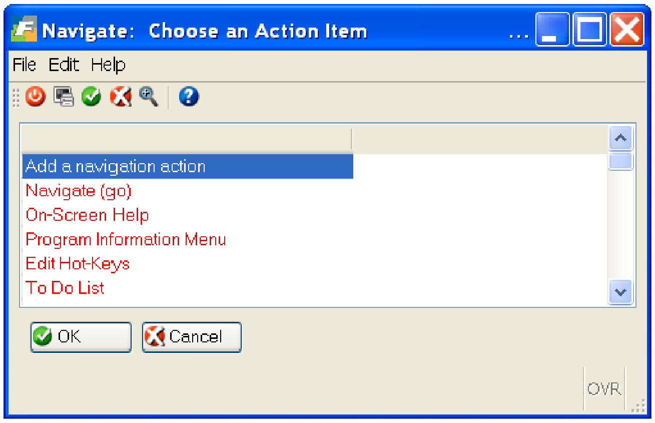
Objective: To place a navigation event in your Customer Entry program. You will add an event to check the amount of disk space available on your computer.

Access the Navigation Menu

This exercise starts from your running Customer Entry program. If not done already, start this program.

1. **From anywhere within your Customer Entry program, press [CTRL]-[g].**

   The Navigate pop-up menu appears.



Add a navigation action
Navigate (go)
On-Screen Help
Program Information Menu
Edit Hot-Keys
To Do List

As you can see, this menu already has several navigation events already defined. You can select any of these events to see what they do.

2. **Select Add a navigation event (option one) from the Navigate menu.**

   The Navigate Commands window appears.

Enter a Navigation Command to Check Disk Space

1. **Using the Navigate Commands window, set Action Code to `check_disk`.**

   The Action Code field contains a unique name for the event you are defining. You should try to make this name as descriptive as possible.

2. **Set Description to Check Disk Space,**

3. **Set the Operating system command field to the LINIX command that checks your disk space (typically the `df` command).**

4. **Enter a Y in the Press ENTER upon retum? field.**

   When the **`df`** command is executed, it will return to the program. Many times commands, such as **`df`**, return too quickly. Therefore, the Press [ENTER] prompt pauses after the UNIX command terminates so you can read its output.

5. **Press [ENTER] to save `check_disk`.**

Run check_disk

1. **Invoke the Navigate menu again by pressing [CTRL]-[g].**

   Notice how **`check_disk`** appears as the second option on the menu.

2. **Select `check_disk`.**

   The **`df`** command runs and its output is displayed to the terminal window. Once complete, the Press [ENTER] prompt appears.

3. **Press [ENTER] to return to your program.**

Edit check_disk

You can always edit a navigation event.

1. **Invoke the Navigate menu again (press [CTRL]-[g]).**

2. **Highlight the Check Disk Space option and press [CTRL]-[z].**

   The Navigate Commands window appears.

3. **Edit `check_disk` or press [ENTER] to save it as is.**

4. **Remain in your Customer Entry program and continue to Exercise 7B.**

Exercise 7B

Objective: To create a navigation event that runs a separate program.

Access the Navigation Menu

1. **From anywhere within your Customer Entry program, press [CTRL]-[g].**

   The Navigate pop-up menu appears.

2. **Select Add a navigation event (option one) from the Navigate menu.**

   The Navigate Commands window appears.

Add an Event to Call the Credit Entry Program

1. **Set Action Code to `credit_program`.**

2. **Set Description to Run Credit Info Program.**

3. **Set Operating system command to:**

   ```
   cd $HOME/labs/aw.4gm/i_cred.4gs; fglrun i_cred.42r –d $DBNAME
   ```

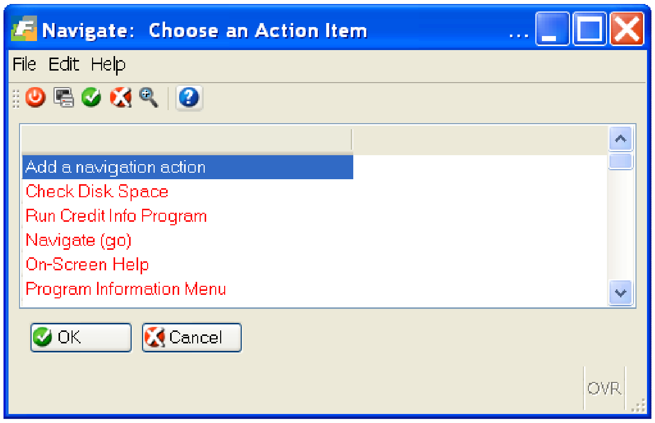
   This command changes to the **`i_cred.4gs`** directory and starts the Credit Entry program.

4. **4. Press [ENTER] to save credit_program.**

   Note that you do not have to set the Press [ENTER] upon return field. When you exit the Credit program, you return directly to the Customer Entry program.

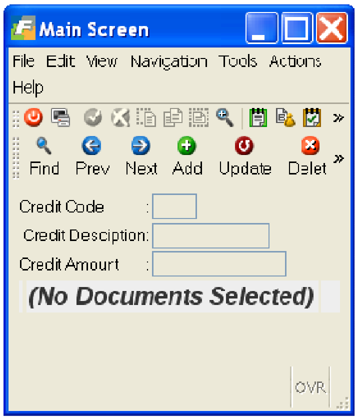
Use the credit_program Event

1. **Initiate the Navigate pop-up menu.**



2. **Select Run Credit Info Program.**

   The Credit Entry program, which you created in Exercise 3, starts.



3. **Select Quit from the Credit Entry program's ring menu.**

   The Credit Entry program exits and you return to the customer Entry program.

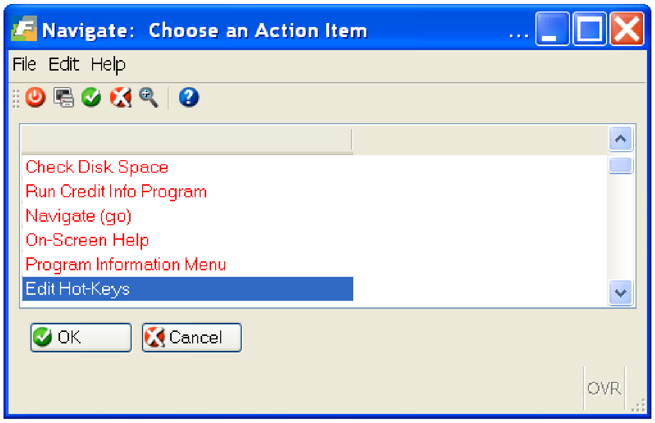
Exercise 7C

Objective: To map a hot key to the `credit_program` event.

Edit Hot Keys

1. **From Customer Entry, initiate the Navigate pop-up menu again (press [CTRL]-[g]).**

2. **Select the Edit Hot-Keys option.**



Check Disk Space
Run Credit Info Program
Navigate (go)
On-Screen Help
Program Information Menu
Edit Hot-Keys

The Hot Keys pop-up menu appears.

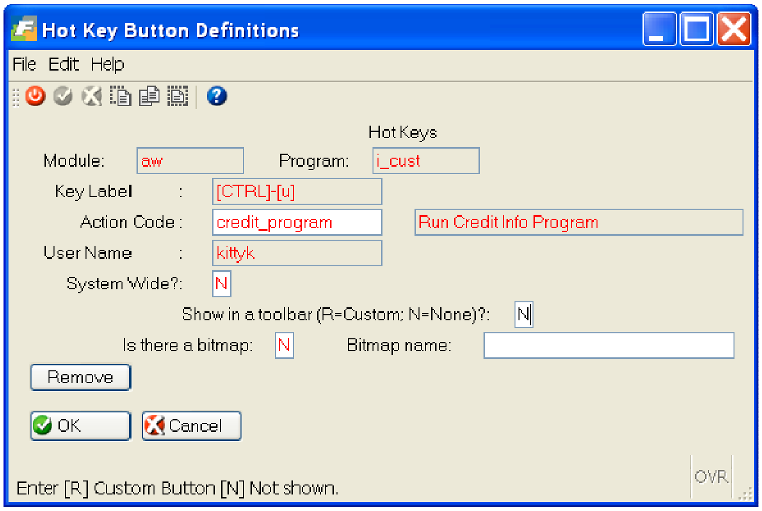
3. **Scroll down to near the bottom of the list. Highlight [CTRL]-[u] (which is Undefined) and press [CTRL]-[z].**

The Hot Keys window appears.

Enter the Navigation Event Codes

1. **Set the Action Code field to credit-program.**

If you forget the Action Code, you can Zoom on this field.



Module: aw Program: i_cust
Key Label : [CTRL]-[u]
Action Code : credit_program Run Credit Info Program
User Name : kittyk
System Wide?: N
Show in a toolbar (R=Custom; N=None)?: N
Is there a bitmap: N Bitmap name:

Enter [R] Custom Button [N] Not shown.

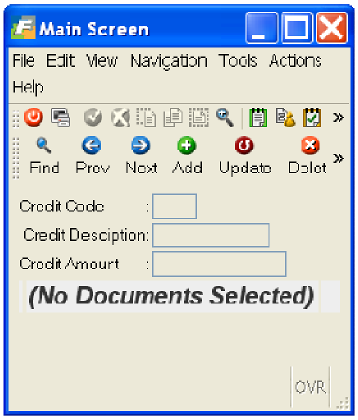
2. **Set Show in toolbar (R=Custom; N=None) field to N. Custom Toolbars are covered later on in this class.**

3. **Press [ENTER] to save the [CTRL]-[u] hot key mapping.**

Press [CTRL]-[u] to Start the Credit Info Program

1. **From anywhere in your Customer Entry program, press [Ctrl]-[u].**

   The Credit Entry program starts.



(No Documents Selected)

2. **When finished, exit the Credit Entry program and return to the Customer Entry program.**

Edit a Hot Key Definition

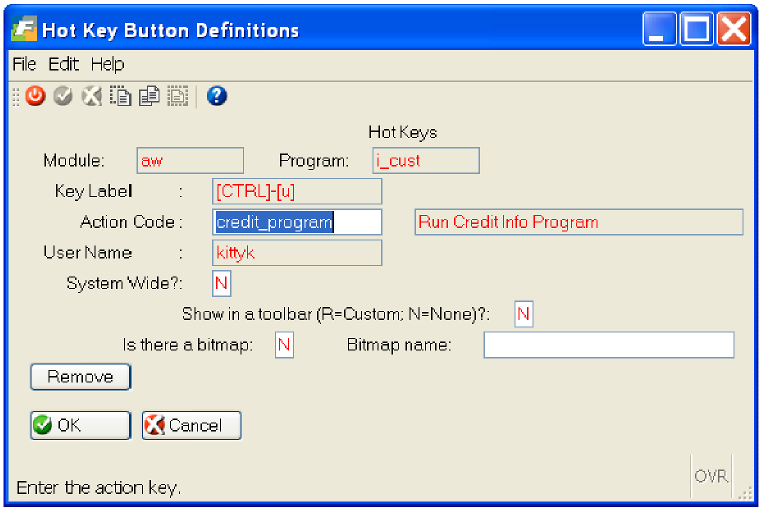
If you ever need to remap a hot key you can change its definition.

1. **Press [CTRL]-[e] to initiate the Hot Keys pop-up menu.**

   The [CTRL]-[e] sequence lets you access this menu directly; you can also select Edit Hot-Keys from the Navigate pop-up menu or choose Hot Key Button Definitions from the Tools pull-down menu.

2. **Highlight the Hot Key you want to edit. For example, highlight the [CTRL]-[u] key.**

3. **Press [CTRL]-[z] to bring up the Hot Keys window.**



Hot Keys

Module: aw Program: i_cust

Key Label : [CTRL]-[u]

Action Code : credit_program Run Credit Info Program

User Name : kittyk

System Wide?: N

Show in a toolbar (R=Custom; N=None)?: N

Is there a bitmap: N Bitmap name:

Enter the action key.

4. **From the Hot Keys window, you can edit the Action Code value.**

   For this exercise, do not change the [CTRL]-[u] hot key. It is enough for you to know how to edit the values in this window.

5. **Press [ENTER] to return to the Customer Entry program.**

6. **Quit out of both the Customer Entry program and the VDT Form Designer.**

Chapter 8

Using the VDT Application Code Generator

Main topics:

- VDT Application Code Generator Overview
- Understanding Library Code and Local Code
- Classifying Functions
- Starting the Tools from the Command Line

VDT Application Code Generator Overview

The VDT Application Code Generator functions as the back-end to the VDT Form Designer. You use the VDT Form Designer to create a form image and the VDT Application Code Generator to create code based on that form image.

The VDT Application Code Generator relies on a form specification (*.per) file to create the 4GL source code. When you save a form image with the VDT Form Designer, a *.per file is created automatically. In a general sense, you must complete the following steps to develop an input program:

1. **Create a form image with the VDT Form Designer.**

2. **Save your form image in the VDT Form Designer to create a form specification (*.per) file.**

3. **Invoke the VDT Application Code Generator, which reads the *.per file and creates 4GL source code based on the instructions in the specification file (see "Creating Form Specification (*.per) Files" on page 105).**

4. **Use the make utility (`fg.make`) to compile the source code into object code and then link it into (*.42r) pseudo code.**

5. **Run the input program and use its toolbars to add, update, and delete data from the database.**

The following figure outlines the steps you take to develop a complete input program using the VDT Application Code Generator.

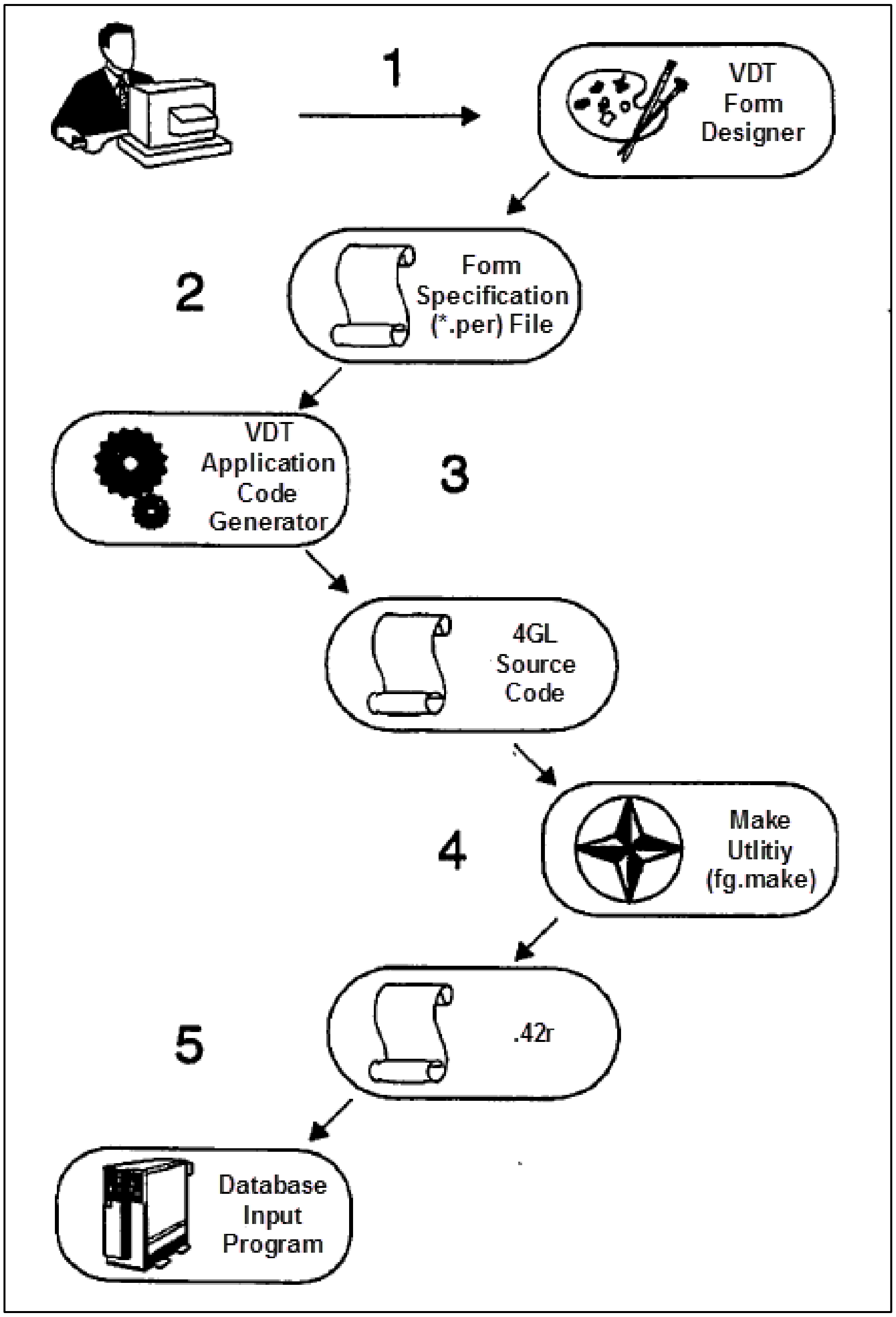
Step 1: Create a form image with the VDT Form Designer.

Step 2: Save your form image to create a form specification (*.per) file.

Step 3: Invoke the VDT Application Code Generator to read the .per file and create 4gl source code.

Step 4: Use the make utility (fg.make) to compile the source code into object code and link it to a .42r pseudo code file.

Step 5: Run the input program and use its commands and toolbars to update data in the database.



Files Created During the Development Process

During the development process, there are several files that get created. Each file is given a special file extension to help you identify its file type.

Tool	File Type File	Extension
VDT Form Designer	Form Specification Files	*.per
VDT Application Code Generator	Compiled Form Files	*.42f
VDT Application Code Generator	Genero-4GL Source Code Files	*.4gl
`fg.make`	Compiled Object Files	*.42m
`fg.make`	Executable Files	*.42r

For example, if you build training program 3, these files are created:

Form Specification	Compiled Form	Source Code	Object	Executable	Other
browse. per	browse.42f	browse.4g1	browse.42m	screen3.42r	Makefile
cust_zm.per	cust_zm.42f	cust_zm.4g1	cust_zm.42m		errlog
order.per	order.42f	detail.4g1	detail.42m		filelist.42s
stk_mnu.per	stk_mnu.42f	globals4gl	Globals.42m		tags
stockzm.per	stockzm.42f	header.4g1	header.42m		linklist
		main.4gl	main.42m		
		midleve1.4g1	midleve1.42m		
		stk_mnu.4g1	stk_mnu.42m		
		stockzm.4g1	stockzm.42m		

Note

The VDT Application Code Generator also provides a code merge (fglpp) utility that creates original (*.org) files and merges extension and trigger (*.ext and *.trg) files to create a *.4gl file (see "Featurizer Overview" on page 250).

Understanding Library Code and Local Code

You can classify code into two main categories:

1. library Code
2. Local Code

Library code has the following characteristics:

- It is shared by different programs.
- It is static; the code never changes.
- It is data independent.
- It is generic.
- It is not created by the VDT Application Code Generator. Library code is hand-coded and always available for use.

Many program features, such as the Standard Toolbar commands, are created from library code:

Local code has the following characteristics:

- It is used by only one program.
- It is designed to change over time.
- It is data dependent.
- It is specific.
- It is created by the VDT Application Code Generator.

There are several visible examples of local code as well, such as reading in a record, adding a record, and saving a record.

Classifying Functions

VDT Application code is highly modular, which means that all the code is written within functions. Most of these functions are small, less than 20 lines long. Functions have the following characteristics:

- All code is organized into logical code blocks.
- Possible points of modification are easily identifiable.
- All functions contain comments that describe specifically what they do.
- Function code can be reused.
- Generated functions have similar names, thus establishing consistent naming conventions.

Functions are classified according to their use. Functions can be divided into three classes:

1. Upper-level Functions
2. Low-level Functions
3. Mid-level Functions

Upper-level functions have the following attributes:

- they are data independent.
- they are generic.
- they are usually library functions.
- they are not created by the VDT Application Code Generator.
- they are typically left unchanged.
- they are usually prefixed with `ring_` or gn_

Low-level functions have the following attributes:

- they are data dependent.
- they are specific.
- they are created by the VDT Application Code Generator.

- they are frequently changed.
- they are usually prefixed with `1lh__` or `1ld_`

Midlevel functions have the following attributes:

- they perform *housekeeping* tasks, such as initializing variables, preparing cursors, and performing construct statements.
- they are created by the VDT Application Code Generator.
- they are typically left unchanged.
- they are always prefixed with `mlh__` or `mld_`

Starting the Tools from the Command Line

In chapter 2, you learned how to start the VDT Form Designer and run the VDT Application Code Generator and Fitrix make utility from within the VDT Form Designer. These programs can also be run from the LINUX command line.

Tool	**Command**
VDT Form Designer	fg.form
VDT Application Code Generator	fg.screen
Make Utility	fg.make

Each command also uses several command flags that you can use to alter how the command works.

VDT Form Designer Command Syntax

The VDT Form Designer uses the following command flags and syntax.

```
fg.form [-dbname database] [-o{O-5}] [-f] [-y]-n]
[-p file.per]
```

`-dbname` *`database`* Specifies the database on which the VDT Form Designer operates.

`-o(0-5)` Specifies the level of information displayed during code generation. To display the least amount of information use `-o0`. To display the greatest amount of information use `-o5`

`-f` Specifies a *fast* generation. The `-f` flag and `–o0` are synonymous.

`-y|-n` Specifies interactive or non-interactive generation mode. The `-y` flag answers yes to all code generation prompts.

`-p` *`file`*`.per` Specifies the name of the form specification file to automatically loads upon start-up.

VDT Application Code Generator Command Syntax

The VDT Application Code Generator uses the following command flags and syntax.

```
fg.screen [-dbname database] [-o{0-5}] [-f] [-y|-n]
[file.per…]
```

`-dbname` *`database`* Specifies the database on which the *Screen* Code Generator operates.

`-o(0-5)` Specifies the level of information displayed during code generation. To display the least amount of information use `-o0`. To display the greatest amount of information use `-o5`

`-f` Specifies a *fast* generation. The `-f` flag and `–o0` are synonymous.

`-y|-n` Specifies interactive or non-interactive generation mode. The `-y` flag answers yes to all code generation prompts.

`-p` *`file`*`.per` Specifies the name(s) of the form specification file(s) that the *Screen* Code Generator reads and processes.

For a description and the syntax of the `fg.make` script see "Compiling Generated Code" on page 238. And for a description of code merging utility (`fglpp`) see "Featurizer Overview" on page 250.

Section Summary

- The VDT Application Code Generator functions as the backend to the VDT Form Designer. You use the VDT Form Designer to create a form image and the VDT Application Code Generator to create code based on that form image.
- During the development process, there are several files that get created. Each file is given a special file extension to help you identify its file type.
- You can classify code into two main categories: (1) Library Code and (2) Local Code.
- VDT Application code is highly modular, which means that all the code is written within functions. Most of these functions are small, less than 20 lines long.
- Functions are classified according to their use. Functions can be divided into three classes: (1) Upperlevel Functions, (2) Lowlevel Functions, and (3) Midlevel Functions.
- The VDT Form Designer, VDT Application Code Generator, and make utility can be run from the LINUX command line.

Exercise 8A

Objective: To build the Customer Entry program from outside the Form Painter. You will rebuild the entire application from the form specification (*.per) files that you created with the VDT Form Designer.

Make a Backup Directory

1. **Click the 🖥 to get to the LINUX prompt if you are not already there.**

2. **Move to the $HOME/ labs/aw. 4gm directory:**

```
cd $HOME/labs/aw.4gm
```

3. **Create a i_cust .bak directory to hold a copy the files in your i_cust.4gs directory:**

```
mkdir i_cust.bak
```

4. **Copy all of the files in i_cust. 4gs to i_cust.bak:**

```
cp i_cust.4gs/*  i_cust.bak
```

5. **Move to your i_cust. 4gs directory:**

```
cd i_cust.4gs
```

Remove Everything Except Your *.per Files

1. **Remove all the files in i_cust.4gs except those with a *.per extension:**

```
cp *.per ../
rm*
mv  ../*.per ./
```

This command leaves `i_cust.4gs` with two files: `cred_zm.per` and `cust .per`.

2. **List your files to verify that only these two files remain:**

```
ls
```

Generate 4GL Code

The `cred_zm.per` and `cust.per` files contain all the information that is needed for the VDT Application Code Generator to re-create source code for your Customer Entry program.

1. **From the i_cust . 4gs directory, enter:**

   fg.screen –o0 -y *.per

   The -o flag specifies the amount of screen output to display. A 0 indicates the minimum amount of output. A 5 indicates the maximum amount. Finally the **`-y`** flag automatically answers "yes" to all prompts.

   The VDT Application Code Generator reads the instructions in the *.per files and creates 4GL source code. When the VDT Application Code Generator is finished, the LINUX prompt reappears.

2. **From the UNIX prompt, list the files in i_cust.4gs:**

   ```
   ls
   ```

   As you can see, the *Screen* Code Generator creates a number of files, including a Makefile and multiple source code (*.4gl) files.

Compile the Code

After generating code, you must convert it into object code, link it to the libraries, and build an executable. All these tasks are handled by the compilation utility, which is known as `fg.make`.

1. **From the `i_cust.4gs` directory, enter:**

   fg.make

   The make utility runs. When it is finished, the LINUX prompt reappears.

2. **List your files again:**

   ls

   Notice that now there are object files (*.42m) and a program file (*.42r).

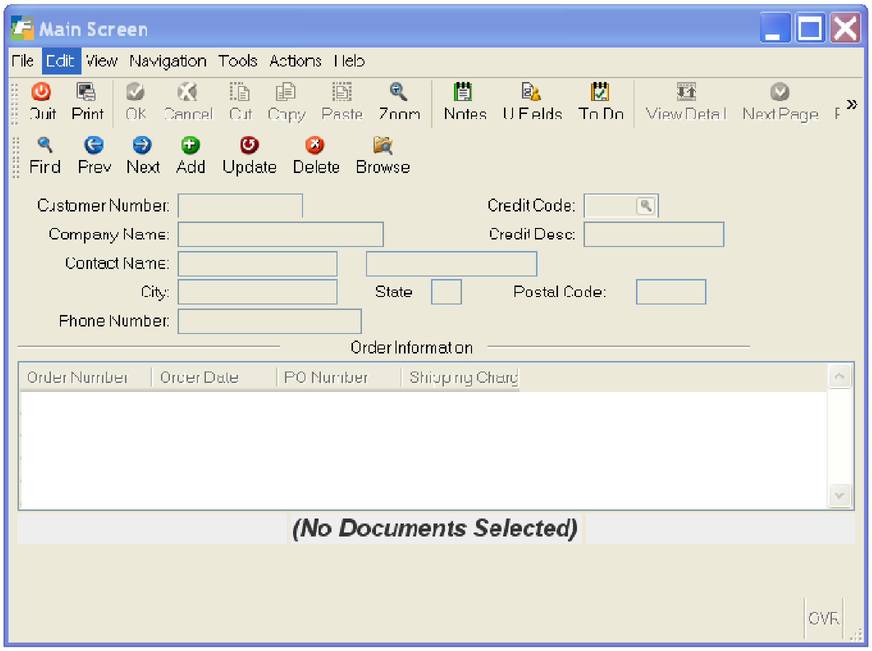
Run the Customer Entry Program

Once `fg.make` is finished, you can run the Customer Entry program again.

1. **Use the following command to run the Customer Entry program. Substitute your databse name for student1 below.**

   fglrun i_cust.42r –d student1

   The Customer Entry program starts.



(No Documents Selected)

2. **Quit the Customer Entry program.**

Exercise 8B

Objective: To gain a basic knowledge of the Fourjs Genero 4GL source code built by the VDT Application Code Generator and to become familiar with Fitrix standards and code structures.

List the Files

1. **List the files in i_cust.4gs:**

   ls

   Notice that there are several files with a *4gl extension. These are source code files.

Examine midlevel.4g1

1. **Use vi to open midlevel .4g1:**

   vi midlevel.4g1

   This file contains generated source code that handles "housekeeping" chores such as initializing variables, preparing cursors, and locking records.

   Notice how all the code is contained in functions. VDT Application Code Generator source code is extremely modular.

   Each function in **`mid1eve1.4g1`** is prefaced with **`ml`**. These characters stand for midlevel. Both the header and detail portion of Customer Entry have midlevel functions associated with them. For this reason, midlevel functions are further classified as **`m1h`** and **`m1d`**, which stand for midlevel header and midlevel detail respectively.

2. **Exit from midlevel. 4gl.**

Examine header.4gl and detail.4g1

1. **Use vi to look through both header. 4g1 and detail. 4gl.**

   Both files contain lowlevel functions. The **header.4gl** lowlevel functions handle header section activities such as inserting, updating, deleting, and validation checking. The **detail.4gl** lowlevel functions do much of the same, but they control the detail portion of the screen.

   Notice how each **header.4gl** function names are prefaced with **llh** and **detail.4gl** functions are prefaced with **lld**.

3. **Exit these files.**

Examine cred_zm.4gl

1. **Use vi to open cred_zm.4gl**

   This file corresponds to your Credit Information zoom screen. Notice that there are sets of functions, prefaced by different capital letters that perform different tasks.

Preface	Use
A	Opens a Zoom window.
Q	Queries for selection criteria.
R	Reads records into the program.
O	Displays records to the zoom window.
Z	Closes the zoom window.

4. **Exit cred_zm.4gl.**

Chapter 9

Creating Triggers

Main topics:

- Trigger Overview
- Understanding the Trigger Concept
- Creating Triggers
- Merging Triggers into Code

Trigger Overview

In most cases, you can use the Form Editor in the VDT Form Designer to accomplish everything an input program requires. The Form Editor lets you:

- define input fields
- specify field attribute logic, such as whether the field can be entered
- attach zoom screens
- attach lookups to validate input values

On occasion, however, you must make custom enhancements to an input program that you cannot create in the Form Editor. For example, you might want to include some of the following enhancement types:

- after field logic
- before field logic
- after input logic
- after change in logic
- before input logic
- after row logic
- before row logic
- event handling logic

You can create all these enhancements using triggers, which are essentially code-level modifications to an input program.

Understanding the Trigger Concept

Triggers are enhancements made directly to the source code generated from the VDT Application Code Generator. A trigger is an automatic way of placing code-level enhancements into the source code.

Triggers are named for logical points in the code. The following list contains some common triggers:

- `after_field`
- `before_field`
- `after_input`
- `before_input`
- `on_event`

Triggers get placed in trigger (".trg) files. A trigger file functions much like a form specification (".per) file. Both contain instructions that the VDT Application Code Generator reads and understands.

A single trigger file can contain more than one trigger.

Triggers do not require you to be an expert on code structure. You simply work with the VDT Form Designer to define the logical points at which your triggers act.

Trigger (*.trg) files should have the same name as the form specification file that they relate to. For example, the `order.trg` file relates to the `order.per` form specification file.

Creating Triggers

Creating triggers is a straightforward task. There are two ways you can construct triggers:

1. **You can use the VDT Form Designer.**

2. **You can create them by hand in trigger (*.trg) files.**

Perhaps the best way to write your first trigger is with the VDT Form Designer; it provides the simplest environment to learn about trigger creation.

Using the Form Painter to Create a Trigger

Before you create a trigger using the VDT Form Designer, you should create a form image and form specification file (see "Creating a Form Image" on page 38).

Once you create a program from which to work, you can define a trigger.

To add a new trigger using the Form Painter:

1. **Select Triggers » from the Define pull-down menu.**

   If your screen type contains more than one input area, the Choose a Trigger Class pop-up menu appears.

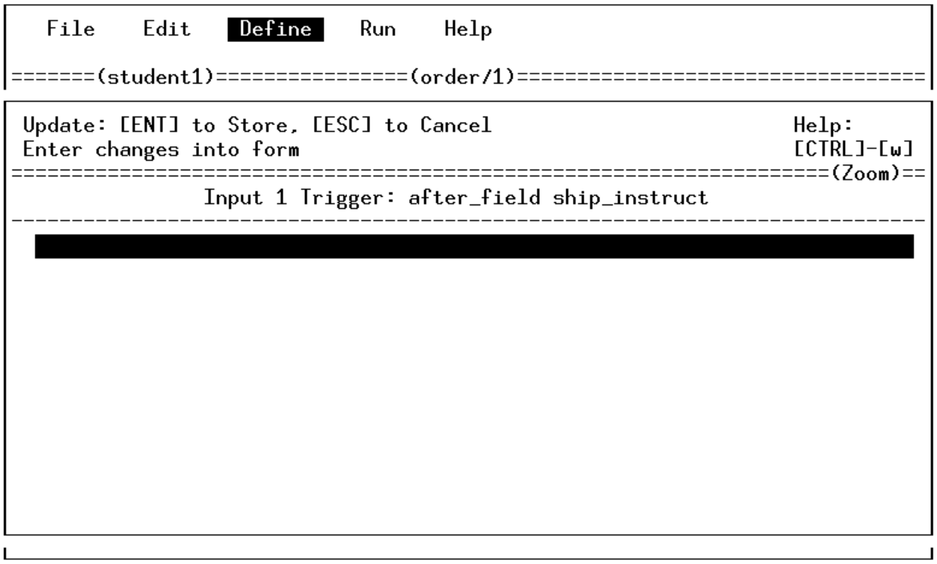
2. **Select the input area for your trigger.**

   The Choose a Trigger pop-up menu appears.

3. **Select the trigger you want to create.**

   Depending on the trigger you select, subsequent pop-up menus appear. For example, if you select the `after_field` trigger, the Choose a Field pop-up menu appears. After you choose a field, the Form Painter opens the Trigger Editor.

Use the Trigger Editor to enter custom 4GL logic.

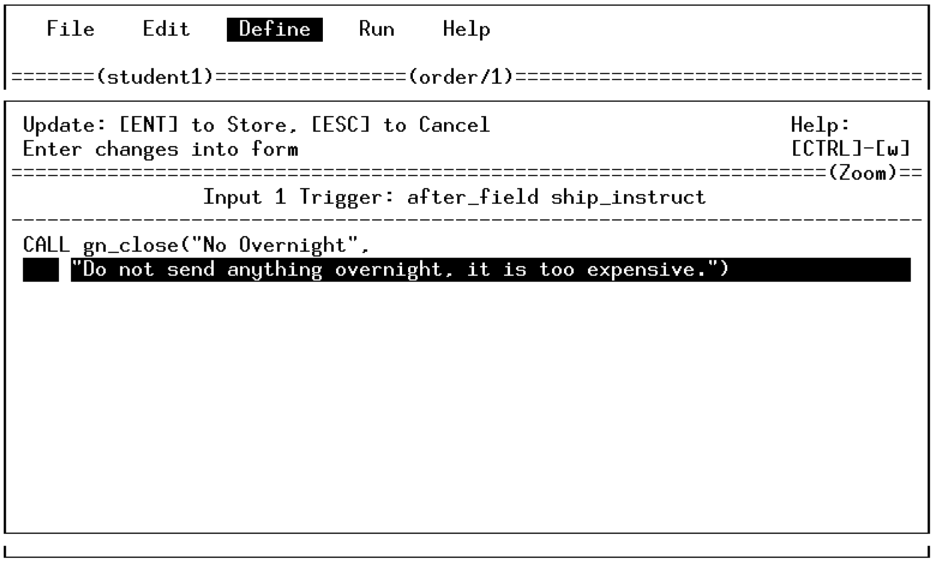


```
  File    Edit    Define    Run    Help

=======(student1)===============(order/1)=================================

 Update: [ENT] to Store, [ESC] to Cancel                          Help:
 Enter changes into form                                          [CTRL]-[w]
=================================================================(Zoom)==
                Input 1 Trigger: after_field ship_instruct
-------------------------------------------------------------------------
```

4. **Enter the custom 4GL logic of your trigger using the Trigger Editor.**

   For example, after the shipping instruction field, you might want to display shipping rate information. With the Trigger Editor, you can specify 4GL logic that displays this information.

Your custom logic can simply display a message after a field.



```
  File    Edit    Define    Run    Help

=======(student1)===============(order/1)=================================

 Update: [ENT] to Store, [ESC] to Cancel                          Help:
 Enter changes into form                                          [CTRL]-[w]
=================================================================(Zoom)==
                Input 1 Trigger: after_field ship_instruct
-------------------------------------------------------------------------
 CALL gn_close("No Overnight",
       "Do not send anything overnight, it is too expensive.")
```

5. **Once you enter your trigger code, press [ESC] to store your trigger.**

   Your trigger gets saved to a trigger (*.trg) file.

6. **Press [CTRL-C] to exit the Choose a Trigger box.**

Creating Triggers by Hand

After a while, you might find it faster and more convenient to create triggers from outside the Form Designer. That is to say, you might want to create trigger files directly using vi or some other LINUX text editor. Creating triggers by hand can be as simple as using the VDT Form Designer as long as you follow the correct syntax.

All triggers follow the same general syntax:

```
input #
   trigger argument
         custom 4GL logic ...
         ;
```

Where # indicates the input area number, *trigger* indicates the trigger command, and *argument* indicates any argument that the trigger accepts.

For example, the following after_input trigger displays a short message:

```
input 1
   after_input
         CALL gn_close(“After Input”, "After input logic" )
            ;
```

Some triggers accept arguments. For example, this trigger accepts a field name (company) as a trigger command argument:

```
input 1
   after_field company
         ALL gn_close(“After Field”, "After field logic" )
         ;
```

For a complete list of triggers, trigger descriptions, and syntax refer to the VDT Application Code Generator Technical Reference.

Merging Triggers into Code

Once you create a trigger, you can merge it into your source code. To merge a trigger, however, you do not need to regenerate all your code. You can simply run either the make utility (`fg.make`) or the Featurizer (fglpp).

If you are using the VDT Form Designer, simply select the Compile 4GL option under the Run pull-down menu. If you are working from the command line, type:

```
   fg.make
```

or:

```
   fglpp [4gl-File-name]
```

Both commands initiate the Featurizer. The Featurizer reads your trigger (*.trg) file and places your code enhancements into the generated source code. When you run `fg.make`, the final source code (*.4gl) files contain your enhancement logic. The Featurizer saves your original source code in files with an *.org extension. When you run the Featurizer (fglpp) only, you can include a specific 4GL file name as an argument, or pass no argument to merge all 4GL's.

Section Summary

- Triggers are enhancements made directly to the source code generated from the VDT Application Code Generator. A trigger is an automatic way of placing code-level enhancements into the source code.
- You can create triggers using the VDT Form Designer or by hand.
- Triggers let you create custom modification to logical points in your program flow.
- There are a number of triggers that can be merged into 4GL source code. Triggers are saved in trigger (*.trg) files, these files are given the same name as the form specification files they relate to. For example the `order.trg` trigger file relates to the `order.per` form specification file.
- The Featurizer reads *.trg files and merges the enhancements into the generated source (*.4g1) code files.

Exercise 9

Objective: To add a simple `before_input` trigger to the Customer Entry program.

Open cust.per in the Form Painter

1. **Move to $HOME/labs/aw. 4gm/i_cust. 4gs directory:**

   ```
   cd $HOME/labs/aw.4gm/i_cust.4gs
   ```

2. **Start the Form Painter.**

3. **Select Open from the File pull-down menu to load cust.**

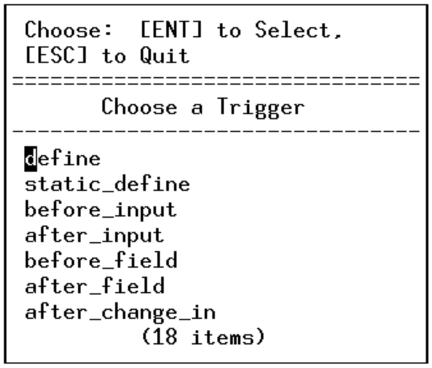
Create a before_input Trigger

1. **Select Triggers » from the Define pull-down menu.**

   The Choose a Trigger Class box appears**.**

2. **Select Input Area 1 from the Choose a Trigger Class box.**

   The Choose a Trigger list box appears.

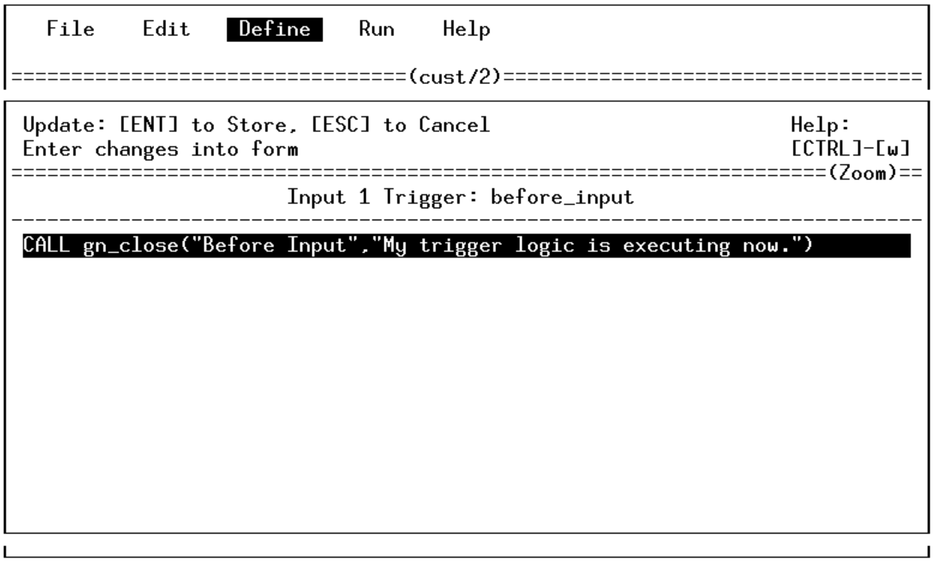


```
 Choose:  [ENT] to Select,
 [ESC] to Quit
================================
       Choose a Trigger
--------------------------------
 define
 static_define
 before_input
 after_input
 before_field
 after_field
 after_change_in
          (18 items)
```

3. **Select before_input from the Choose a Trigger list box.**

   An editing window appears.

4. **Complete a `gn_close` function call as follows:**



```
================================(cust/2)===================================
 Update: [ENT] to Store, [ESC] to Cancel                          Help:
 Enter changes into form                                          [CTRL]-[w]
===================================================================(Zoom)==
                      Input 1 Trigger: before_input
---------------------------------------------------------------------------
 CALL gn_close("Before Input","My trigger logic is executing now.")
```

5. **Press [ESC] to save this before_input trigger.**

   The Choose a Trigger list box appears again.

6. **Press [CTRL]-[c] to close the Choose a Trigger box.**

7. **Select Save Trg File from the File pull-down menu.**

Compile the Code

1. **Select Compile 4GL from the Run pull-down menu.**

   The compilation utility calls the Featurizer (the code merging utility). The Featurizer merges your custom "display" logic into the generated source code.

Run the Customer Entry Program

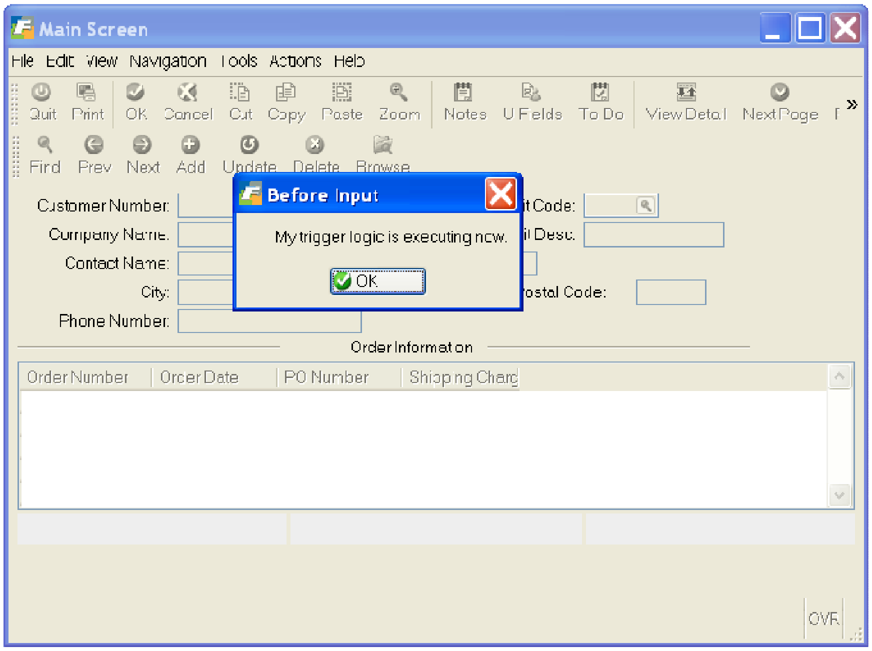
1. **Select Run 4GL Program from the Run pull-down menu.**

   The Customer Entry program starts.

Check the before_input Trigger

1. **Select Add from the ring menu.**

   Your custom "display" logic appears in a dialog box with an OK button.



My trigger logic is executing now.

2. **Finish adding the record.**

3. **Use Find to select a record and select update.**

   Again, your custom logic appears.

4. **Quit the program and the Form Painter.**

Examine header.4gl

1. **Use vi to open header. 4g1.**

2. **Search for before_input.**

   Notice that your custom logic is inserted just before the input command:

```
#_before_input
    # FGLPP BEGIN cust.trg (.4gs)
    CALL gn_close("Before Input","My trigger logic is executing now.")
    # FGLPP END   before_input
#_end
```

   The #_ characters mark a trigger tag. In other words, these symbols define locations where triggers can be inserted**.**

3. **Using vi, search for other trigger tags.**

   This step familiarizes you with the types of triggers that are available. You will be adding custom logic to some of these locations at a later time.

4. **Exit header. 4g1.**

Note

When you make a change to a form (such as adding a field or field label), you must rebuild the program by running both the VDT Application Code Generator and `fg.make`. If you are only adding custom code via triggers, save the trigger file then run the `fg.make`. The VDT Application Code Generator is not required

Chapter 10

Managing Screen to Table Flow

Main topics:

- Understanding Program Data Flow
- I/O Triggers
- Referencing Input Fields
- Common Global Variables
- The Scratch Variable

Understanding Program Data Flow

Before you start building input programs with VDT Application Code Generator, it is helpful to understand how data is handled by programs created with the VDT Application Code Generator. In a general sense, input programs must perform two tasks:

1. **Move data entered by the program user to the database.**

2. **Move data stored in the database to the screen.**

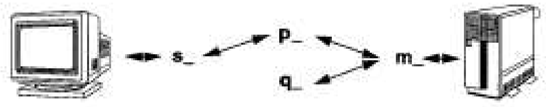
The Code Generator accomplishes both tasks by creating four records (`p_, m_, q_,` and `s_`) and two "prep" functions (`p_prep` and `m_prep`).

Data Flow Records

The p _ record: This record parallels the data elements defined on the screen. The p_ record only contains those fields displayed on your input program.

The m_ record: This record parallels information in the columns of a table. The m_ record contains variables with the same names as the columns in the database table.

Four records transfer data between the program user and the database.



The q_ record: This record contains all the columns not used by the input program but contained in the table.

The s_ record: This record contains values that get entered from or passed to the screen.

All records start with their various type (`p_, m_,` etc.). After the type, the record is named with the last six characters of the table name. For example `p_stomer` represents the p_ record for the `trcustomer` table.

After the table name, the p_ record is built from all the input fields used by the input program. The following example shows a typical p_record:

```
p_orders record   # Record like the order screen
     customer_num like trorders.customer_num,
     fname like trcustomer.fname,
     lname like trcustomer.lname,
     company like trcustomer.company,
     address1 like trcustomer.address1,
     address2 like trcustomer.address2,
     city like trcustomer.city,
     state like trcustomer.state,
     zipcode like trcustomer.zipcode,
     phone like trcustomer.phone,
     order_date like trorders.order_date,
     po_num like trorders.po_num,
     order_num like trorders.order_num,
     ship_instruct like trorders.ship_instruct,
     ship_weight like trorders.ship_weight,
     ship_charge like trorders.ship_charge,
     t_price money(10)
 end record,
```

The m_ record does not use the column names like the p_ record. Instead the m_ record uses * notation. For example, m_stomer.* represents the m_ record for the trcustomer table. The * notation is used to allow the m_ record to accept data all at once. The following shows two example m_ records:

```
m_orders record like trorders.*. # Record like the header table
m_ritems record like tritems.*, #  Record like the detail table
```

The q_ record is defined like the p_ record, but it contains all the table columns not used by the program as input fields. For example:

```
q_orders record    # Parallel order record
    row_id integer, # SQL rowid
    backlog like trorders.backlog,
    ship_date like trorders.ship_date,
    paid_date like trorders.paid_date
    #_define_1
    #_end
end record,
```

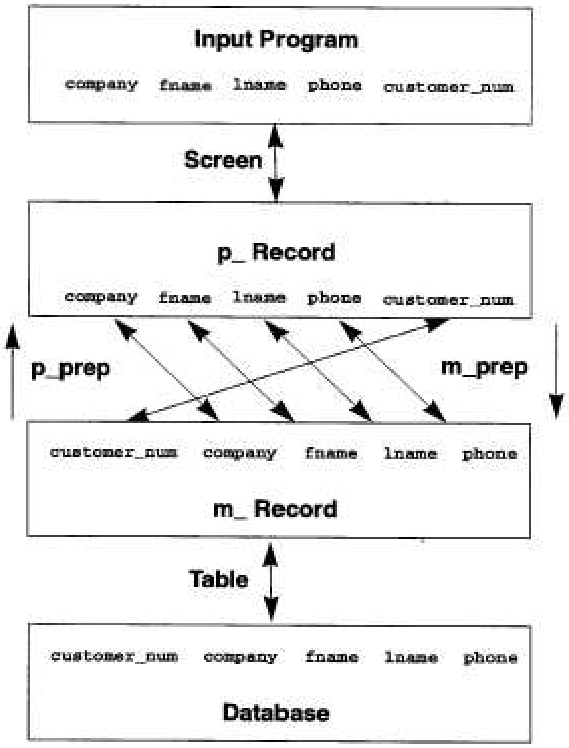
The s_ record gets defined in the Instruction section of the form specification file. It reflects the actual values displayed by the input program.

Data Flow Functions

The p_prep function: This function transfers data from the m_ record to the p_ record.

The m_prep function: This function transfers data from the p _ record to the m_ record.

Data flows between the input program and the database by way of four records and two "prep" functions.


Input Program
company fname lname phone customer_num
Screen
p_ Record
company fname lname phone customer_num
p_prep
m_prep
customer_num company fname lname phone
m_ Record
Table
customer_num company fname lname phone
Database

Lowlevel Functions Used by the Data Flow

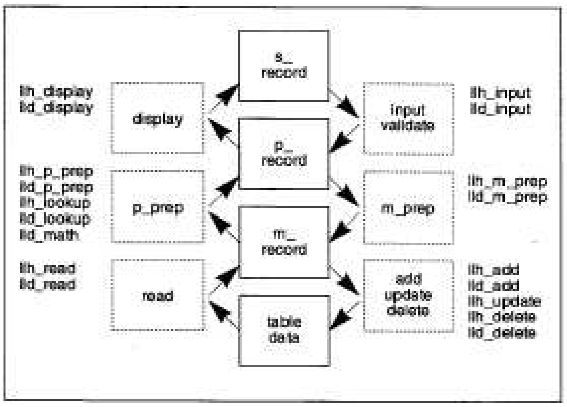
Lowlevel functions control data flow (as illustrated below). The header. 4g1 and detail. 4g1 files contain the data flow functions.:

Dataflow	**Header Functions**	**Detail Functions**
From Database to Input Program		
database to m_ record	llh_read( )	lld_read( ) .
m_ record to p _ record	Ilh_p_prep( )	lld_p _prep( )
p_ record to s_ record	llh_display( )	lld_display( )
From Input Program to Database		
s_ record to p_ record	Ilh_input( )	lld_input( )
p_ record to m_ record	llh_m_prep( )	lld_m_prep( )

From m_ Record to Database

create a new row	Ilh_add( )	lld_add( )
update a row	llh_update( )	none
delete a row	llh_delete( )	lld_delete( )

Data input and display as it is associated with lowlevel functions



I/O Triggers

There are several useful triggers that are involved with the p_prep and m_prep functions. The following shows some of the triggers that insert code into the llh* and lld* functions shown on the previous page.

Trigger	Use
`on_disk_read`	Inserts code just after the SQL select loads the m_record.
`on_disk_add`	Inserts code just after m_record variables are inserted into the table.
`on_disk_update`	Inserts code just after a record is updated.
`on_disk_delete`	Inserts code just after a record is deleted.
`on_disk_record_prep`	Inserts code just after the m_record is loaded with p_record values.
`on_screen_record_prep`	Inserts code just after the p_record is loaded with m_record values.

Referencing Input Fields in Triggers

Frequently, you want to manipulate data in fields. You can do so with various triggers. When you reference an input field, though, you must always qualify it with its p_ record name. For example, illustrates an affect field trigger with an incorrect input field reference:

```
after_field company
    if company is null
    then
           CALL gn_close(“Error”, "You must fill in the company
        field")
    end if ;
```

Instead, you must qualify input fields with there p_ record name. This example shows a correctly referenced input field:

```
after_field company
    if p_stomer.company is null
    then
         CALL gn_close(“Error”, "You must fill in the company
    end if ;
```

In this case, `p_stomer.company` is the name of the p_ record that coincides with the company field.

The p_ record name is always found in globals.4gl. Field names are found in the globals.4g1 file as well or in the form specification (*.per) file under the ATIRIBUTES section.

To reference table columns, you must qualify the column name with its m_ record.

Common Global Variables

The Code Generator always creates a common set of variables in your globa1s.4g1 file. These variables, which can also be referenced in triggers, are very useful. You can find these variables under the Library communications section of your globa1s.4gl file.

```
###############################################################
# Library communication area 5.40.01.01
###############################################################
# Global variables in this section should not be changed.
# They are used to communicate to the screen library functions,
# and must be of the same type as defined in the library.
# Don't remove these comments.  The codegenerator keys on them.
#
progid       char(17),   # Program identification
scr_id       char(8),    # Current screen id
menu_item    char(10),   # Current menu item running
scr_funct    char(20),   # Current screen function being run
sql_filter   char(512),  # Filter portion of SQL statement
sql_order    char(100),  # Order portion of SQL statement
input_num    smallint,   # Current input section within screen
p_cur        smallint,   # Current input array element
s_cur        smallint,   # Current screen array element
scr_fld      char(40),   # Current screen field
nxt_fld      char(40),   # Programmatic next screen field
prev_data    char(80),   # Data before field entry
this_data    char(80),   # Data after field entry
data_changed smallint,   # Has the field data changed?
hotkey       smallint,   # The hot key that has been pressed
wnMain       ui.Window,  # Genero - ui.window
fmMain       ui.Form,    # Genero - ui.Form
scratch      char(2047)  # Scratchpad for scribbling on and
                         # communicating between functions
# End library communication area
###############################################################
```

Warning: Never change the definition of these variables at the program level.

Using the Scratch Variable

The `scratch` variable is used as a *scratch pad* for temporary data values. It is used throughout generated code.

Quite frequently, `scratch` is used by VDT generated code for passing character type data between functions, such as SQL statements, messages, table names and column names.

Warning: Avoid using the scratch variable because you may overwrite information needed by the generated code.

Section Summary

- Input program data gets passed through the program code by way of records. In all, there are four records that the generator creates: the p_ record, m_ record, s_ record, and q_ record.

- The s_ record reflects the actual values displayed by the input program. The p_ record is formatted to parallel the input program fields. The q_ record contains table values not used by the input program. The m_ record parallels the columns in the database table.

- Two functions convert the m_ record to the p_ record and vice versa. These functions, known as `p_prep` and `m_prep`, control the mapping between the table columns and the program input fields.

- Data movement outside the program occurs all at once. Data values are accepted into a program from the screen *en masse* by the input command. Values are displayed to the screen all at once by the display command. The same holds true for inserts and most selects.

- Several lowlevel functions control the flow of data between the database, m_ record, p_ record, and input program. There are several I/O triggers that let you add custom logic to these functions.

- When you reference a column or input field in a trigger, you must preface it with its record type. For example, the `1name` field in the `trcustomer` table, when called in a trigger, should be referenced as `p_stomer.lname`.

- There are a variety of useful variables that are always generated in the `globa1s.4g1` file.

Exercise 10A

Objective: To reference a field on the screen and perform error-checking logic on that field.

You will reference a p_ record variable and use an `after_field` trigger to perform validation. The error-checking logic that you create will require the user to supply a phone number when entering a new customer record in the Customer Entry program.

Add a Trigger

Your trigger will test for a null value in the Phone Number field.

1. **Start the VDT Form Designer in your i_cust. 4gs directory.**

2. **Open the main Customer Entry form (cust) in the VDT Form Designer.**

3. **Move to the pull-down menus and select Triggers » from the Define pull-down menu.**

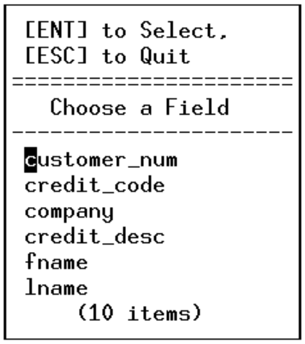
   The Choose a Trigger Class box appears.

4. **Since your Phone Number field is in the header section, select Input Area 1 from the Choose a Trigger Class box.**

   The Choose a Trigger list box appears. Because you want check a field for a null value, you want to evaluate the field once the user has moved past it. You want to use an **`after_field`** trigger.

5. **Select `after_field` from the Choose a Trigger list box.**

   The Choose a Field list box appears.

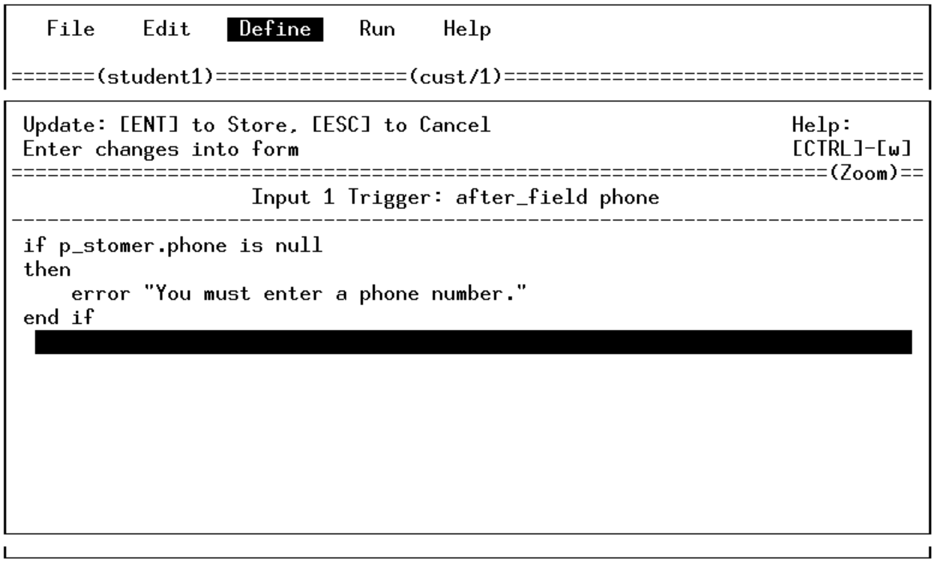


```
 [ENT] to Select,
 [ESC] to Quit
======================
   Choose a Field
----------------------
 customer_num
 credit_code
 company
 credit_desc
 fname
 lname
     (10 items)
```

6. **Select phone from the Choose a Field list box.**

   The editing window appears.

7. **In the editing window, add the following custom logic:**



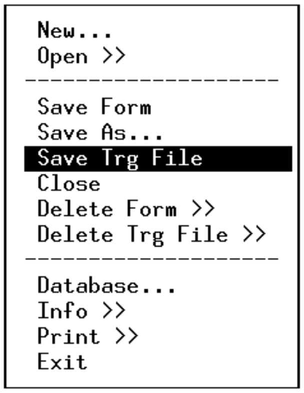
```
  File    Edit    Define    Run    Help

=======(student1)================(cust/1)===================================

 Update: [ENT] to Store, [ESC] to Cancel                          Help:
 Enter changes into form                                          [CTRL]-[w]
===================================================================(Zoom)==
                    Input 1 Trigger: after_field phone
---------------------------------------------------------------------------
 if p_stomer.phone is null
 then
     error "You must enter a phone number."
 end if
```

Important

Since you are referencing a field on the screen, the field name in your logic must be qualified with its p_ record. If this is not done, a syntax error occurs.

8. **Press [ESC] to save your custom logic then [CTRL]-[c] to close the Choose a Trigger list box.**

9. **Select Save Trg File from the File pull-down menu. This option writes your trigger logic into a trigger (*.trg) file.**



Compile the Code

- **Select Compile 4GL from the Run pull-down menu.**

The compilation utility calls the Featurizer. The Featurizer reads the trigger (*.trg) file and merges the `after_field` logic into the generated source code.

Run the Customer Entry Program

- **Select Run 4GL Program from the Run pull-down menu.**

  The Customer Entry program starts.

Test the after_field Trigger

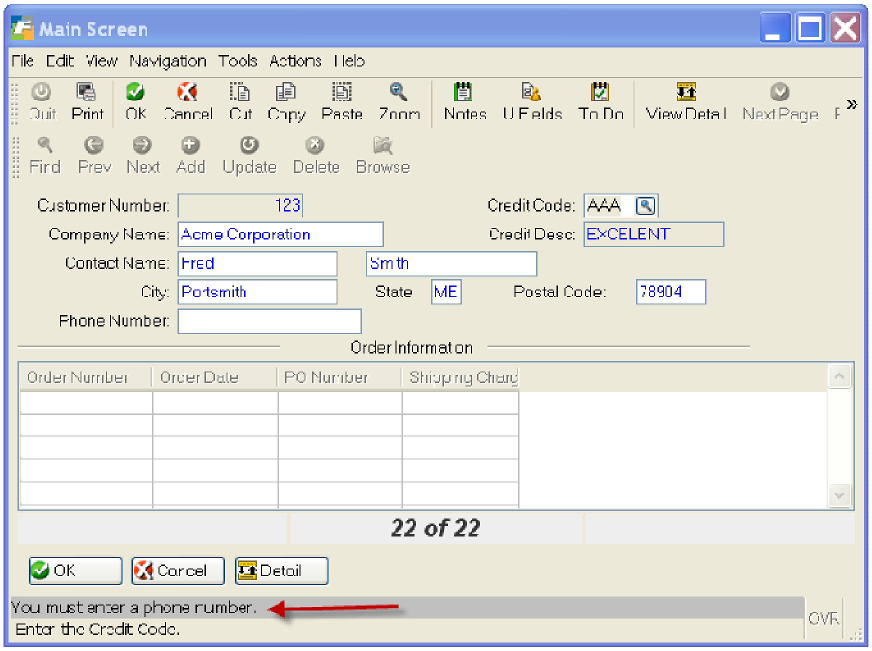
1. **Select Add from the ring menu.**

   The logic that you wrote in Exercise 9 appears.

2. **Enter data into the fields preceding the Phone Number field.**

3. **Leave the Phone Number field blank and press [ENTER].**

   Your error message appears at the bottom of the screen and your cursor moves to the Credit Code field.


Main Screen
File Edit View Navigation Tools Actions Help
Quit Print OK Cancel Cut Copy Paste Zoom Notes U.Fields To Do View Detail Next Page
Find Prev Next Add Update Delete Browse
Order Information
Order Number
Order Date
PO Number
Shipping Charg
OK
Cancel
Detail
OVR

Customer Number: 123
Company Name: Acme Corporation
Contact Name: Fred Smith
City: Portsmith State ME Postal Code: 78904
Phone Number:
Credit Code: AAA
Credit Desc: EXCELENT

22 of 22

You must enter a phone number.
Enter the Credit Code.

This result is not entirely desirable. The error message works great, but you also must control the cursor movement. As it stands, you can save a record without entering a phone number.

4. **Press [ENTER] to save this record and Quit to return to the VDT Form Designer.**

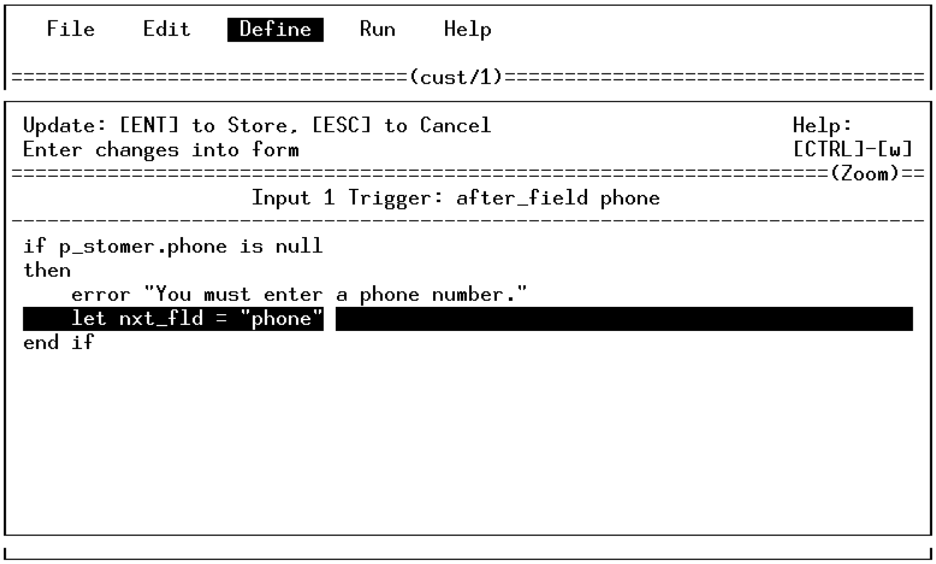
Modify the after_field Trigger

You can use the `nxt_fld` global variable in your trigger to control the condition on which the cursor can move to the next field.

1. **Return to the trigger editing window:**

Select	From
Triggers>>	The Define pull-down menu.
Input Area 1	The Choose a Trigger Class box.
After-field	The Choose a Trigger list box.
Phone	The Choose a Field list box.

2. **Modify your trigger code to look as follows:**



```
================================(cust/1)==================================
Update: [ENT] to Store, [ESC] to Cancel                          Help:
Enter changes into form                                          [CTRL]-[w]
=================================================================(Zoom)==
                    Input 1 Trigger: after_field phone
--------------------------------------------------------------------------
if p_stomer.phone is null
then
    error "You must enter a phone number."
    let nxt_fld = "phone"
end if
```

5. **Press [ESC] to save your custom logic then [CTRL]-[c] to close the Choose a Trigger list box.**

6. **Select Save Trg File from the File pull-down menu.**

Compile and Run

1. **Select Compile 4GL from the Run pull-down menu.**

2. **Select Run 4GL Program from the Run pull-down menu.**

   The Customer Entry program starts.

Test the after_field Trigger

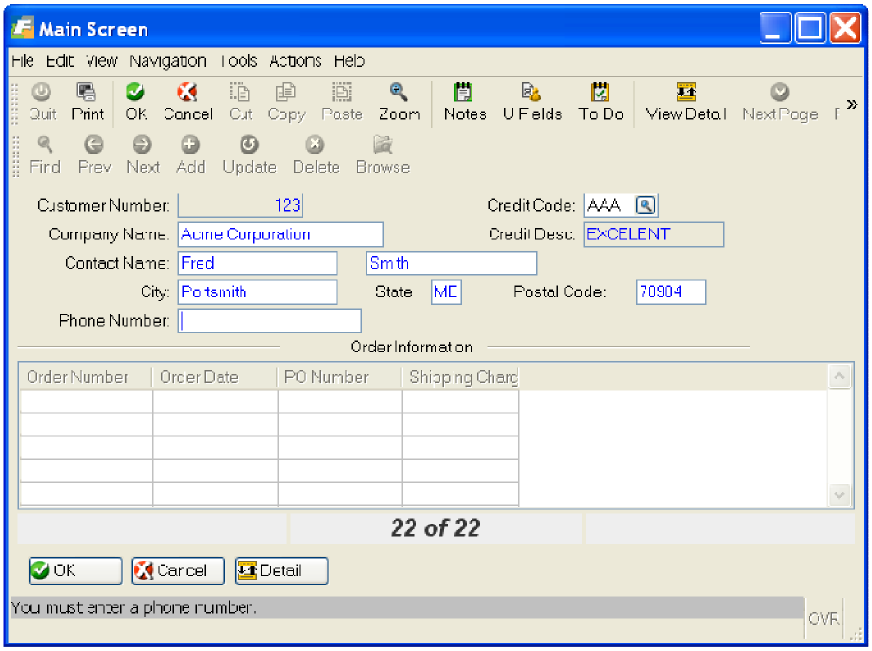
1. **Select Add from the ring menu.**

   The before_input logic that you wrote in Exercise 9 appears.

2. **Enter data into the fields preceding the Phone Number field.**

3. **Leave the Phone Number field blank and press [ENTER].**

   This time the error message appears and your cursor remains trapped in the Phone Number field until you add a value.


Main Screen
File Edit View Navigation Tools Actions Help
Quit Print OK Cancel Cut Copy Paste Zoom Notes U Fields To Do View Detail Next Page
Find Prev Next Add Update Delete Browse
OK Cancel Detail
OVR

Customer Number:	123	Credit Code:	AAA
Company Name:	Acme Corporation	Credit Desc:	EXCELENT
Contact Name:	Fred Smith		
City:	Potsmith	State: ME	Postal Code: 70904
Phone Number:			

Order Information

Order Number	Order Date	PO Number	Shipping Charg

22 of 22

You must enter a phone number.

4. **Add a phone number, press [ENTER], and then Quit to return to the VDT Form Designer.**

Remove a Trigger

By now you're tired of seeing the `before_input` logic you wrote in Exercise 9. You can remove this logic as simply as you added it.

1. **Move to the trigger editing window:**

   Select **From**

Triggers>>	The Define pull-down menu.
Input Area 1	The Choose a Trigger Class box.
before_input	The Choose a Trigger list box.

2. **Delete both lines of the before_input trigger.**

   You can delete a line quickly by pressing [CTRL]-[d].

3. **Press [ESC] to save your deletion then [CTRL]-[c] to close the Choose a Trigger list box.**

4. **Select Save Trg File from the File pull-down menu.**

   The before_input logic is removed.

Note

The File pull-down menu also has a Delete Trg File » option. In this case you do not want to delete a trigger file because both your `after_field` and `before_input` triggers were in the same file. Use the Delete Trg File >>> option when you want to remove **ALL** the triggers in that file.

Compile and Run

1. **Select Compile 4GL from the Run pull~down menu.**

2. **Select Run 4GL Program from the Run pull~down menu.**

   The Customer Entry program starts.

3. **Press Add to verify that the before_input logic has been removed.**

4. **Once you have proven this to yourself, remain in the Customer Entry program and continue to Exercise 10B.**

Exercise 10B

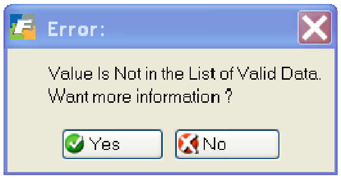
Objective: To replace the error statement with Fitrix's `fg_err` function. This function lets you write custom error messages.

Test the after_field Trigger

1. **From the Customer Entry program, select Add.**

2. **Enter an invalid value in the Credit Code field (TTT).**

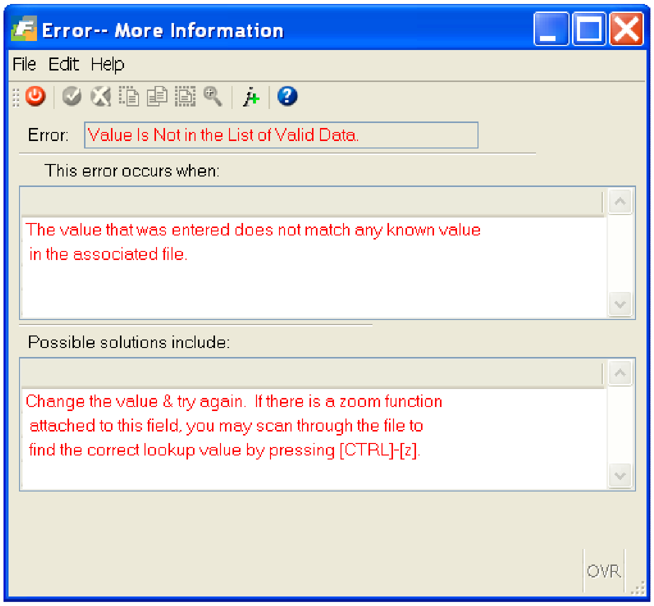
An error message appears informing you that the value is not in the list of valid data. This message also includes the ability to see more information about the error.



Value Is Not in the List of Valid Data.
Want more information ?

3. **Press [Y] to see additional error information.**

An error window appears.



Error: Value Is Not in the List of Valid Data.

This error occurs when:

The value that was entered does not match any known value in the associated file.

Possible solutions include:

Change the value & try again. If there is a zoom function attached to this field, you may scan through the file to find the correct lookup value by pressing [CTRL]-[z].

In this exercise, you will call a similar error window when the user leaves the Phone Number field empty (null).

Create Error Text

1. **The header portion of an error message is stored in the stxerorh table. Use dbaccess to insert your new error message into the stxerorh table in your student database. The sql to do this is below. Also save this sql as stxerorh.sql in the $fg/data/sql.4gc directory.**

```
insert into stxerorh
    (language, userdef, err_module, err_program, err_number,
err_line)
       values
    ("ENG","","aw","i_cust", 20, "You must enter a phone number.");
```

2. **The detail portion of an error message is stored in the stxerord table. Use dbaccess to insert your new error message into the stxerord table. The sql to do this is below. Also save this sql as stxerord.sql in the $fg/data/sql.4gc directory.**

```
insert into stxerord
   (language, userdef, err_module, err_program, err_number, a_b,
line_no, err_text )
   values
   ("ENG","", "aw", "i_cust", 20, "a", 1,
       "The phone number field requires a phone number.");
insert into stxerord
   (language, userdef, err_module, err_program, err_number, a_b,
line_no, err_text )
   values
   ("ENG","", "aw", "i_cust", 20, "b", 1,
      "Enter a phone number value.");
```

Note:

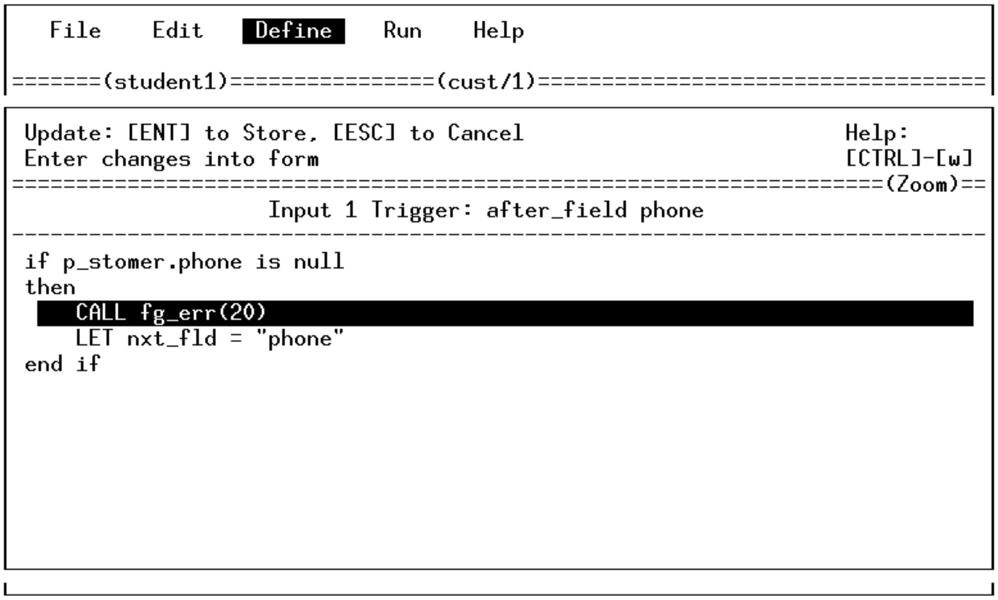
If a stxerorh.sql or stxerord.sql file had already existed in your $fg/data/sql.4gc directory you add to it instead of overwrite it.

Add a Call to fg_err in Your after_field Trigger

1. **Return to the trigger editing window:**

Select	**From**
Triggers>>	The Define pull-down menu.
Input Area 1	The Choose a Trigger Class box.
after_field	The Choose a Trigger list box.
phone	The Choose a Field list box.

2. **Replace the error line with a call to fg_err:**



```
=======(student1)================(cust/1)==================================
Update: [ENT] to Store, [ESC] to Cancel                            Help:
Enter changes into form                                            [CTRL]-[w]
================================================================(Zoom)==
                    Input 1 Trigger: after_field phone
-------------------------------------------------------------------------
if p_stomer.phone is null
then
    CALL fg_err(20)
    LET nxt_fld = "phone"
end if
```

3. **Return to the pull-down menus and select Save Trg File » from the File pull-down.**

Compile, Run, and Test

1. **Select Compile 4GL from the Run pull-down menu.**

2. **Select Run 4GL Program from the Run pull-down menu.**

   The Customer Entry program starts.

3. **Leave the Phone Number field empty to see what happens.**

4. **Remain in the Customer Entry program and continue to Exercise 10C.**

Exercise 10C

Objective: To require input in the Credit Code field.

You will build an `after_input` trigger that requires the user to enter a value in this field before the record can be saved.

Examine the Credit Code Field

1. **From the Customer Entry program, select Add.**

2. **Press [TAB] to move past the Credit Code field.**

3. **Complete the record and press [ENTER].**

   Notice how the program accepts this record without a value in the Credit Code field.

4. **Quit from the Customer Entry program and the VDT Form Designer.**

Examine the cust.trg File

1. **Use vi to open the cust . trg file.**

```
######################################################################
#
# Copyright (C) 1996-2010 Fourth Generation Software Solutions Corp.
# Atlanta, GA
# All rights reserved.
# Use, modification, duplication, and/or distribution of this
# software is limited by the software license agreement.
# Sccsid:  %Z%  %M%  %I%  Delta: %G%
######################################################################
#
# Form Painter version: 5.30.01.01 build: 1017
input 1

    after_field phone
        if p_stomer.phone is null
        then
            CALL fg_err(20)
            LET nxt_fld = "phone"
        end if;
```

To this file, you will add your `after_input` trigger. It is important to note that you can create triggers by hand using vi. You do not need to build them using the VDT Form Designer, although the VDT Form Designer makes it easier.

2. **Below the last line (end if;) add the following custom logic:**

```
after_input
   if p_stomer.credit_code is null
   then
          CALL fg_err(21)
          let nxt_fld = “credit_code"
   end if;
```

3. **Save and exit Cust.trg.**

4. **Add the new error message to your student database using the following sql. Remember to add this sql to $fg/data/sql.4gc/stxerorh.sql.**

```
insert into stxerorh
```

```
    (language, userdef, err_module, err_program, err_number,
err_line)
     values
    ("ENG","","aw","i_cust",21, "You must enter a Credit Code.");
```

Merge Your New Trigger Logic

- **At the LINUX prompt, run fg.make:**

  ```
  fg.make
  ```

  Remember that `fg.make` (which is analogous to the Compile 4GL Code option under the VDT Form Designer's Run pull-down menu) calls the Featurizer. The Featurizer is the utility that merges trigger (*.trg) files into 4GL source code files.

Run the Customer Entry Program

1. **• Run the Customer Entry program:**

   ```
   fglrun i_cust.42r
   ```

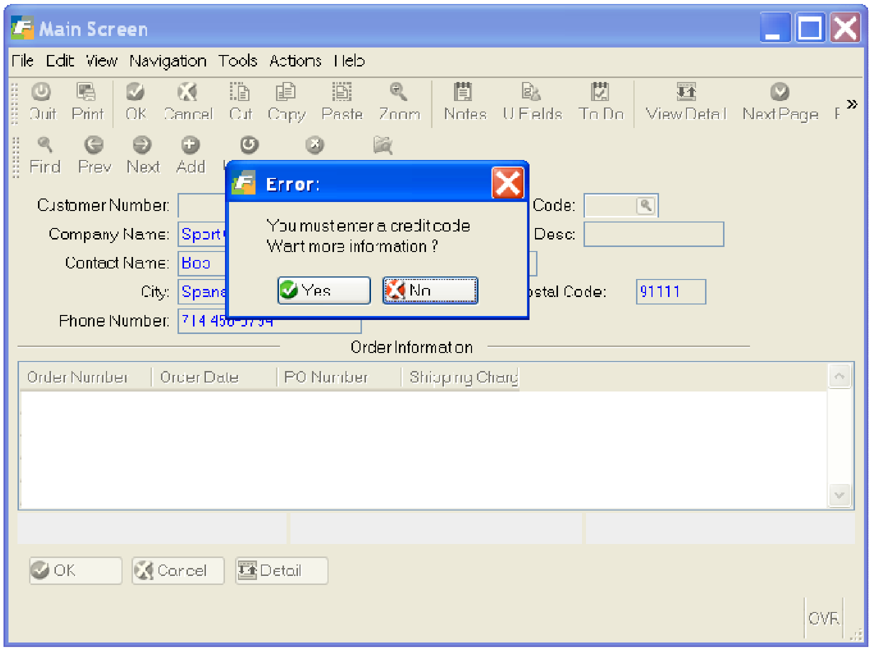
The Customer Entry program starts.

Test your after_input Logic

1. **Select Add from the ring menu.**

2. **Fill in all the fields except the Credit Code field and press [ENTER].**

   Your error message appears and the cursor moves to the Credit Code field.


Main Screen
File Edit View Navigation Tools Actions Help
Quit Print OK Cancel Cut Copy Paste Zoom Notes U Fields To Do View Detail Next Page
Find Prev Next Add
Order Information
Order Number
Order Date
PO Number
Shipping Charg
OK
Cancel
Detail
OVR

Error:

You must enter a credit code
Want more information ?

Customer Number:
Company Name: Sport
Contact Name: Boo
City: Spana
Phone Number: 714 456-5754
Code:
Desc:
Postal Code: 91111

You cannot save this record until you enter a value in the Credit Code field.

3. **Enter a Credit Code value, save this record, and press Quit to return to the LINUX command line.**

Chapter 11

Screen Handling and Add-on Headers

Main topics:

- Using Different Screen Types
- The socketManager Function
- Linking in Add-On Screens

Using Different Screen Types

In chapter two, you learned about the different screen types you can build using the VDT Form Designer. These screen types are classified into three groups:

1. Main Screens

2. Secondary Screens

3. Auxiliary Screens

Main Screens

A main window constitutes the main part of your input program. There are two main screen types: header and header / detail screens.

header: This is a flat type. Header screens contain one input area and one main table.

header/detail: This is a flat type (header) with another scrolling (detail) section joined to the header. Header / detail screens are suited for order forms where there is one occurrence for customer information and multiple line items for merchandise.

Secondary Screens

Secondary screens are not used as stand-alone data-entry screens. Instead, they are called from the main screen. There are four secondary screen types: add-ons, extension, query, and view.

add-on header: This is a header screen used in conjunction with another header or header/ detail screen to provide an extra window of fields. This screen type generates disk read and write functions.

add-on detail: This is a scrolling detail-only screen. This screen can be called from any other screen to display detail information. This screen type generates disk read and write functions.

extension: This is a special type of screen that enables you to include an extension of the main header table or detail table. This screen type shares data with the main screen.

query: this screen is used for building an SQL query. It can replace the mlh_construct function.

view-detail: This is a detail-only screen that allows you to view data but not alter it.

view-header: This is a flat screen used to view header information.

Auxiliary Screens

Auxiliary screens are unlike any other screen type. These types are used in conjunction with the main screen and are basically used to locate and select information.

browse: This is a scrolling type screen. Its main table is the same as the header section main table. A browse screen enables you to view one row of the header table per line rather than one row per screen. Only one browse screen can be used per program.

zoom: This is a special type of screen that enables you to view and/or retrieve data from another table (or set of tables which are "joined").

Linking Different Screen Types to the Main Screen

You can divide the input program creation process into two main tasks: painting the form images and linking screens together. This chapter shows you how to create and link add-on header screens.

In an earlier chapter, you learned how to link in zoom screens, and in later chapters you will learn how to link in other secondary screens.

Linking in an add-on header screen requires you to create a special trigger file. You call the `socketManager` function from within this file.

The socketManager Function

The `socketManager` function controls which code block or flow different screen types use. For every screen type, there exists default library code that is processed when that screen type gets initiated. When you link secondary screens to your main screen, you must use the socketManager function to call the library code associated with your secondary screen type.

The `socketManager` function syntax looks as follows:

```
socketManager("screen_name", "screen_type", flow")
```

`screen_name` This argument represents the form specification (*.per) file less the .per extension.

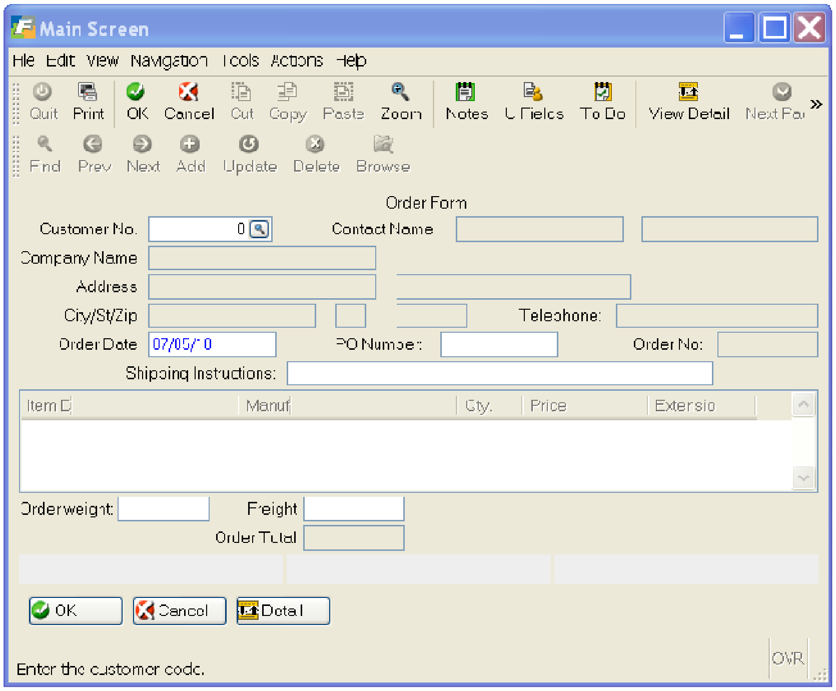
`screen_type` This argument represents the screen type. Valid screen types include: add-on header, add-on detail, extension, query, view header, and view detail.

`flow` Flow indicates a default block of library code associated with each screen types. In most cases, the flow is default. Extension screen types, however, require you to specify between one of three screen types: flat_ext, deep_ext, and view.

Designing Add-On Header Screens

Add-on header screens provide input fields to an additional table. Many times, you may want users to add data to this table during the data-entry process. While inputting orders a user might come across an order from a new customer. When the Customer No. field is assigned a zero, an add-on header screen appears, and the program user can enter information about the new customer before entering that customer's order.

A value of zero in the Customer No. field triggers an add-on header screen



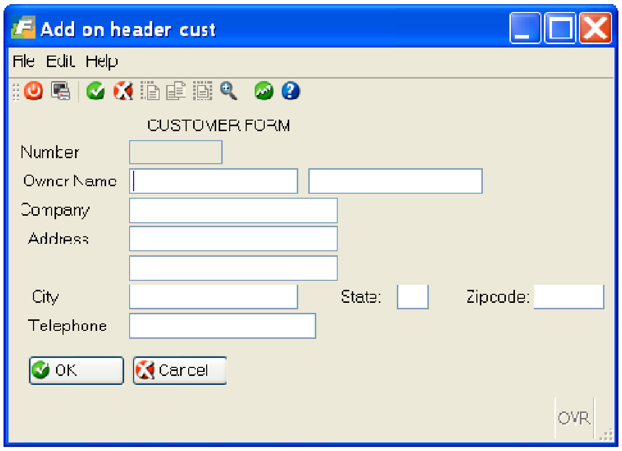
Customer No. 0

Order Date 07/05/10

Enter the customer code.

.

The program user can quickly enter information about a new customer, in an add-on header screen, before resuming order entry.



Building Add-On Header Screens

To build an add-on header, use the VDT Form Designer to create the form image (select add-on header as the screen type). After you define the form image, save it to a form specification (*.per) file.

Linking in Add-On Header Screens

To link in your add-on header, you must create a trigger file that contains both the `switchbox_items` trigger and one or more initiating event triggers, such as an `after_field` trigger. For more on trigger files, see "Creating Triggers" on page 150.

You can use either the VDT Form Designer or a text editor to create this trigger file. For it to work correctly, you must specify four pieces of information:

1. The name of the add-on header file, less the .per extension.

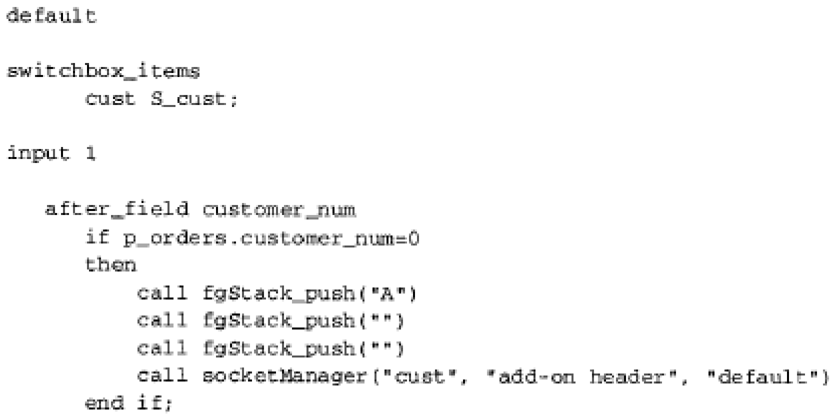
2. The trigger or event that initiates the add-on header screen. For example, the add-on header discussed on the previous page was initiated when the Customer No. field contained a value of zero.

3. The condition in which the add-on header screen is called. You specify condition settings with the **`fgStack_push`** function. All add-on header screens require you to set three attributes with the **`fgStack_push`** function: mode, filter, and order by.

4. The **`socketManager`** function.

In addition, your trigger file should be named after the main screen from which the add-on header gets called. For example, if the main screen is defined in the `order.per` specification file, the trigger file where you link your add-on header should be named `order.trg`.

The following code illustrates a `switchbox_items` trigger and an `after_field` trigger. Together these triggers specify all the information necessary to link in the `cust.per` add-on header screen.



```
default

switchbox_items
      cust S_cust;

input 1

   after_field customer_num
      if p_orders.customer_num=0
      then
          call fgStack_push("A")
          call fgStack_push("")
          call fgStack_push("")
          call socketManager("cust", "add-on header", "default")
      end if;
```

switch box_items trigger

after_field trigger

three calls to fgStack_push

a call to socket_manager

Using the example, you can see where each piece of information necessary to link in the add-on header screen gets supplied.

The `default` section contains the `switchbox_items` trigger. This trigger requires two arguments: the add-on header form specification file name (less the *.per extension) and the screen function. (The screen function name is always an s_ followed by the form specification file name.)

```
default

   switchbox_items
         cust S_cust;
```

The `input 1` section contains the trigger or event that initiates the add-on header screen. In the example, an `after_field` trigger initiates the add-on header screen.

```
input 1

   after_field customer_um
```

In addition, the `input 1` section contains the `fgStack_push` function, which sets add-on header conditions. For add-on header screens, you need to call this function three times. Even if you do not want to set some of these conditions, you still must pass this function three times passing null values for the conditions you do not want to set.

The first call indicates the mode that the add-on header screen starts in. An A indicates add mode. You can also specify a “U” for update mode.

```
call fgStack_push("A")
```

The second call indicates the selection filter. If you are opening your add-on header in update mode, you can pass it a filter indicating which records you want updated.

```
call fgStack_push("")
```

The last call relates to both update mode and the filter you specify. It constitutes an order by clause. If your filter selects multiple records, you can order those records by the criteria you specify in the third fgStack-push function call.

```
call fgStack_push("")
```

Finally, this section calls the socketManager function, which designates the correct flow for your add-on header screen.

```
call socketManager("cust", "add-on header", "default")
```

Section Summary

- You build input programs based on many different screen types. Each type has its own function.
- In all there are ten screen types. These ten types can be classified into three groups: main, secondary, and auxiliary.
- When you build input programs you must first create the form images and then link these images together using the `socketManager` function.
- The `socketManager` function controls which code block or flow different screen types use. For every screen type, there exists default library code that is processed when that screen type gets initiated. When you link secondary screens to your main screen, you must use the `socketManager` function to call the library code associated with your secondary screen type.
- Add-on header screens provide input fields to an additional table. Many times, you may want users to add data to this table during the data-entry process.
- To build an add-on header, use the VDT Form Designer to create the form image (select add-on header as the screen type). After you define the form image, save it to a form specification (*.per) file.
- To link in your add-on header, you must create a trigger file that contains both the `switch-box_iterms` trigger and one or more initiating event triggers, such as an `after_field` trigger.

Exercise 11 A

Objective: To become familiar with add-on header screens.

Add-on header screens provide additional data-entry screens that can be incorporated into your input programs. These screens write to tables other than the header or detail table.

Recall that in Exercise 3, you created the Credit Entry program. You later built a hot key to initiate this program from within the Customer Entry program. Add-on header screens provide much the same functionality, but they are further integrated into your base program.

Run scr_demo 5

The screen demonstration five program shows a good example of an add-on header screen.

1. **At the LINUX prompt type:**

```
scr_train 5
```

2. **From the Training prompt compile, generate, and run:**

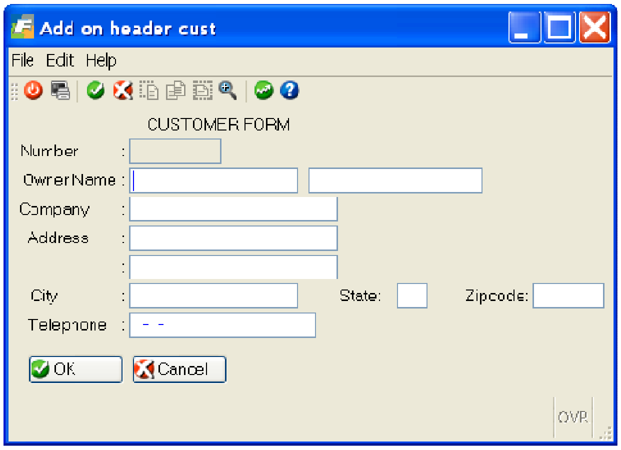
```
fg.screen -o0 -y

fg.make

fglrun screen  *.42r
```

The training 5 program starts.

Add a Customer

1. **Select Add and enter 0 into the Customer Number field.**

   An add-on header screen appears which looks similar to your Customer Entry program.

   This screen lets you add another customer record to the customer table.

2. **Fill in the Customer Form and press [ENTER].**

   You've just added a new customer on the fly. Notice how back on the Order Form, the new customer number is returned and placed in the Customer Number field**.**

3. **Complete the Order Form and press [ENTER] to save it**

Exercise 11 B

Objective: To create and use your own add-on header screen.

This add-on header will let users enter sales representatives to a new table from within the Customer Entry program.

To complete this exercise, you must perform the following major steps:

1. **Add a column named `sales_code` to the `customer` table.**

2. **Add a Sales Code field to your Customer Entry program.**

3. **Create a new table called `trsalesrep`.**

4. **Create an add-on header screen based on the `trsalesrep` table.**

5. **Incorporate this screen into your Customer Entry program.**

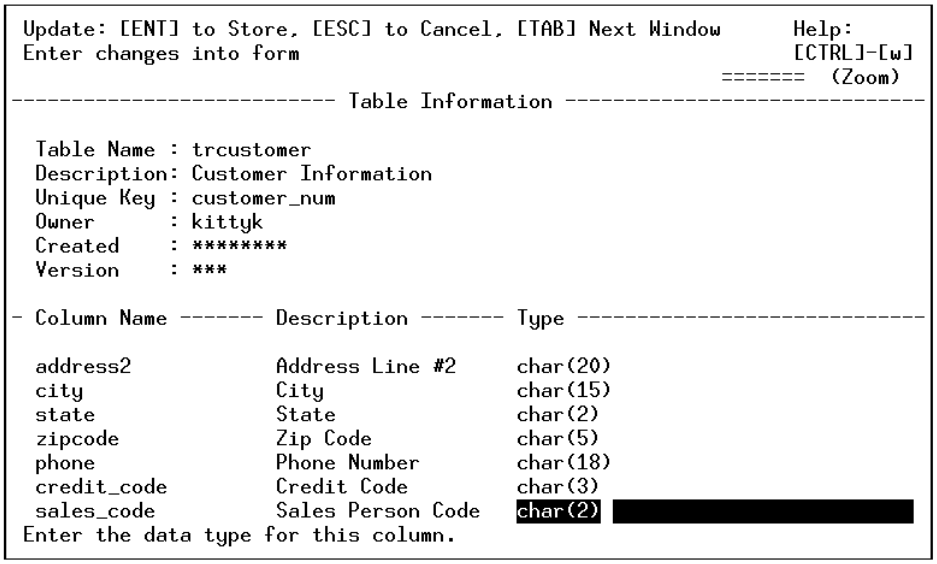
Add a Column

If you haven't done so already, move to your `i_cust.4gs` directory.

1. **Start the VDT Form Designer and select Database from the File menu.**

   The Database option, as you recall, lets you change the structure of your database. You can add, delete, and alter the columns in a table.

2. **Find the trcustomer table and add the `sales_code` column.**



```
 Update: [ENT] to Store, [ESC] to Cancel, [TAB] Next Window      Help:
 Enter changes into form                                          [CTRL]-[w]
                                                            =======  (Zoom)
--------------------------- Table Information ------------------------------

  Table Name : trcustomer
  Description: Customer Information
  Unique Key : customer_num
  Owner      : kittyk
  Created    : ********
  Version    : ***

- Column Name ------- Description ------- Type -----------------------------

  address2            Address Line #2     char(20)
  city                City                char(15)
  state               State               char(2)
  zipcode             Zip Code            char(5)
  phone               Phone Number        char(18)
  credit_code         Credit Code         char(3)
  sales_code          Sales Person Code   char(2)
 Enter the data type for this column.
```

3. **Save this addition and press Quit to return to the VDT Form Designer. Exit the VDT Form Designer**

4. **Copy your database change to the central database changes directory. You can copy the entire dbadmin.sql file because changes are added to dbadmin.sql in the local program directory.**

   cp dbadmin.sql $fg/data/sql.4gc/trcustomer.sql

Add a Field

1. **Return to the VDT Form Designer, open your `cust` form file.**

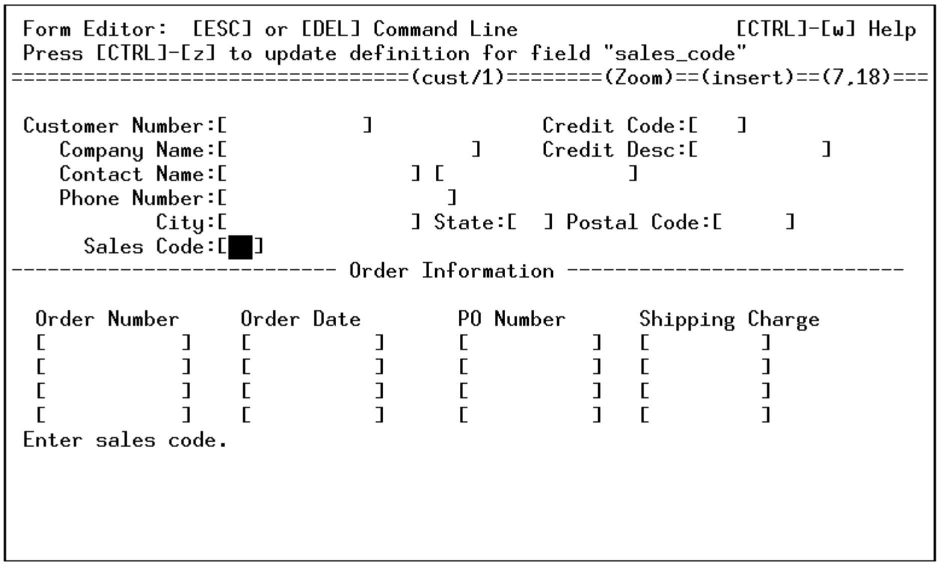
2. **Add the Sales Code field to your Customer Entry form.**

   Probably the best location for this field is just above the Order Information detail section. Use

Mark, Cut, and Paste to return the Phone Number field to its original location (below the Contact Name field). Then add the Sales Code field. Define this field using the following settings:

Table Name	**trcustomer**
Column Name	sales_code
Input Area	1
Entry ?:	Y
Message:	Enter sales code

When you're finished, your form should look as follows:



```
 Form Editor:  [ESC] or [DEL] Command Line                  [CTRL]-[w] Help
 Press [CTRL]-[z] to update definition for field "sales_code"
=================================(cust/1)========(Zoom)==(insert)==(7,18)===

 Customer Number:[           ]             Credit Code:[   ]
    Company Name:[                    ]    Credit Desc:[          ]
    Contact Name:[               ] [                ]
    Phone Number:[                  ]
            City:[               ] State:[  ] Postal Code:[     ]
      Sales Code:[█]
--------------------------- Order Information ---------------------------

  Order Number     Order Date        PO Number      Shipping Charge
  [          ]    [          ]      [          ]   [         ]
  [          ]    [          ]      [          ]   [         ]
  [          ]    [          ]      [          ]   [         ]
  [          ]    [          ]      [          ]   [         ]
 Enter sales code.
```

3. Select Save Form from the File pull-down to save this change.

Function of an Add-On Screen

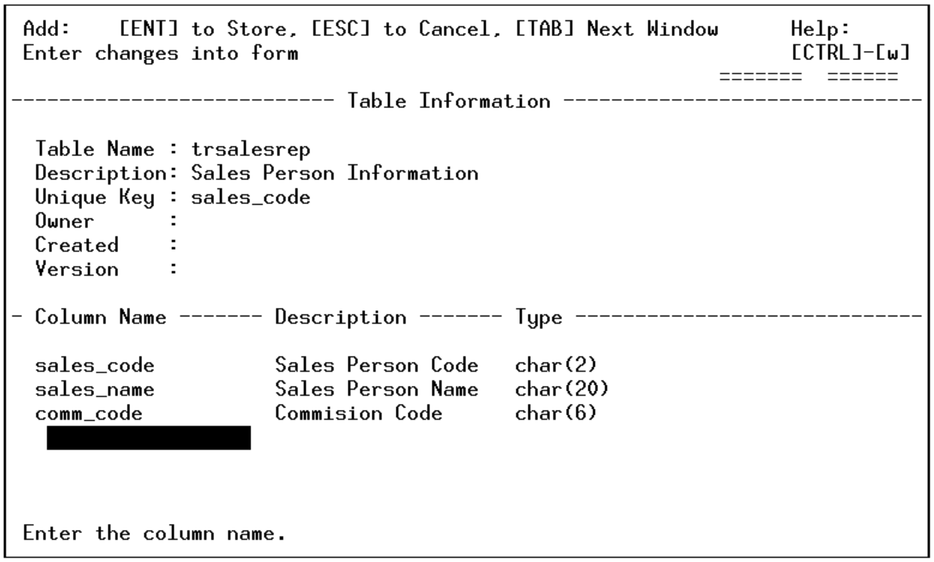
At this point, you could rebuild your Customer Entry program and start entering a sales person code for each customer. But this would simply be meaningless data; you could enter any characters into this field, none of which would stand for anything useful.

A better approach is to build an add-on screen based on a separate table. This table can contain information that is relevant to the sales code. You could add informative columns to this table, such as the sales person's name and rate of commission.

Add a New Table

Once again select Database from the File pull-down menu.

1. **Select Add from the ring menu and create the following entry:**



```
Add:    [ENT] to Store, [ESC] to Cancel, [TAB] Next Window      Help:
Enter changes into form                                         [CTRL]-[w]
                                                              =======  ======
--------------------------- Table Information ------------------------------

  Table Name : trsalesrep
  Description: Sales Person Information
  Unique Key : sales_code
  Owner      :
  Created    :
  Version    :

- Column Name ------- Description ------- Type -----------------------------

  sales_code          Sales Person Code   char(2)
  sales_name          Sales Person Name   char(20)
  comm_code           Commision Code      char(6)




 Enter the column name.
```

2. **Press [ESC] to save this table, but remain in the Table Information window.**

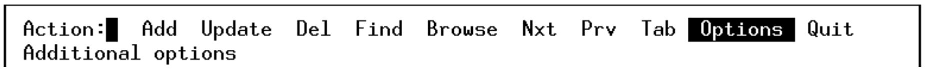
Note

You may receive a Warning message about the Unique Key field. If so, simply press OK to continue.

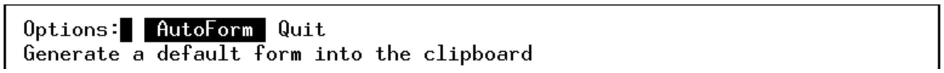
Use AutoForm

Once `salesrep` is built, you can use the AutoForm option to build a default data-entry screen based on `salesrep`.

1. **Select the Options command and then choose AutoForm.**



```
Action:   Add  Update  Del  Find  Browse  Nxt  Prv  Tab  Options  Quit
Additional options
```



```
Options:   AutoForm  Quit
Generate a default form into the clipboard
```

A default entry screen is built and placed on the Clipboard in the VDT Form Designer.

2. **Select Quit from the ring menu to return to the VDT Form Designer.**

Create a New Add-On Header Form

1. **Select New from the VDT Form Designer's File pull-down menu.**

2. **Name the new form "reps."**

3. **Select add-on header as the screen type.**

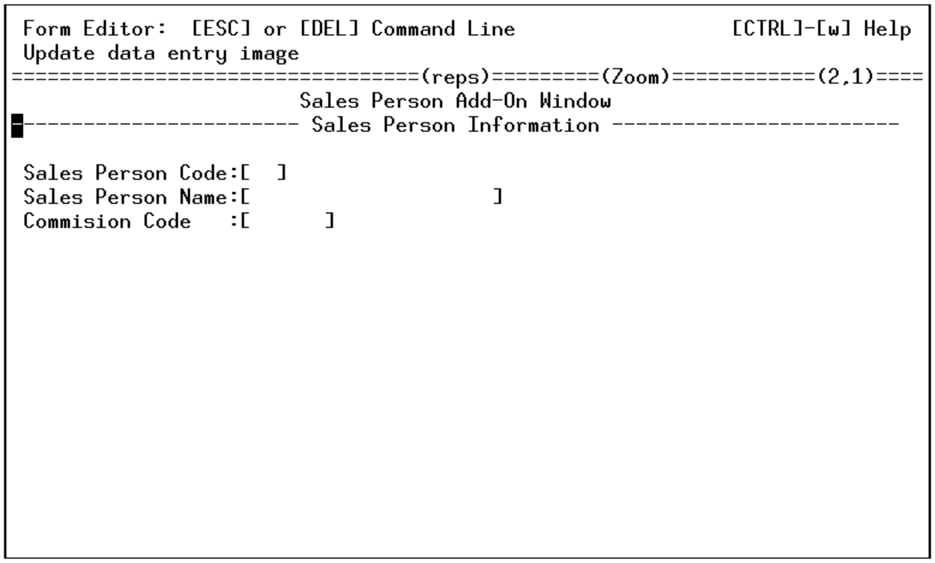
4. **Place the following title centered on the top line of the form:**

   Sales Person Add-On Window

Paste in the AutoForm

Now add the default AutoForn image.

5. **After you add the title line, press [CTRL]-[pl to add the AutoForm image.**



```
 Form Editor:  [ESC] or [DEL] Command Line                [CTRL]-[w] Help
 Update data entry image
================================(reps)=========(Zoom)============(2,1)====
                         Sales Person Add-On Window
█----------------------- Sales Person Information ------------------------

 Sales Person Code:[   ]
 Sales Person Name:[                  ]
 Commision Code   :[      ]
```

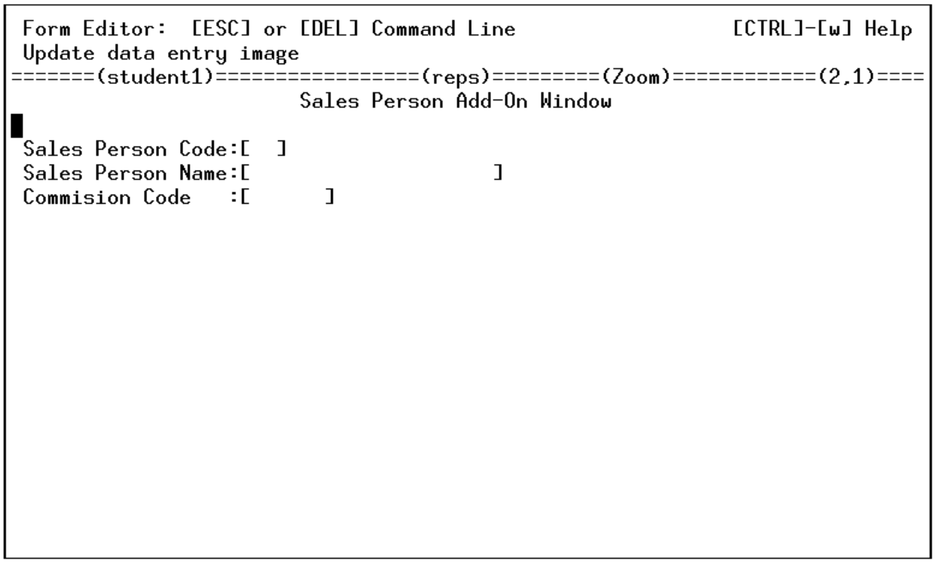
A form built from the `salesrep` table appears. You can use the arrow keys to position in on your screen.

6. **"Tack" the image down by pressing [ESC].**

   As you can see the AutoForm image also contains a title line. You can delete this extra title line with the [F2] key.

7. **Place your cursor on the first character of the extra title line and press [F2].**

8. **When complete, your reps form should look as follows:**



```
 Form Editor:  [ESC] or [DEL] Command Line                 [CTRL]-[w] Help
 Update data entry image
=======(student1)================(reps)=========(Zoom)============(2,1)====
                         Sales Person Add-On Window

 Sales Person Code:[  ]
 Sales Person Name:[                     ]
 Commision Code   :[      ]
```

Generate Code

Once you save your reps add-on form, you can generate code for it.

- **Select Generate 4GL from the Run pull-down menu.**

  At this point, you do not have to compile it.

  Instead, use the VDT Form Designer to reopen your cust form.

Update Genero Schema File and Database Changes

1. **Exit out of the VDT Form Designer. You must update the Genero schema used by the Genero compiler.**

2. **Move to the schema directory.**

   cd $fg/data

3. **Generate the Genero schema file (substitute your database name).**

   sch.sh student1

4. **Save all database changes to the central database changes directory.**

   cd sql.4gc

   cp $HOME/labs/aw.4gm/i_cust.4gs/dbadmin.sql trcustomer.sql

5. **Restart the VDT Form Designer in your i_cust.4gs program directory (substitute your database name).**

cd $HOME/labs/aw.4gm/i_cust.4gs

fg.form -dbname student1

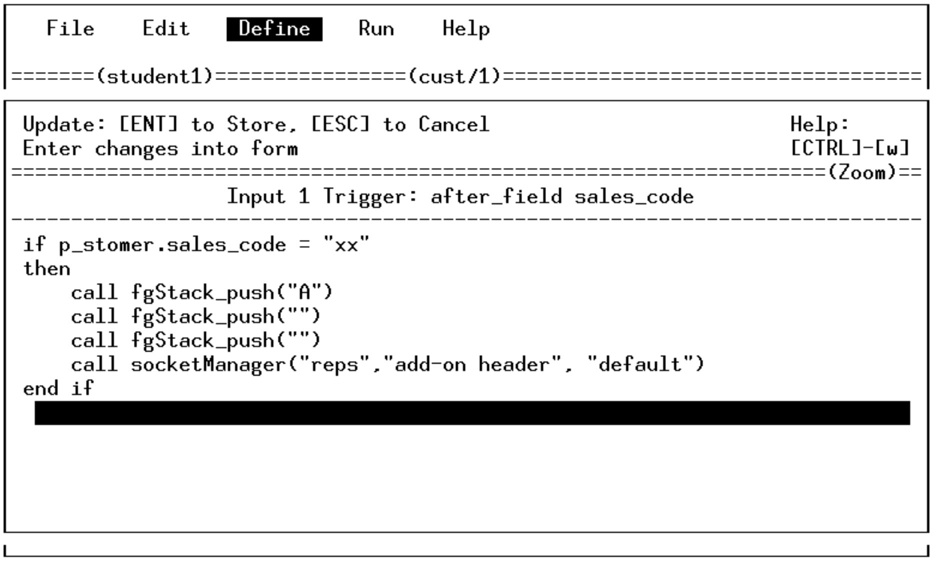
6. **Open your cust form file.**

Incorporate Your reps Add-On

After reps is built, you need to attach it to your to your Customer Entry program. You attach add-on screens using triggers.

For your Customer Entry program, you will build custom logic in an `after_field` trigger. This trigger will evaluate your Sales Code field. When this field contains an `xx` value, it will call your add-on.

1. **In your `cust` form (i.e., your Customer Entry screen), build the following after_field trigger:**



```
=======(student1)================(cust/1)==================================
 Update: [ENT] to Store, [ESC] to Cancel                          Help:
 Enter changes into form                                          [CTRL]-[w]
==================================================================(Zoom)==
                 Input 1 Trigger: after_field sales_code
---------------------------------------------------------------------------
 if p_stomer.sales_code = "xx"
 then
     call fgStack_push("A")
     call fgStack_push("")
     call fgStack_push("")
     call socketManager("reps","add-on header", "default")
 end if
```

2. **After you create this trigger, select Save Trg File from the File pull down menu.**

3. **Once your trigger is saved, select Compile 4GL Code from the Run pull-down menu.**

   Don't try to run your program yet, it won't work until you complete the next exercise,

Chapter 12

Working with Switchboxes

Main topics:

- Switchbox Overview
- How do Screens Get Into Switchbox
- Zooms and Switchboxes

Switchbox Overview

VDT Application Code Generator generated code features Switchbox logic. In general terms, a switchbox manages flow control between library functions and local functions. There are two types of Switchboxes:

1. **Screen-Level switchbox**

2. **Function-Level Switch box**

Screen-Level Switchbox

The screen level switchbox resides in main.4g1 and passes control to the appropriate program screen. Screen-level switchbox is controlled by the switchbox function. This function reads the value in the global scr_id variable. The scr_id variable can contain any valid form specification file in your program less the .per extension. For example, your input program might contain the following form specification files:

Filename	**Screen Type**	**scr_id Value**
browse.per	browse	browse
oust.per	add-on header	cust
oust_zm.per	zoom	cust_zm
order.per	header / detail	default
stookzm.per	zoom	stockzm

As you can see, your header / detail screen (or main screen) receives default as its `scr_id` value. If your program contained a header screen instead of a header/detail screen, the header screen would receive default as its `scr_id`.

Depending on the value in `scr_id` flow is passed to the function level switchbox.

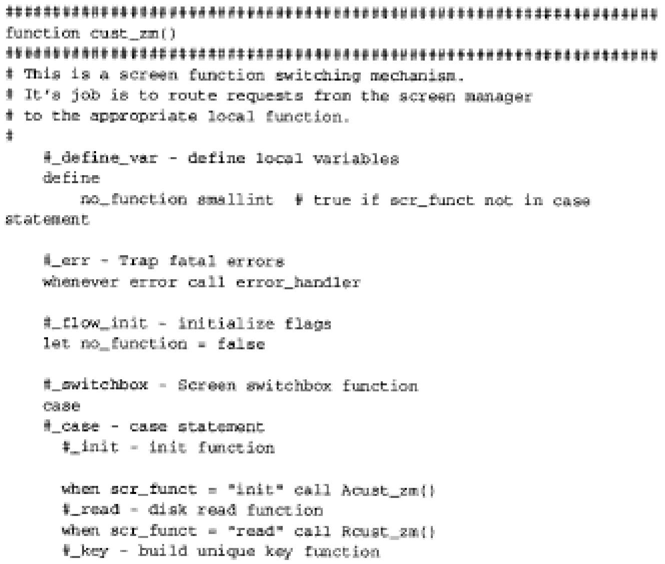
Function-Level Switchbox

The function-level switchbox determines what happens next. For each form specification file in your program (i.e., for each screen used by your program) a function-level switchbox is generated. The function-level switchbox reads the value in the scr_funct variable. Once this value is read, the function level switchbox uses a large case statement to determine the appropriate action.

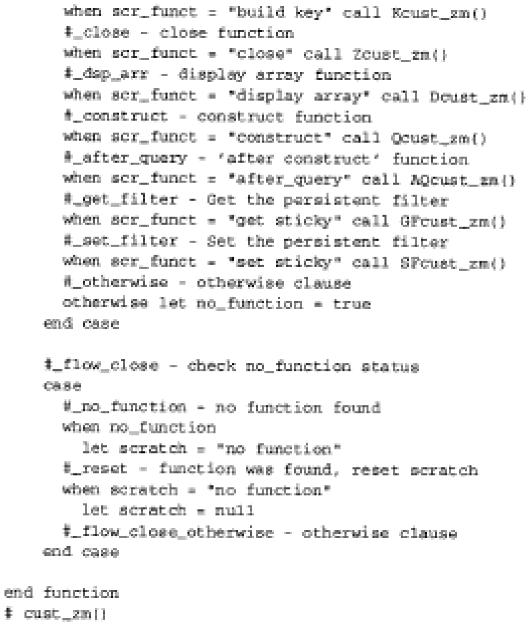
When the VDT Application Code Generator creates each function-level switchbox, it names the swichbox after the form specification file or `scr_id` that it relates to. The only exception being header and header/ detail form specification files. These files use the `lib_screen` function as their function-level switchbox.

For example, if the `scr_id` variable equals `cust_zm`, a `cust.zm` function is generated in the `cust_zm.4gl` file. This function contains all the possible actions that can take place from within the `cust_zm` screen.

The following code illustrates an example `cust_zm` switchbox function.



```
######################################################################
function cust_zm()
######################################################################
# This is a screen function switching mechanism.
# It's job is to route requests from the screen manager
# to the appropriate local function.
#
    #_define_var - define local variables
    define
        no_function smallint  # true if scr_funct not in case
statement

    #_err - Trap fatal errors
    whenever error call error_handler

    #_flow_init - initialize flags
    let no_function = false

    #_switchbox - Screen switchbox function
    case
    #_case - case statement
      #_init - init function

      when scr_funct = "init" call Acust_zm()
      #_read - disk read function
      when scr_funct = "read" call Rcust_zm()
      #_key - build unique key function
```



```
    when scr_funct = "build key" call Kcust_zm()
    #_close - close function
    when scr_funct = "close" call Zcust_zm()
    #_dsp_arr - display array function
    when scr_funct = "display array" call Dcust_zm()
    #_construct - construct function
    when scr_funct = "construct" call Qcust_zm()
    #_after_query - 'after construct' function
    when scr_funct = "after_query" call AQcust_zm()
    #_get_filter - Get the persistent filter
    when scr_funct = "get sticky" call GFcust_zm()
    #_set_filter - Set the persistent filter
    when scr_funct = "set sticky" call SFcust_zm()
    #_otherwise - otherwise clause
    otherwise let no_function = true
  end case

  #_flow_close - check no_function status
  case
    #_no_function - no function found
    when no_function
      let scratch = "no function"
    #_reset - function was found, reset scratch
    when scratch = "no function"
      let scratch = null
    #_flow_close_otherwise - otherwise clause
  end case

end function
# cust_zm()
```

As you can see from the sample code, there are several logical points within a switchbox function. The extended `case` statement provides several code points that you can customize using triggers or block commands (see "Creating Triggers" on page 150 and "Block Commands" on page 251).

How Screens Get Into Switchbox

The screen level switchbox function, which actually uses the name `switchbox`, determines which program screen is active and selects the correct program flow based on the active screen. The `switchbox` function evaluates the value in scr_id to know which screen and thus which series of code to process. For this reason, it is important that you define the links between your main program screen and your secondary screens accurately. In chapter 11, you learned how to link an add-on header screen to a main screen using an `after_field` trigger, the function, and the `socketManager` function. You must also use a `switchbox_items` trigger. By using the `switchbox_items` trigger, you declared your add-on header screen to the `scr_id` variable. In essence, you made the `switchbox` function aware of your add-on header screen.

For the screen level switchbox function to work, you must make sure that all your secondary screens get linked in properly using the `switchbox_items` trigger.

Code to place zoom screens into the `switchbox` function gets generated automatically. When the VDT Application Code Generator reads a zoom attachment (i.e., the zoom= line in the form specification (*.per) file), it places not only the library function that invokes the zoom screen, but also the entry into the `switchbox` function. The VDT Application Code Generator adopts responsibility for placing all zoom support logic into code.

The main program screen (your header or header/detail) also gets placed in automatically when you run the VDT Application Code Generator. All other screen types must be added using the `switchbox_items` trigger.

The switchbox_items Trigger

You make screens known to the `switchbox` function with the `switchbox _items` trigger. The `switchbox _items` trigger uses the following syntax:

```
defaults
   switchbox_items
         screen_name screen_function_name
```

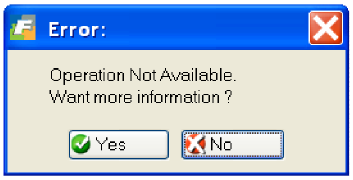
Here is an example of an add-on screen being placed into the switchbox function by the `switchbox _items` trigger:

```
defaults
   switchbox_items
         cust S_cust ;
```

The above code, placed in a trigger (*.trg) file results in the following line added to `switchbox` in main.4gl:

```
when scr_id = “cust" call S_cust()
```

If a request is passed to switchbox by a library function and the switchbox function does not know the screen to pass it to, then the following error message appears:



Operation Not Available.
Want more information ?

Section Summary

- Numerous screens combine to constitute an input program. All programs have a main screen (called the "default" screen) which is either a header or header/detail type screen. Other screens such as zoom screens and add-on header screens are attached to the main screen.

- All screens that interact with an input program must be known to the `switchbox` function. The `switchbox` function constitutes the screen-level `switchbox`. There is also a function-level `switchbox`. Both types of `switchbox` functions exist in every input program.

- Library functions pass generic requests to local code via the two `switchbox` function levels. The first `switchbox` level (the screen level) uses the `switchbox` function. Its job is to receive the request from the library functions and determine which program screen to use. The `switchbox` function is generated in local code and placed in the main.4gl file.

- The second `switchbox` level (the function level) evaluates the screen-level request and passes control to the appropriate lowlevel function, which handles the request. The low-level function contains all the code to process the request. When complete, program control returns to the library function.

- The second-level `switchbox` contains functions with a variety of names. The `lib_screen` function is the second-level `switchbox` function for the main screen. This function handles requests including highlighting fields and recording values.

- Since the `switchbox` function passes requests based on the program screen, all screens interacting with the input program must be "known" to the switchbox function. In other words, all screens must have logic in `switchbox` so that when a request is passed to the `switchbox` function, it knows where to pass the request.

- The VDT Application Code Generator automatically adds the main screen and zoom screens to the screen level switchbox in main.4gl.   All other screen types must be added using the `swichbox_items` trigger.

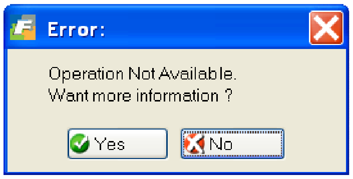
- You can use the `switchbox _items` trigger to make your screen known to the `switchbox` function. Thus, when requests to perform something to your screen are received by the `switchbox` function, it can direct control to the appropriate code.

Exercise 12

Objective: To create a `switchbox _items` trigger that "links" the Sales Person add-on screen to the Customer Entry screen.

Examine main.4gl

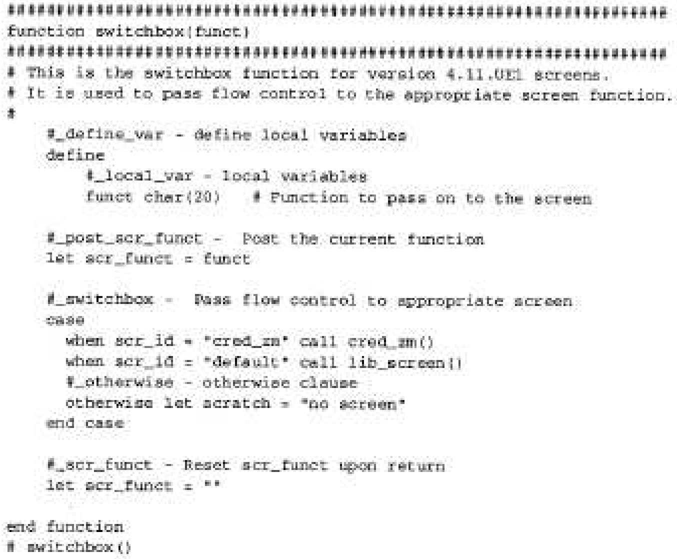
Had you tried to run your Customer Entry program at the end of Exercise 11, and attempted to access your new add-on screen, the following error message would have occurred:



Operation Not Available.
Want more information ?

1. **Exit the VDT Form Designer and use vi to open main. 4gl.**

2. **Search for the switchbox function.**



```
####################################################################
function switchbox(funct)
####################################################################
# This is the switchbox function for version 4.11.UE1 screens.
# It is used to pass flow control to the appropriate screen function.
#
    #_define_var - define local variables
    define
        #_local_var - local variables
        funct char(20)    # Function to pass on to the screen

    #_post_scr_funct -  Post the current function
    let scr_funct = funct

    #_switchbox -  Pass flow control to appropriate screen
    case
      when scr_id = "cred_zm" call cred_zm()
      when scr_id = "default" call lib_screen()
      #_otherwise - otherwise clause
      otherwise let scratch = "no screen"
    end case

    #_scr_funct - Reset scr_funct upon return
    let scr_funct = ""

end function
# switchbox()
```

This function contains a "flow control" `case` statement that is based on the `scr_id` variable. As you can see, your `reps` add-on header screen is not yet a part of this statement. Before your add-on header screen works properly, you have to create a special trigger, known as the `switchbox _items` trigger. This trigger makes your add-on header screen known to switchbox.

Add the switchbox_items Trigger

The `switchbox _items` trigger creates a "when" clause in the `switchbox` function. This trigger goes in the "defaults" section of the trigger file.

3. **Start the VDT Form Designer and open your cust form..**

4. **Select Triggers >> from the Define pull-down menu.**

   The Choose a Trigger Class box appears**.**

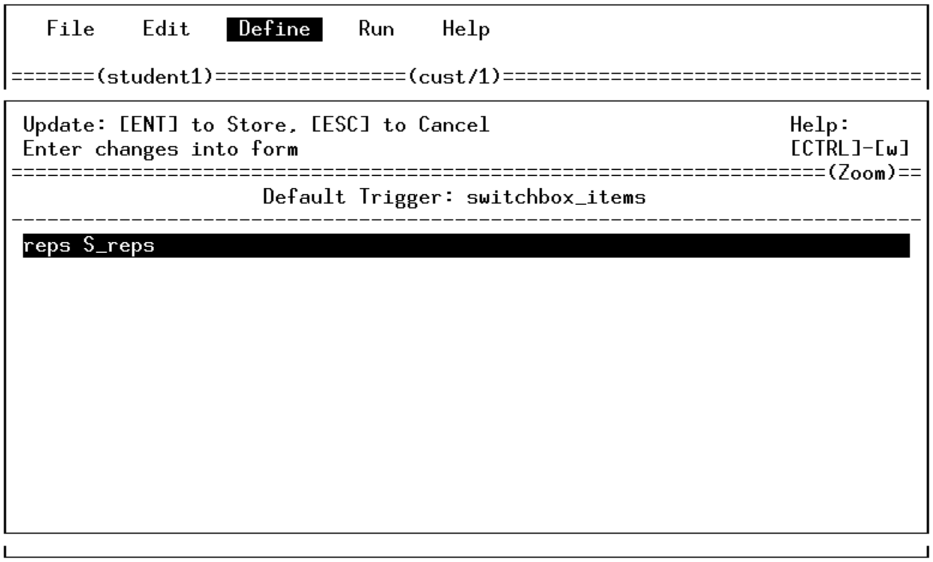
5. **Select Default as the Trigger Class.**

   The Choose a Trigger list box appears.

6. **Select `switchbox _items` trigger.**

   The editing window appears.

7. **Add the following line then save your trigger (select Save Trg File then Save Form. from the File pull-down menu).**

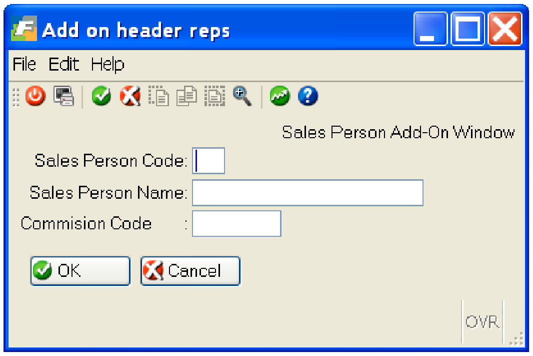


```
=======(student1)================(cust/1)===================================
 Update: [ENT] to Store, [ESC] to Cancel                          Help:
 Enter changes into form                                          [CTRL]-[w]
=====================================================================(Zoom)==
                      Default Trigger: switchbox_items
-----------------------------------------------------------------------------
 reps S_reps
```

The first value (in this case reps) represents the name of your add-on header screen. The second value (S_reps) represents the name of the function that will control your screen.

8. **Compile and run the Customer Entry program.**

   What happens when you type xx in the Sales Code field? You should see the Sales Person add-on screen.



Sales Person Code:
Sales Person Name:
Commision Code :

Chapter 13

Adding Window Titles

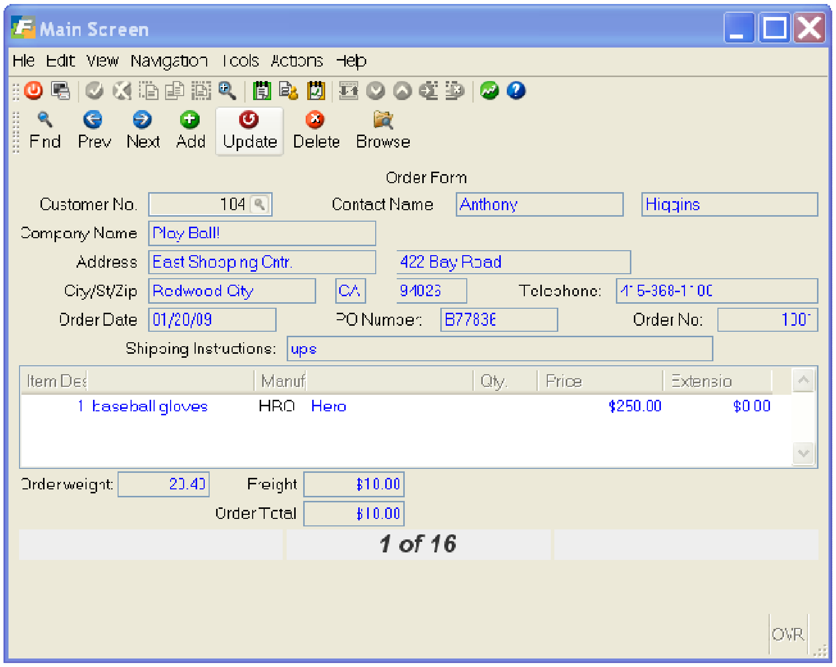
Main Topics:

- Adding Window Titles
- Localized Strings

Adding Window Titles

By default window titles are not descriptive. The default title of the main window is Main Screen. The additional windows are titled with the name of the .per file.

Below are examples of a main program window and a zoom with the default window titles.



Order Form

Customer No. 104 Contact Name Anthony Higgins

Company Name Play Ball!

Address East Shopping Cntr. 422 Bay Road

City/St/Zip Redwood City CA 94023 Telephone: 415-368-1100

Order Date 01/20/09 PO Number: B77836 Order No: 1001

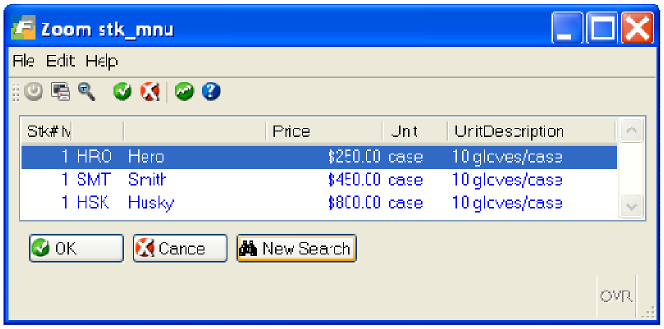
Shipping Instructions: ups

Item Des	Manuf	Qty.	Price	Extensio
1 baseball gloves	HRO Hero		$250.00	$0.00

Orderweight 20.40 Freight $10.00

Order Total $10.00

1 of 16



Stk# M		Price	Unit	UnitDescription
1 HRO	Hero	$250.00	case	10 gloves/case
1 SMT	Smith	$450.00	case	10 gloves/case
1 HSK	Husky	$800.00	case	10 gloves/case

You add descriptive window titles using is a 5 step procedure.

1. **Add a Text= clause to the LAYOUT keyword in the .per file.**
2. **Use fglform –m <per file name> to generate a localized string source file. The name of this string source file must match the program name.**
3. **Modify the string source file, manually entering descriptive window titles.**
4. **Create the binary string file using fglmkstr.**
5. **Compile the program using fg.make.**

1. Add TEXT=%"module.program.perfilename" to the LAYOUT keyword in .per file.

You use vi or another text editor to add the TEXT=%"module.program.perfilename" to the .per file. Below are examples from training program 5. Notice the format of the string between the quotes. The format consists of 3 parts - the module name without the .4gm, the program directory name without any extension, and the per file name without the .per extension.

order.per:

```
SCHEMA student1
LAYOUT (TEXT=%"train.screen5.order")
VBOX nm_vbox_main (TAG="tg_vbox_main")
GRID
{
                                Order Form
 Customer No.:[A0     ]    Contact Name:[A1             ][A2              ]
 Company Name:[A3                  ]
      Address:[A4                  ][A5                    ]
  City/St/Zip:[A6              ][A7] [A8   ] Telephone:[A9                ]

   Order Date:[AA      ]     PO Number:[AB        ]    Order No:[AC       ]
```

cust.per:

```
SCHEMA student1

LAYOUT (TEXT=%"train.screen5.cust")
VBOX nm_vbox_main (TAG="tg_vbox_main")
GRID
{
                              CUSTOMER FORM

            Number          :[A0     ]
```

2. Generate a localized string source file using fglform –m

You create the framework of the localized string source file using fglform –m <perfile name>. The string source file must match the 42r file. The string source file for the screen5 training program was made using the following commands.

In the $fg/acconting/train.4gm/screen5.4gs directory:

```
fglform –m order.per > screen5.str
fglform –m cust.per  >> screen5.str
fglform –m cust_zm.per >> screen5.str
fglform –m stk_mnu.per >> screen5.str
fglform –m stockzm.per >> screen5.str
```

3. Modify the string source file, manually entering descriptive window titles.

The output of fglform –m from the above commands must be modified; adding the descriptive window titles. Use vi to modify string source file as outlined below.

Before modification:

```
"train.screen5.order"="train.screen5.order"
"train.screen5.cust"="train.screen5.cust"
"train.screen5.cust_zm"="train.screen5.cust_zm"
"train.screen5.stk_mnu"="train.screen5.stk_mnu"
"train.screen5.stockzm"="train.screen5.stockzm"
```

After modification:

```
"train.screen5.order"="Order Form"
"train.screen5.cust"="Customer Form"
"train.screen5.cust_zm"="Customer Zoom"
"train.screen5.stk_mnu"="Stock/Manufacturer Zoom"
"train.screen5.stockzm"="Stock Zoom"
```

4. Create the binary string file using fglmkstr.

You create a binary string file using fglmkstr using the following command.

```
fglmkstr screen5.str
```

This command creates a screen5.42s. Notice how the 42s file is named according to the program.

5. Compile the program using fg.make.

In order for the binary string file to be used the .per files must be recompiled. You can use fg.make to recompile them or compile each one manually using fglform <per file name>.

Section Summary

- Localized strings can be used to add descriptive window titles
- Adding descriptive window titles is a 5 step process

1. Add a Text= clause to the LAYOUT keyword in the .per file.
2. Use fglform –m <per file name> to generate a localized string source file
3. Modify the string source file, manually entering descriptive window titles.
4. Create the binary string file using fglmkstr.
5. Compile the program using fg.make**.**

Exercise 13A

Objective: Use localized strings to add descriptive window titles to the Customer Entry Program.

Add a Text= clause the .per files

1. **cd $HOME/labs/aw.4gm/i_cust.4gs**

2. **Add the indicated text to each of the .per files using vi.**

cust.per:

```
LAYOUT (TEXT=%"aw.i_cust.cust")
VBOX nm_vbox_main (TAG="tg_vbox_main")
GRID
```

cred_zm.per

```
LAYOUT (TEXT=%"aw.i_cust.cred_zm")
VBOX nm_vbox_main (TAG="tg_vbox_main")
TABLE nm_table_zoom (WIDTH=49, TAG="tg_table_zoom")
```

Use fglform –m <per file name> to generate a localized string source file

1. **Run fglform –m <per file name> on each per file redirecting output to the string source fie named according the program name.**

```
fglform –m cust.per > i_cust.str
fglform –m cred_zm.per >> i_cust.str
```

Modify the string source file, manually entering descriptive window titles

1. **Use vi to modify the i_cust.str file as indicated below.**

```
"aw.i_cust.cust"="Customer Form"
"aw.i_cust.cred_zm"="Credit Zoom"
```

Create the binary string file using the fglmkstr utility

1. **Run fglmkstr on the localized string file. This creates an i_cust.4gs file.**

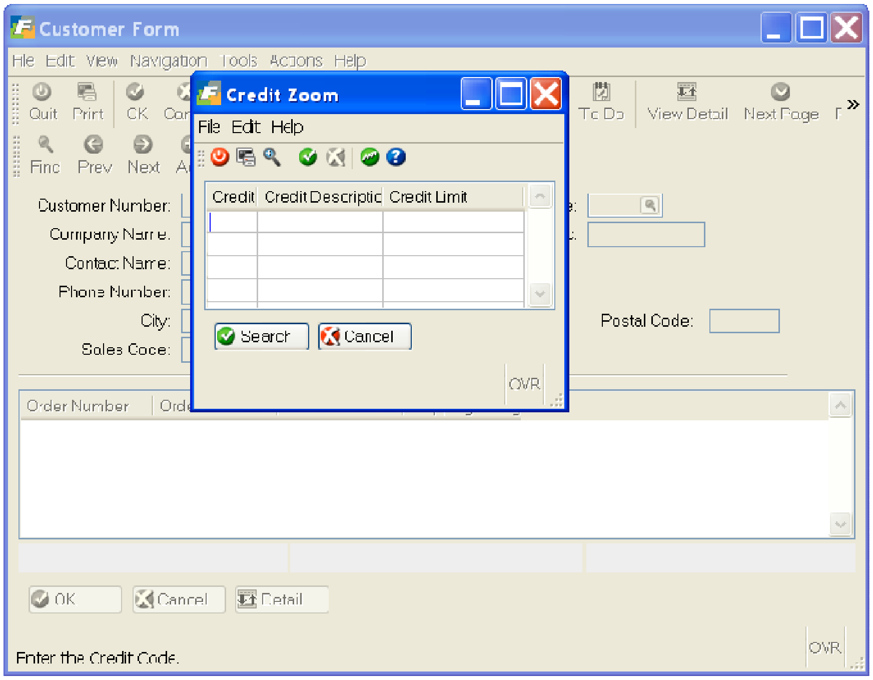
```
fglmkstr i_cust.str
```

Compile and Test the Program

In order for the program to use the localized strings, the changed .per files must be recompiled. The easiest way to do this is to use fg.make.

1. **run fg.make in the program directory**

2. **run your program to test the window titles.**



Enter the Credit Code.

Chapter 14

Creating Extension Screens

Main topics:

- Extension Screen Overview
- Attaching Extension Screens to Main Screens

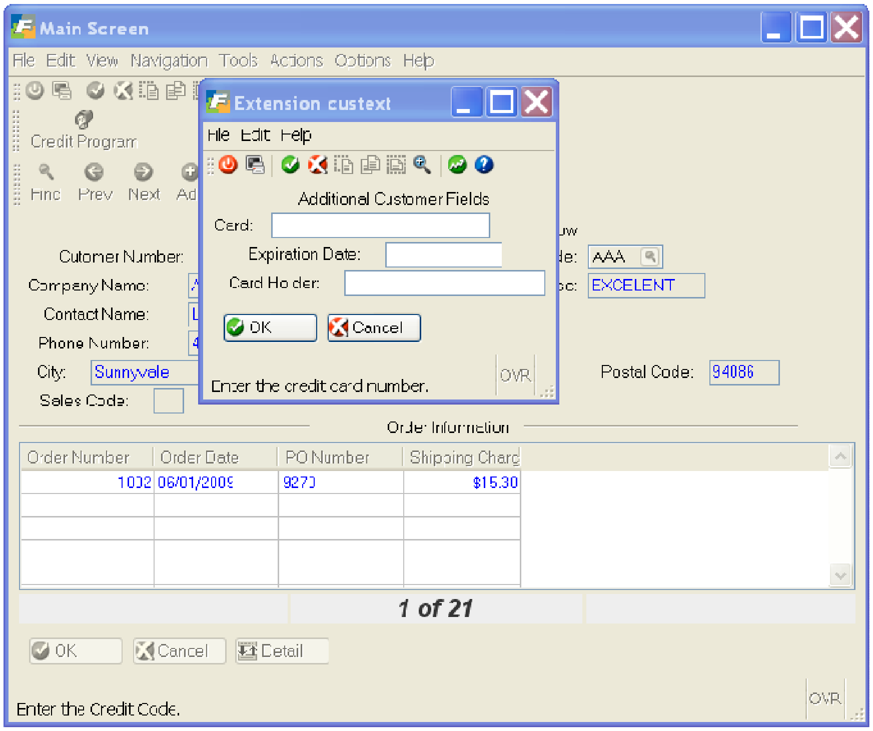
Extension Screen Overview

Extension screens provide users with additional screens. In effect, extensions screens "extend" the main screen.

Many times, tables contain too many columns to fit on a single input screen. Because of a limited amount of "screen real estate," it is sometimes useful to create extension screens from the main screen. By adding extension screens you can simplify and clarify your main screen.

In addition, extension screens can provide conditional data-entry logic. For example, one of your input programs might contain a Payment Method field. Perhaps your company recognizes three types of payment methods: cash, check, and charge. Depending on the value in the Payment Method field, you can initiate different extension screens. Say for example that the charge value initiates an extension screen that contains Card Type, Number, and Expiration Date fields.

The following figure shows an extension screen for adding additional customer information:



Additional Customer Fields

Card:

Expiration Date:

Card Holder:

Enter the credit card number.

Customer Number:

Company Name:

Contact Name:

Phone Number:

City: Sunnyvale

Sales Code:

de: AAA

c: EXCELENT

Postal Code: 94086

Order Information

Order Number	Order Date	PO Number	Shipping Charg
1032	06/01/200	9273	$15.30

1 of 21

Enter the Credit Code.

Attaching Extension Screens to Main Screens

Extension screens, like the other screen types you've learned about, are attached to the main screen by the socketManager function. But also like the other screens, extension screens use a syntax all their own.

You can initiate an extension screen from a program event. There are useful triggers that work well with extension files, such as:

- after_input
- after_field
- before_field
- on_event

You can initiate extensions screens from program events and a custom toolbar. This is covered in chapter 18.

You must complete the following basic steps to create and attach an extension screen:

1. **Use the VDT Form Designer to paint the extension image and save the image to a form specification (*.per) file (see "Creating a Form Image" on page 38).**
2. **Create a switchbox_items trigger to declare the extension screen to the screen-level switchbox function (see "The switchbox_items Trigger" on page 201).**
3. **Create a trigger that initiates the extension screen.**
4. **Use `socketManager` to attach your extension screen.**

1. Paint the Extension Screen Image

Use the VDT Form Designer to create the extension screen image for your extension screen. When you create the screen, make sure to select extension as the screen type. Remember, extension screens are for additional input fields that cannot fit or are not contained on the main screen. Unlike add-on header screens, extension screens work off the same table as the main screen.

Once you create the image, save it with the Save Form option under the File pull-down. The Save Form option generates a form specifiction (*.per) for your extension screen.

2. Add the Extension Screen to the switchbox_ltems Trigger

Like other screens, you need to declare extension screens using the `switchbox_items` trigger. For example, if your extension screen is named `custext`, your `switchbox_items` trigger would look as follows:

```
defaults
```

```
switchbox_items
   custext S_custext;
```

3. Create a Trigger to Initiate the Extension Screen

Next, create a trigger that initiates the extension screen. For example, if you want to initiate your extension screen after the user moves past the Customer No. field, insert the following lines of code:

```
input 1
after_field customer_num
```

4. Use socketManager to Attach the Extension Screen

Finally, use `socketManager` to attach the extension screen. Unlike the add-on header and zoom screen types, extension screens don't require you to use the `fgStack_push` function. You only need to use the `socketManager` function. For example, to attach the `custext` extension screen to your main screen, insert:

```
input 1
after_field customer_num

   call socketManager("custext "extension", "flat_ext");
```

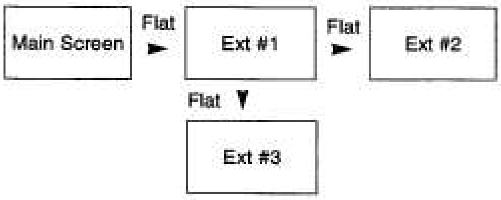
When you attach extensions screens with `socketManager`, the flow parameter differs slightly. Instead of using `default` as the flow parameter, extension screens use one of three values: `flat_ext`, `deep_ext`, and `view`. Extension screens require multiple flow values because you can link multiple extension screens together. The following list explains the different flow parameters available with extension screens.

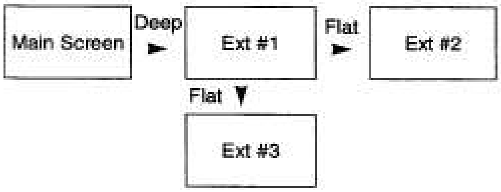
`flat_ext` The `flat_ext` flow parameter determines how the program handles an interrupt (i.e., user pressing [ESC]). If a user presses [ESC] in Ext #1 in the first diagram below, all edits to Ext #2 and #3 are retained.

`deep_ext` The `deep_ext` flow parameter operates in the exact opposite of the `flat_ext` parameter. If a user presses [ESC] in Ext #1, all edits in Ext #2 and Ext #3 are rolled back.

`view` This flow only lets users view the data within extension screens.

If a user pressed [ESC] in Ext #1, all edits to Ext #2 and #3 are rolled back.





By putting all these code pieces together and using the Featurizer to merge your trigger file, your extension screen gets attached.

Section Summary

- Extension screens provide users with additional screens. In effect, extensions screens "extend" the main screen.
- Extension screens provide extra space, so you can simplify and clarify the main screen. In addition, extension screens can be used as data-entry control devices.
- You can initiate an extension screen from any program event, such as a trigger, a pop-up menu, or a mapped hot key.
- You attach extension screens with the `socketManager` function. The `socketManager` recognizes three different flow parameters for extension screens: `flat_ext`, `deep_ext`, and `view`.
- Each flow parameter has a different function. The `flat_ext` flow is for extension screens that are independent from calling screens. The `deep_ext` flow is for extension screen that are dependent on calling screens. The `view` flow is for extension screens that only display data (i.e., users can't add or update values on a view extension screen).

Exercise 14

Objective: To create an extension screen that allows you to enter additional data into the `customer` table.

You will start by adding three columns: `card_no`, `exp_date`, and `card_holder`. You will then place these columns on a `custext` extension screen. Finally, you will incorporate this screen into your Customer Entry program with an `after_input` trigger.

Add the Columns

1. **Using the VDT Form Designer (or dbaccess) add the following columns to the customer table.**

Column Name	Description	Type
card_no	Card Number	char(20)
exp_date	Expiration Date	date
card_holder	Card Holder	char(20)

Save these changes and return to the VDT Form Designer. Remember to update your database's schema file in $fg/data with sch.sh <database name>.

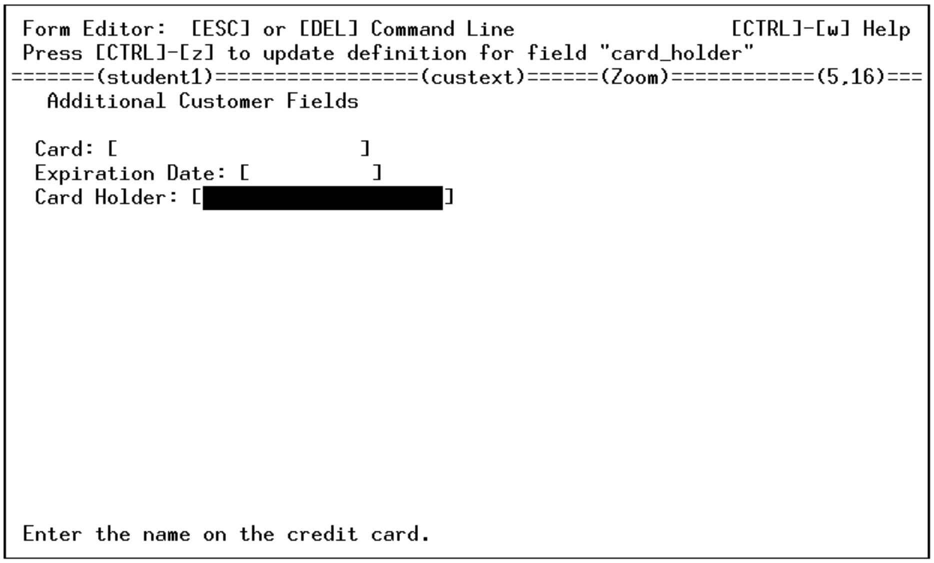
Create the Extension Screen

1. **Use the VDT Form Designer to create a new form. Name it `custext` and define it as type extension.**

2. **Create a title line.**

   Additional Customer Fields

3. **Label and define three fields, one for each of the columns you just added. Your extension screen should look as follows:**



```
 Form Editor:  [ESC] or [DEL] Command Line              [CTRL]-[w] Help
 Press [CTRL]-[z] to update definition for field "card_holder"
=======(student1)================(custext)======(Zoom)============(5,16)===
   Additional Customer Fields

  Card: [                     ]
  Expiration Date: [          ]
  Card Holder: [                    ]









 Enter the name on the credit card.
```

Save and Generate

1. **Use Save Form to save your new form.**

2. **Invoke the VDT Application Code Generator to create 4GL code for your new form.**

3. **When the Generator has finished, exit the VDT Form Designer and list your files (type 1s at the LINUX prompt).**

   Notice that the Generator has created a new source code file for your **`custext.per`** file. This source code file contains all the lowlevel source code to drive your **`custext`** extension screen.

Create an after_input Trigger

You can use the `after_input` trigger to attach your `custext` extension screen to your Customer Entry program. Several other triggers will work as well, but the `after_input` trigger is a common choice.

1. **Use vi to open cust. trg.**

2. **In the input 1 section add the following lines of code:**

   ```
   # after_input trigger to call my custext extension screen
   call socketManager("custext", "extension". "flat_extlt);
   ```

Note

If you already have an `after_input` trigger defined, which you should because you created one in Exercise 10C you must add these lines below it. You do not, and cannot, add two identical triggers (for example, two `after_field` `customer_num` triggers). You should just combine the code under one trigger. Make sure to remove the semi-colon that terminates the first `after_input` trigger or a syntax error will occur.

Your complete `after_input` trigger should look as follows:

```
after_input
    if p_stomer.credit_code is null
    then
        error "You must fill in the Credit Code field"
        let nxt_fld "credit_code"
    end if
   # after_input trigger to call my custext extension screen
    call socketManager{"custext"1 "extension", «flat_ext"};
```

Also add a custext line to your switehbox_items trigger.

This trigger should now include three lines. A **`reps`** line, a **`saleszm`** line; and a custext line.

```
switchbox_items
    reps S_reps
    saleszm saleszm
    custext S_custext;
```

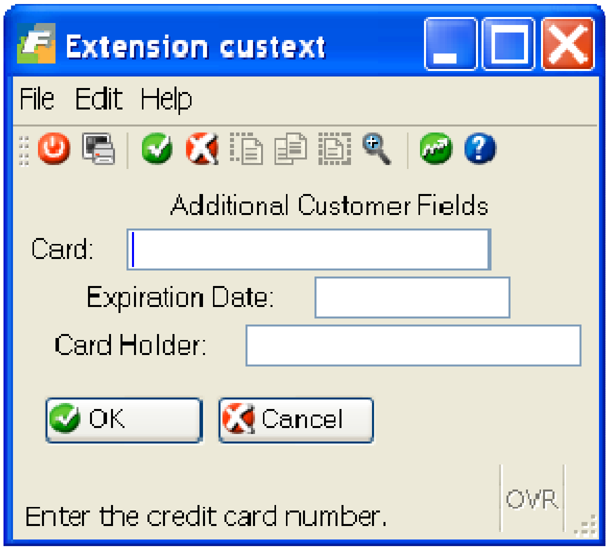
Save cust.trg.

Compile the code and run Customer Entry.

Select Add to create a new record.

Fill in all the fields on the header portion of the screen and click the Detail button Detail **to move to the detail portion.**

Your **`custext`** extension screen appears.



Additional Customer Fields

Card:

Expiration Date:

Card Holder:

Enter the credit card number.

Complete the Additional Customer screen and quit out of your Customer Entry program.

Chapter 15

Version Control and Conventions

Main topics:

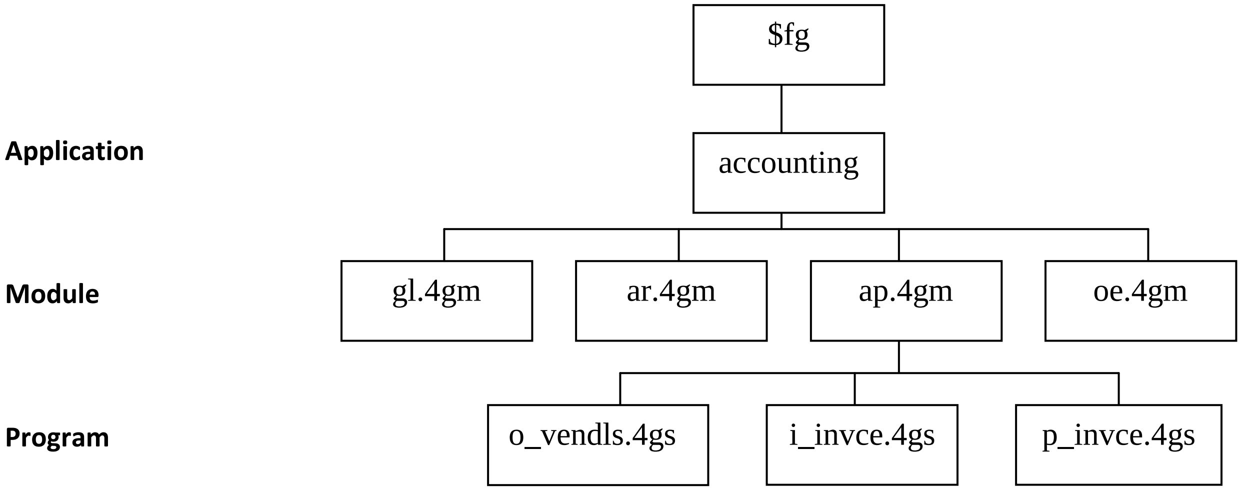
- The Fitrix Directory Structure
- Version Control Overview
- Building Custom Versions
- Table Naming Conventions

The Directory Structure

All software products utilize a four-tiered directory structure: Fitrix, application, module, and program.

The directory contains all your programs. It is usually represented by the `$fg` environment variable. The application tier is rather general. It contains a set of related modules. The module level is more specific. Every module directory is given a .4gm extension. Within each module directory exists a set of related programs. The program tier is the lowest tier. Each program directory contains a single input, output, or posting program. Program directories have a .4gs extension.

The following graphic shows a sample directory structure:

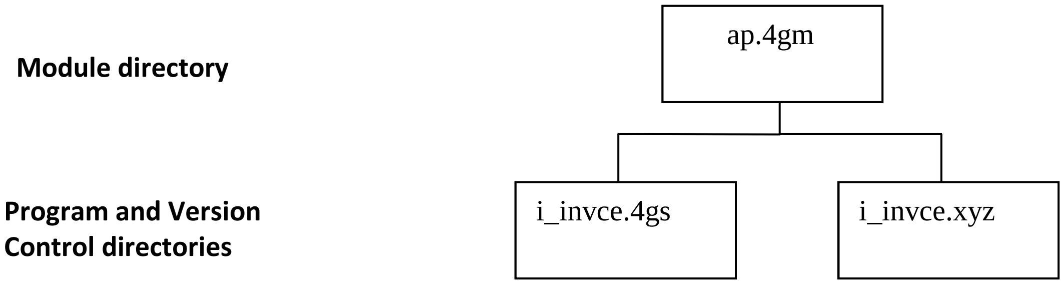


Version Control Overview

Version control lets you create multiple flavors of a program without duplicating code. Version control is useful when two or more users require different program functionality.

Version Control Directories

Version control uses custom directories that are parallel to program (*.4gs) directories. In the custom directories, you place specification or trigger files that are unique to your custom version. By default, version control recognizes *.4gc directories as custom version directories. You can have as many custom version directories as you want. For example, if you want to have a custom version of the Invoice Entry program, you need to create a custom directory.



The `cust_path` variable lets you specify the order in which version control works. To merge base functionality with the new functionality you've added in i_invce.xyz, set `cust_path` as follows:

```
cust_path = xyz:4gs ; export cust_path
```

The `cust_path` variable describes the starting point from which the merge utility should start on the `cust_path`. For the above example, `cust_key` should be set as follows:

```
cust_path = xyz:4gs ; export cust_path
```

fg.newver

The fg.newver script sets up a custom directory for you. When you run fg.newver, make sure you set cust_key to the directory extension you are using for your custom programs. The default is '4gc'. The syntax of fg.newver is:

```
fg.newver <base directory>
```

Where <base directory> is the .4gs directory without the .4gs extension.

```
Example: fg.newver i_cust
```

fg.newver does the following:

1. **Creates a Makefile**
2. **Creates a base.ext file with start file statements for each 4gl in the .4gs directory**
3. **Creates a base.set file from the base.set in the .4gs directory**
4. **Copies over any .42f files from the .4gs directory**

Building Custom Versions

You can use version control logic to build multiple versions of a base program or to build increasingly rich enhancements to a base program. Perhaps the simplest case involves modifying an input screen.

For example, suppose you are customizing the `i_invce.42r` program, which is located `in ap.4gm/i_invce.4gs` directory. You know that this program is built from a series of form specification (*.per) files, where each *.per file represents a different program screen. If, on your custom version, you want to add an input field to the main screen, you would need to complete the following steps:

1. **Make sure you set cust_key and cust_path correctly. Use fg.newver to create and setup your custom program directory. (`i_invce.xyz`).**
2. **Copy the main screen .per file to your custom directory.**
3. **Use the VDT Form Designer to add a field to the screen.**
4. **Run the VDT Application Code Generator (`fg.screen`) in the custom directory (`i_invce.xyz`).**

Once initiated, the Screen Code Generator takes the following steps:

1. **Searches your current directory (`i_invce.xyz`) and reads the modified form specification file.**

2. **Searches the base directory (`i_invce.4gs`) for additional specification files.**

3. **Generates the 4GL code necessary to build your custom program.**

You can then run the `fg.make` utility in the custom directory to compile the custom program. Once complied, you can issue the following command to run the custom version:

```
fglrun *.42r –d <Database Name>
```

Procedures to Modify Programs

Modifying Screens:

1. **Copy the .per files to be modified to your customer directory**

2. **Make your changes**

3. **Regenerate the code using fg.screen**

4. **Recompile using fg.make**

Modifying Reports:

1. Copy the report.ifg file to be modified to your custom directory

2. Make changes to the report.ifg

3. Regenerate the code using fg.report

4. Recompile using fg.make

Making Changes to .4gl files:

1. Make changes using an .ext file in your custom directory

2. Add your .ext file to the base.set file in your custom directory

3. Merge the code and recompile using fg.make

Modifying or Adding Any Database Tables:

1. Make sure to save any alter statements and create statements in an .sql file. Add the .sql file to $fg/data/sql.4gc. If you use a different custom directory extension, change the .4gc extension. A good convention to follow is to use <table

name>.sql for the .sql file name. Making these .sql files clearly documents the changes you made and allows you to re-use the statements when you apply the modification to other operating environments.

2. If you add columns to an already existing table, remember to include sql statements to properly initialize the new column.

Document Your Modification:

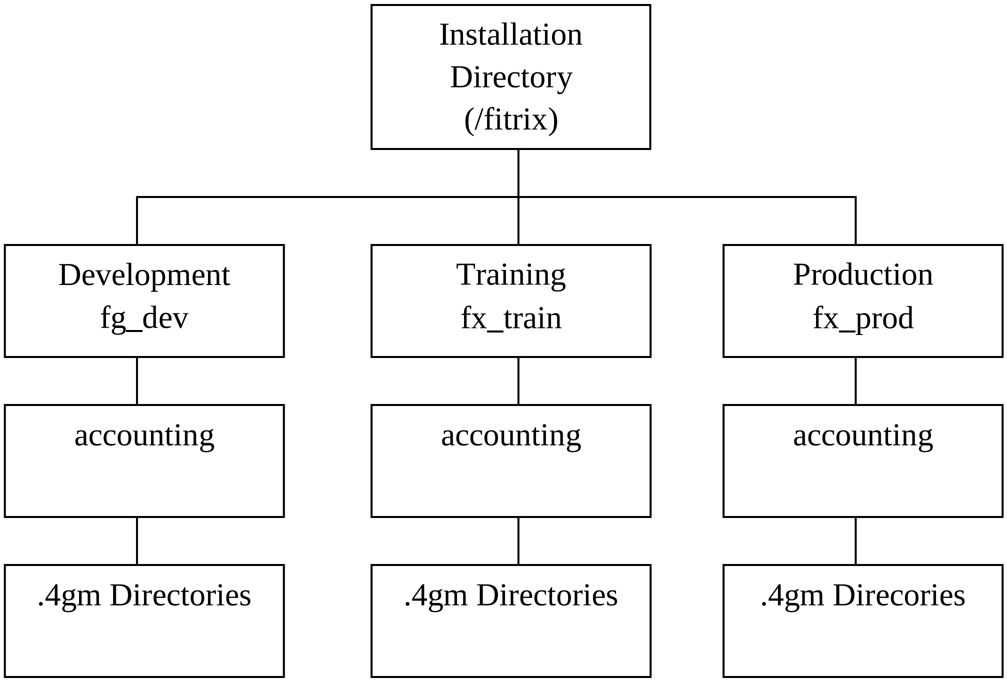
1. Copy the INFO file form the .4gs directory to your custom directory.
2. Add entries to the bottom of the INFO file for your changes. By doing this you are also documenting the patch level of the program for which you made your changes.

Development, Training and Production Environments

The Fitrix product comes with 3 environments:

- The Development environment "fx_dev" is a completely separate area to allow your programmers to develop customizations to Fitrix without disturbing your production software. Once a change has been developed and tested, the new software should be installed in the production area. The Development database is called 'standard' and it is fully populated with sample data from a sample company when Fitrix is installed.
- The Training Environment "fx_train" is another completely separate area to allow end users or programmers to train on the Fitrix software.
- The Production Environment "fx_prod" contains the versions of the programs your business will run, and the "live" database you will use. The Production database is called "live" and it is empty when Fitrix is installed so that it is ready for you to begin setting up your company's data.

Each environment is under the main installation directory according to the following diagram.



Moving Modifications to Other Environments

You ordinarily make your modifications in the development environment. After your modification is tested in the development environment you need to move your modification to the training and production environments.

Below is a procedure you can use to move your modification to another environment. The instructions below are for moving a modification from the development environment (/firtix/fx_dev) to the production environment (/fitrix/fx_prod). These instructions also use a .4gc as the custom directory extension.

1. Verify that the target environment does not already contain a .4gc version of the program. If the target environment does have a .4gc version of the directory, rename the target environment directory to a xxxx.4gc.yyyymmdd where xxxx is the base name of the directory and yyyymmdd is the current date.

   Example: mv i_invce.4gc i_invce.4gc.20101001

2. **Navigate to the parent module directory in the development directory. Execute:**

   ```
   cp –R i_invce.4gc /fitrix/fx_prod/accounting/ar.4gm
   ```

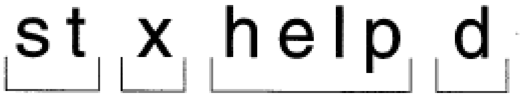
3. **If the modification included database changes, execute the .sql file you created earlier against the appropriate databases for the production environment.**

4. **If the production environment is on a separate physical server where the Linux version may be different than the source system, it is always a good idea to re-compile the program on the new server (assumes you have a development license on the production server).**

Table Naming Conventions

Many accounting application tables follow a specific naming convention. The manufacturing application modules follow a different table naming convention. For accounting tables, each table name is composed of eight characters, and the last six characters must be unique. The eight-character name is divided into four sections.

Table names are divided into four sections.



The first two characters classify the table as either a application table or a Screen Code Generator table:

st	Application Table
cg	Code Generator Table

The third character relates to the product, for example:

s	Screen
d	Database Program in VDT Form Designer
m	User-Defined Menus
x	Non-Product Specific

The next four characters classify the type of data, for example:

eror	Error Text
help	Help Text
mssg	Messages
note	User-Defined Notes

The last character specifies the role of the table:

r	Reference (usually the same as a header)

d	Detail
h	Header

Section Summary

- Fitrix uses a four-tiered directory structure. The top tier is set by the $fg variable. It points to the installation directory. The second tier is known as the application directory. It contains an entire suite of related modules. Fitrix uses `accounting` as application directory names.

- Beneath the application tier is the third tier or module tier. Each module directory contains a set of related input, output, and posting programs. All module directories use a "".4gm extension. For example, you might have a set of Accounts Payable programs in a ap .4gm directory.

- The final and fourth tier is known as the program directory. Each program directory contains a single generated program built by the VDT Application Generator or Report Generator. Fitrix program directories use a "".4gs extension.

- Version control lets you create multiple flavors of a program. Version control is useful when you want to customize base functionality.

- Accounting tables follow a specific naming convention. Each table name is composed of eight characters and the last six characters must be unique. The eight-character name is divided into four sections. The manufacturing application uses a different table naming convention, where names vary in length, and are not named for an owning module, as manufacturing tables are typically used by multiple modules.

Exercise 15

Objective: To create a custom version of your Customer Entry program in a version control directory.

Version control is extremely useful when you need to customize a specific portion of your base program. In this exercise you will modify your Customer Entry program in a version control directory. You will end up with two versions of Customer Entry, but you will only have one code stream.

Create a i_cust.4gc Directory

By default, Fitrix Visual Menus recognizes the *.4gc extension as a version control extension. To create a version control program, you must create a new directory parallel to your program directory. Give this new directory the same name as your program directory, but replace the *.4gs extension with a *.4gc extension.

1. **Use the cd command to move to your aw. 4gm directory.**
2. **Use `mkdir` to create a new directory. Name it `i_cust.4gc`.**
3. **Use cd again to move to `i_cust.4gc`.**

Copy your *.per Files

1. **In your i_cust.4gc directory, copy over all your form specification files (*.per) from i_cust. 4gs:**

   ```
   cp ../i_cust.4gs/*.per .
   ```

2. **Start the Form Painter from your i_cust .4gc directory and open your cust .per file.**
3. **Add a title line to read:**

   Acme Inc Customer Entry Window

   In most cases, you will customize more than the title line. But adding this line adequately demonstrates version control.

4. **Save your changes and exit the Form Painter.**

Generate and Compile

1. **From i_cust.4gc, run the Screen Code Generator.**

2. **When the Generator finishes, use ls to view the files it created.**

   Notice that the Generator creates a whole new set of *.4g1 files and a Makefile.

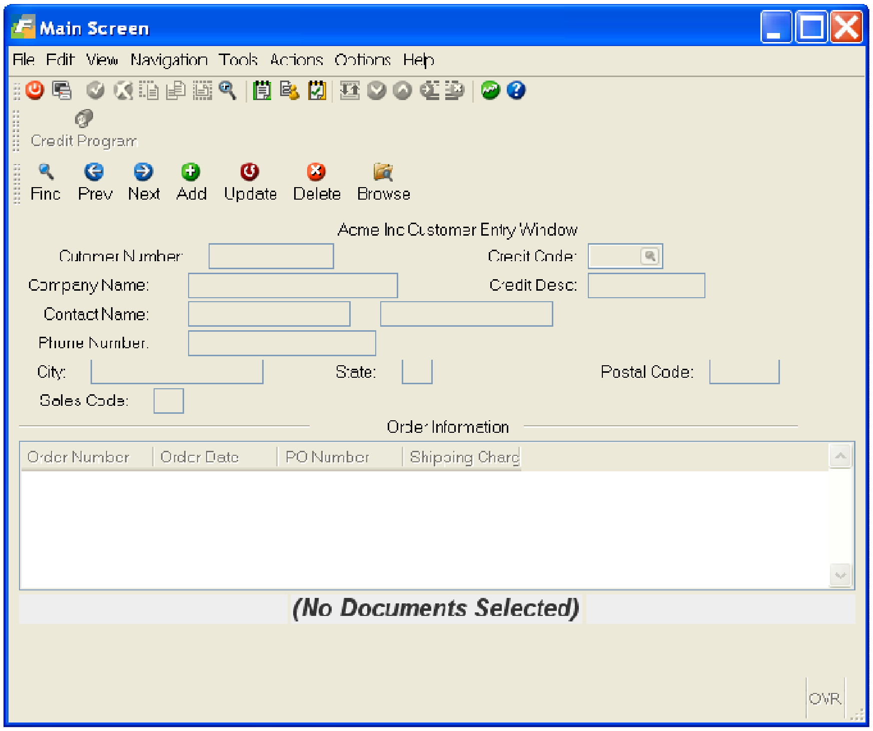
3. **Now run `fg.make` to link and compile the code.**

Run Your Custom Version

After you generate and compile, run your version control program:

- `fglrun i_cust.42r –d <database name>`

Once you initiate Customer Entry, your custom version appears:



(No Documents Selected)

Chapter 16

Compiling Generated Code

Main topics:

- Compiling Generated Code
- Modifying Libraries
- Understanding the Library Philosophy
- Adding a Custom Library

Compiling Generated Code

Compiling generated code means turning 4GL source code and triggers into a runnable program (that is capable of being executed by the ‘fglrun’ runner). Fitrix gives you the ability to do this for a single program or a group of programs.

You compile code using the Make Utility. This utility is run with the `fg.make` command. When you run `fg.make`, it completes the following tasks:

- Merges Custom Code: The fg.make command calls the Featurizer (fglpp) program. The Featurizer merges custom code into your program source code (see "Featurizer Overview" on page 250).
- Compiles Source Code and Form Specification Files: `fg.make` also compiles both your source code (.*4gl) files and form specification (*.per) files.
- Links Local Function Calls to Library Functions: The `fg.make` command resolves library function calls in local (i.e., source code) to their corresponding library functions.
- Produces Runnable Program File: The last task of `fg.make` is to construct a runnable program file. The `fg.make` command creates Genero *.42r.

The final three tasks are controlled by the standard LINUX make utility, which is called by fg.make. In general, the LINUX make utility tracks the dependencies that files have to each other.

The LINUX make utility uses a specification file of its own. This file, called the Makefile contains all the instructions necessary for make to work. You do not have to create the Makefile however. It is created automatically by the VDT Application Code Generator.

For the most part, you do not need a complete understanding of the LINX make utility in order to use it. You should simply realize that it is called from the fg.make command and it produces a program file that you can run.

The `fg.make` command uses a number of command flags:

```
fg.make [-h][-L library] [-m{n|o|f|of}]
[-f] [-a]
```

-h	Prints an entire list of `fg.make` command flags.
-L library	Specifies the name of any additional libraries you want `fg.make` to link in.
-mn	Does everything except merge code. In other words, when you use the -mn flag, the Featurizer is not called.

-mo Runs the Featurizer (merges code) but does not perform a compilation.

-mf Overrides timestamp comparison logic and forces a custom code merge.

-mfo Overrides timestamp comparison logic and forces a custom code merge. This flag does not perform a compilation, however.

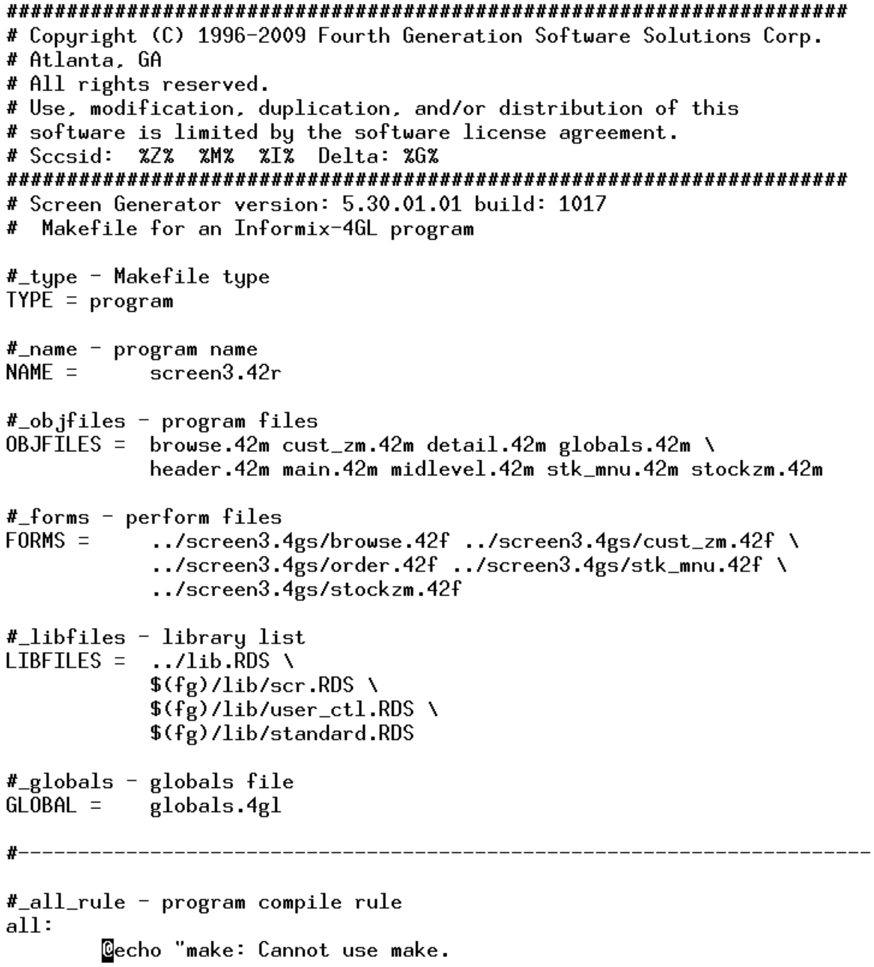
-f Performs a fast link. You should only use this flag in compiles where no new calls to library functions have been added.

-a Causes all files to be compiled regardless of dependencies.

-l Link only.

The Makefile

It does help, though, to have a working knowledge of the `Makefile`. The `Makefile` contains several sections. Each section supplies make with information about your program.



```
######################################################################
# Copyright (C) 1996-2009 Fourth Generation Software Solutions Corp.
# Atlanta, GA
# All rights reserved.
# Use, modification, duplication, and/or distribution of this
# software is limited by the software license agreement.
# Sccsid:  %Z%  %M%  %I%  Delta: %G%
######################################################################
# Screen Generator version: 5.30.01.01 build: 1017
#  Makefile for an Informix-4GL program

#_type - Makefile type
TYPE = program

#_name - program name
NAME =      screen3.42r

#_objfiles - program files
OBJFILES =  browse.42m cust_zm.42m detail.42m globals.42m \
            header.42m main.42m midlevel.42m stk_mnu.42m stockzm.42m

#_forms - perform files
FORMS =     ../screen3.4gs/browse.42f ../screen3.4gs/cust_zm.42f \
            ../screen3.4gs/order.42f ../screen3.4gs/stk_mnu.42f \
            ../screen3.4gs/stockzm.42f

#_libfiles - library list
LIBFILES =  ../lib.RDS \
            $(fg)/lib/scr.RDS \
            $(fg)/lib/user_ctl.RDS \
            $(fg)/lib/standard.RDS

#_globals - globals file
GLOBAL =    globals.4gl

#-----------------------------------------------------------------------

#_all_rule - program compile rule
all:
        @echo "make: Cannot use make.
```

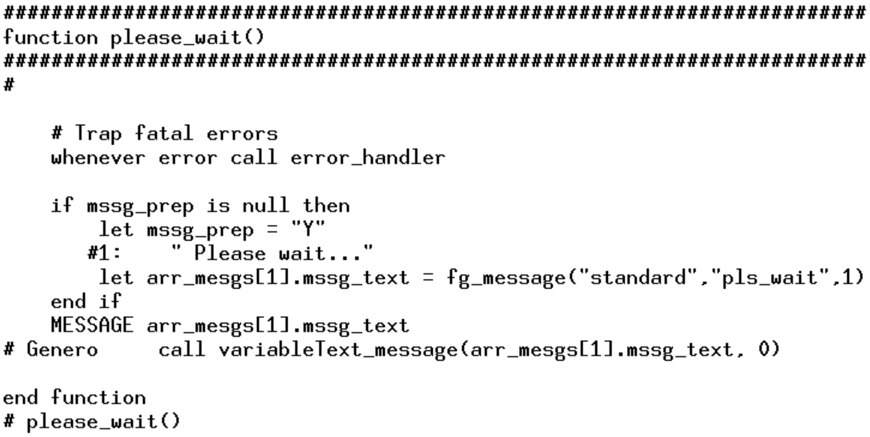
As you can see, the `Makefile` lists the files necessary to create your program. For example, the `LIBFILES` section shows all the libraries used by your program (`lib.RDS, scr.$DS, user_ctl.RDS` and `standard.RDS`).

If cust_path is set, and additional entry is produced in the Makefile called CUST_PATH. The CUST_PATH entry in the Makefile overrides any environment variable setting.

Library Overview

A library holds code shared by multiple programs. The code is structured into functions. Each function performs a single task and works independently from other code. For example, several programs require a message that reads, "Please wait." Instead of duplicating the same lines of code in each program directory, you can simply place a call to the library function that displays the "Please wait" message.

The VDT Application Code Generator makes extensive use of libraries. These libraries are contained in the `$fglibdir/lib` directory. Each library contains related functions. For example, the standard library contains functions shared by both input and output programs, such as the "Please wait" message, which is in the `pls_wait.4gl` file:



```
#########################################################################
function please_wait()
#########################################################################
#

    # Trap fatal errors
    whenever error call error_handler

    if mssg_prep is null then
        let mssg_prep = "Y"
       #1:    " Please wait..."
        let arr_mesgs[1].mssg_text = fg_message("standard","pls_wait",1)
    end if
    MESSAGE arr_mesgs[1].mssg_text
# Genero     call variableText_message(arr_mesgs[1].mssg_text, 0)

end function
# please_wait()
```

Creating Custom Libraries

If you have programs that share common functions that are not in the libraries, you can create your own custom library.

Custom libraries are created at the module directory level (the *.4gs level). Just like the libraries, custom libraries contain functions that perform specific, independent tasks. These functions are placed in source code (*.4gl) file.

For example, to create a custom library called my1ib:

1. **At the program directory level, create a `mylib.4gs` directory.**

   ```
   mkdir mylib.4gs
   ```

2. **Move to mlib`.4gs` and create each custom function in its own source code (*.4gl) file.**

3. **Copy a library Makefile into `mylib.4gs`.**

   ```
   cp $fg/lib/standard.4gs/Makefile mylib.4gs
   ```

   In order to compile your library code there must be a `Makefile` present. You can build a Make file by hand or you can modify the one in the `standard.4gs` library.

4. **Replace the Makefile's section with your function filenames.**

   For example, if `mylib.4gs` contains `wincl.4gl, windl.4gl,` and `winop.4gl` The LIFILES section should read:

   ```
   LIBFILES = \
      $(LIB)(wincl.o) \
      $(LIB)(windl.o) \
      $(LIB)(winop.o)
   ```

5. **Change the `LIB=../standard.a` line to read:**

   ```
   LIB= ../mylib.a
   ```

6. **Finally, run `fg.make` in the `mylib.4gs` directory.**

To create a custom library for an existing fitrix module to add functions and override existing functions in a lib.4gs directory, you can create a lib.4gc directory. When you use the default .4gc extension you need not add the library to a program’s local Makefile. But you must stick to a strict convention for the LIB line in the lib.4gc’s Makefile. You must have the following LIB line in your lib.4gc’s Makefile:

```
LIB= ../lib4gc.a
```

Using a Custom Library

Once you create a custom library, you can use it in your programs. You must add your custom library to the `LIBFILES` section of the program's `Makefile`. In other words, if you call custom library functions in your program code, you must tell the LINUX rnake utility where to look to find the custom library functions.

For example, if your program calls `windl`, which is in your custom `mylib.4gs` library, the `LIBFILES` section must include `mylib.a.`

You can add libraries to your program's `Makefile` using the libraries trigger. For example, the following libraries trigger adds the `mylib` library to your program's `Makefile`:

```
defaults
    libraries
            .. /mylib.a
```

This trigger changes your `Makefile` to look as follows:

```
#_libfiles library list
LIBFILES ../lib.a \
    .. /mylib.a \
    $(fg) /lib/scr.a \
    S(fg) /lib/user_ctl.a \
    $(fg) /lib/standard.a
```

Another trigger, the `custom_libraries` trigger, also lets you add libraries to your program's `Makefile`. The `custom_libraries` trigger places your custom library above the `../lib.a \` line in the `Makefile`. For example, the following `custom_libraries` trigger places your `mylib` library first on the `LIBFILES` list.

```
defaults
    custom_libraries
            ../mylib.a
```

This trigger changes your `Makefile` to look as follows:

```
#_libfiles library list
LIBFILES .. /mylib.a \
    .. /lib.a \
        $(fg)/lib/scr.a \
        $(fg)/lib/user_ctl.a \
        $(fg)/lib/standard.a
```

Section Summary

- Compiling generated code means turning 4GL source code into a runnable program. The VDT Application Code Generator gives you the ability to do this for a single program or a group of programs.
- You compile code using the Make Utility. This utility is run with the `fg.make` command.
- The .make command merges custom code, compiles source code and form specification files, links local function calls to library functions, and produces a rurmable program file.
- The `fg.make` command uses the standard LINUX make utility, which is called by .make. In general, the LINUX make utility tracks the dependencies that files have to each other.
- The `Makefile` contains several sections. Each section supplies make with information about your program. For example, the `LIBFILES` section shows all the libraries used by your program.
- A library holds code shared by multiple programs. The code is structured into functions. Each function performs a single task and works independently from other code.
- The VDT Application Code Generator makes extensive use of libraries. These libraries are contained in the `$fglibdir/lib` directory.
- If you have programs the share common functions that are not in the libraries, you can create your own custom library.
- Custom libraries are created at the module directory level (the *.4gs level). Just like the libraries, custom libraries contain functions that perform specific, independent tasks. These functions are placed in source code (*.4gl) files.
- Once you create a custom library, you can use it in your programs. You must add your custom library, however, to the `LIBFILES` section of your program's `Makefile`. In other words, if you call custom library functions in your program code, you must tell the LINUX make utility where to look to find the custom library function.
- If you create a lib.4gc at the Fitrix module directory level, you need not add the library to your program’s Makefile. You must remember the LIB = ../lib4gc.a line in the lib.4gc’s library Makefile.

Exercise 16A

Objective: To create a custom library and add a function to it.

Create a Library Directory

1. **Use the cd command to move to your aw. 4gm directory.**

2. **Use `mkdir` to create a new directory called `mylib.4gs` and use cd to move to that directory.**

   This is your custom library directory. Within this directory, you can create custom functions for your programs.

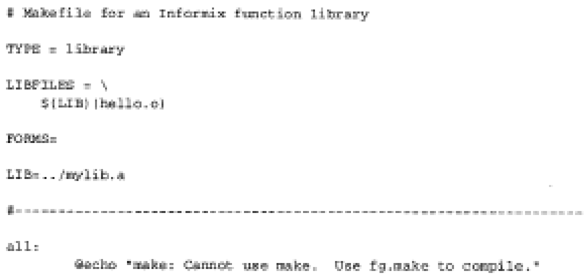
Create a Custom Library Function

1. **Use vi to open a new file called hello.4gl.**

2. **Add the following function to your new file:**

```
function hello ()
    call gn_close("Hello", "Hello fitrix world.")
end function
```

3. **Use vi to create a new Makefile that looks as follows:**



```
# Makefile for an Informix function library

TYPE = library

LIBFILES = \
    $(LIB)(hello.o)

FORMS=

LIB=../mylib.a

#-----------------------------------------------------------------

all:
        @echo "make: Cannot use make.  Use fg.make to compile."
```

4. **While you are still in mylib. 4gs, run fg.make.**

   The `fg.make` script compiles your library and creates a parallel RDS version of your library at the module directory level.

Add a libraries Trigger

To use your new `hello()` function, you must add your custom library to the `Makefile` file in your `i_cust.4gs` directory. A special trigger, called `libraries` lets you do this.

1. **Use cd to move to your `i_cust.4gs` directory.**

2. **Use vi to open your `cust.trg` trigger file.**

3. **Add the following code to the defaults section of cust.trg:**

```
libraries
   ../mylib.a
   ;
```

This trigger adds your custom `mylib` library to the `LIBFILES` list in the program `Makefile`.

4. **Save and quit from `cust.trg`.**

Add a before_input Trigger

To implement your new `hello()` function, you must use it from somewhere in your program. Perhaps the simplest way to use it is with a `before_input` trigger.

3. **Use vi to open cust.trg.**

4. **In the input 1 section, add the following lines of code:**

```
before_input
   call hello();
```

This trigger simply calls your `hello()` function, which is in your custom `mylib` library.

5. **Save and quit from cust.trg.**

Compile the Code

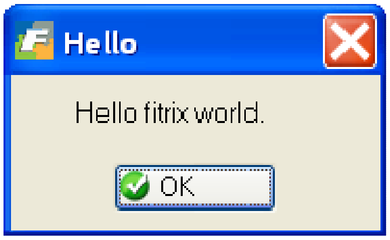
1. **Run fg.make to compile the code.**

Run Your Customer Entry Program

2. **Run your Customer Entry program.**

3. **Select Add from the ring menu.**

What happens? Do you see the "Hello fitrix world" message?



Hello

Hello fitrix world.

4. **Quit from your Customer Entry program.**

Exercise 16B

Objective: To call `hello()` from the Credit Entry program.

Custom libraries allow you to call custom functions from anywhere in your module directory. In other words, custom libraries work with all the programs in your module. You have already used the `hello()` function in your Customer Entry program. Now you will add a call to this function from your Credit Entry program.

Create a cred.trg Trigger File

1. **Use cd to move to the i_cred. 4gs directory.**
2. **Use vi to create a cred.trg file.**
3. **Add the following libraries trigger to cred.trg:**

```
defaults
    libraries
        ../mylib.a
        ;
```

4. **Save and exit cred.trg.**

Add a libraries Trigger

1. **Use vi to open credit.trg.**
2. **Just below your libraries trigger, add the following code:**

```
input 1
    before_input
            call hello () ;
```

Your complete credit.trg file should look as follows:

```
defaults
    libraries
          .. /mylib.a
          ;
input 1
    before_input
          call hello();
```

3. **Save and exit credit.trg.**

Compile the Code

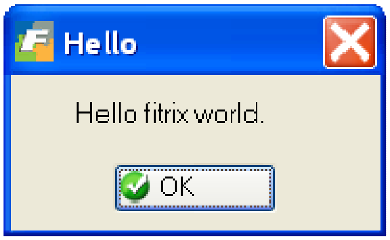
1. **Run fg.make to compile the code.**

Run Your Credit Entry Program

1. **Run your Credit Entry program.**

2. **Select Add from the ring menu.**

   What happens? Do you see the "Hello fitrix world" message?



Hello fitrix world.

3. **Quit from your Credit Entry program.**

Chapter 17

Using the Featurizer

Main topics:

- Featurizer Overview
- Creating Block Commands
- Pluggable Feature Sets
- Triggers Versus Block Commands

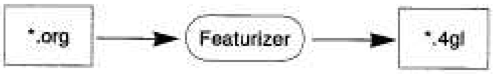
Featurizer Overview

The Featurlzer performs two tasks:

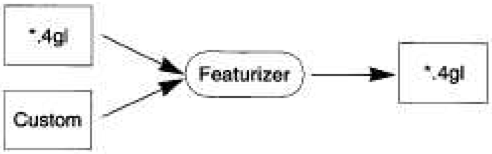
1. **It copies *.org files, which are created by the VDT Application Code Generator, into *.4g1 files.**

2. **After it creates the *.4g1 files, the Featurizer merges the custom code into the source *.4g1 files.**

The Featurizer copies generated *.org files into *.4gl files.



After copying the *.org files, the Featurizer then merges custom code into source code.



Both the VDT Application Code Generator and the `fg.make` command run the Featurizer automatically. You can also run the featurizer directly with the `fglpp` command.

For instance, if you want to merge custom code into `header.4g1`, type:

```
fglpp header.4g1
```

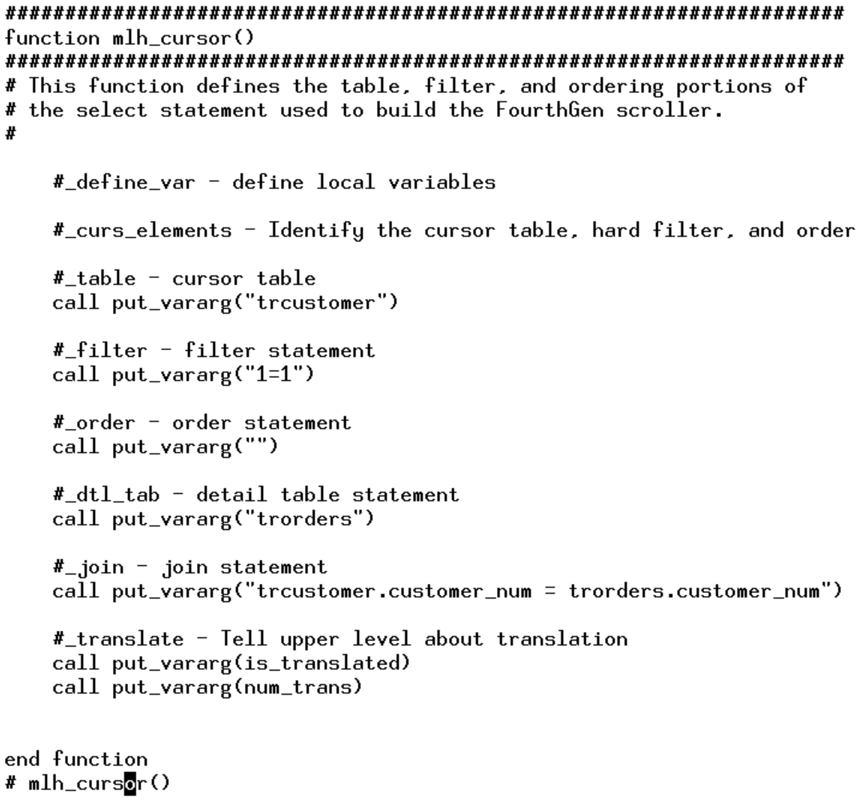
You have already learned how to create custom modifications in trigger (*.trg) files. In addition to trigger files, though, the Featurizer also reads extension files and merges them into your source code (*.4gl) files. Extension (*.ext) files are similar to trigger files, but extension files act on physical locations in source code. Within extension files you create block commands.

Block Commands

Block commands let you customize physical points within generated source code. Because block commands act on physical locations, you must address where you want your block command to go. A code address contains three parts: filename, function name, and block tag.

You already know about filenames and function names, but block tags are a new concept. For example, the `mlh_cursor` function in the `midlevel.4gl` file contains eight block tags. You can easily identify block tags because they all begin with the same two characters (#_) followed by their block name. For example, `#_define_var` is the first block tag in the `mlh_cursor` function:

This function contains eight block tags.



```
########################################################################
function mlh_cursor()
########################################################################
# This function defines the table, filter, and ordering portions of
# the select statement used to build the FourthGen scroller.
#

    #_define_var - define local variables

    #_curs_elements - Identify the cursor table, hard filter, and order

    #_table - cursor table
    call put_vararg("trcustomer")

    #_filter - filter statement
    call put_vararg("1=1")

    #_order - order statement
    call put_vararg("")

    #_dtl_tab - detail table statement
    call put_vararg("trorders")

    #_join - join statement
    call put_vararg("trcustomer.customer_num = trorders.customer_num")

    #_translate - Tell upper level about translation
    call put_vararg(is_translated)
    call put_vararg(num_trans)


end function
# mlh_cursor()
```

Block tags pinpoint physical locations within generated source code. When you want to alter source code contained in a block tag, you can use block commands. Block commands use the following syntax:

```
start file “filename”
```

```
block_command function_name block_tag
```

For example, the following block command adds a line to the `#_define_var` block tag in the `mlh_cursor` function:

```
start file "midlevel.4gl"
   after block mlh_cursor define_var
         tmp_num smallint;
```

The start file command, on the first line, specifies the file to use (in this case it is `midlevel.4gl`).

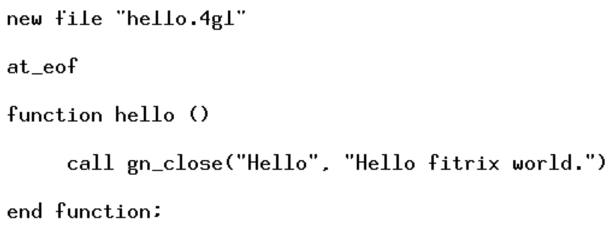
The first argument on the second line is the name of the block command (in this case it is after block). The second argument specifies the function name (`mlh_cursor`). The third argument specifies the block tag minus the #_ characters (`define_var`).

The third line contains your custom code (in this case the third line defines the variable `tmp_nurn`).

You may also generate new .4gl files using the following block command:

```
new file “filename.4gl”
```

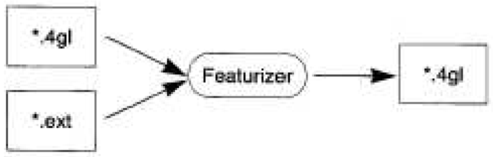
The new file command is useful in libraries. You can contain all new code in extension files using the new file command and the at_oef (at end of file) block command. For example, you could have moved the hello() function in exercise 17A to an extension file using the following extension:



```
new file "hello.4gl"

at_eof

function hello ()

     call gn_close("Hello", "Hello fitrix world.")

end function;
```

You place block commands within extension (*.ext) files, which get read by the Featurizer and merged into your source code.

The Featurizer reads your extension files and merges them into generated source code.



Pluggable Feature Sets

Unlike trigger files, which the Featurizer reads and merges automatically, you must declare extension files within a feature set (`base.set`) file. A feature set file simply contains the names of the extension files you want the Featurizer to merge into your code.

Feature set files are extremely useful because they let you add custom code in a pluggable fashion. For example, you may have three extension files that add custom functionality to your program (`acme.ext, abc.ext,` and `xyz.ext`). Some departments might want the functionality added by all three extension files while others may only want the functionality in the `xyz.ext`.

For your first group of departments, your base. set file would contain the name of all three extension files minus the *.ext extensions:

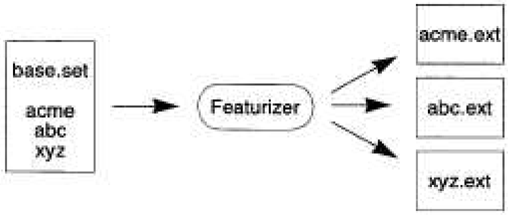
```
acme
abc
xyz
```

For your second group of departments, your `base.set` file would only contain the name of the `xyz.ext` file:

```
xyz
```

When you run the Featurizer, it looks at your base.set file to determine which extension files to merge into your source code.

The Featurizer reads your base.set file to determine which extension files to merge.



Section Summary

- The Featurizer performs two tasks: It copies each *.org file into a *.4g1 source code file and it merges custom code into the *.4gl files.
- Both the VDT Application Code Generator and the `fg.make` command run the Featurizer automatically. You can also run the featurizer directly with the `fglpp` command.
- Block commands let you customize physical points within generated source code. Because block commands act on physical locations, you must address where you want your block command to go. A code address contains three parts: filename, function name, and block tag.
- You can contain new code for a 4gl in an extension file using the new file command and at_eof block command.
- You already know about filenames and function names, but block tags are a new concept. You can easily identify block tags because they all begin with the same two characters (#_) followed by their block name.
- When you want to alter source code contained in a block tag, you can use block commands.
- You place block commands within extension (*.ext) files, which get read by the Featurizer and merged into your source code.
- Unlike trigger files, which the Featurizer reads and merges automatically, you must declare extension files within a feature set (`base.set`) file. A feature set file simply contains the names of the extension files you want the Featurizer to merge into your code.

Exercise 17A

Objective: To use a block command to add a warning message to the Customer Entry program when the contact field is not entered.

There are several fitrix library functions that open dialog boxes useful for messages and warnings. A list of these functions is below.

You have already seen the gn_close dialog box in previous exercises. These functions will not be covered in detail in this class. You will use the gn_yesno_text function for this exercise.

- gn_yesno() - offers a Yes/No dialog box.

  Example

  ```
  LET yes_no=gn_yesno("Date for aging","Use Invoice Date ?","Y")
  ```

  The last argument ("Y") to the utility is the default value when the window displays.

- gn_yesnoquit – Same as gn_yesno, but with a quit option.
- gn_close – displays a message, with an OK option.

  Example

  ```
  CALL gn_close(“Out of Balance”,”Document is out of Balance”)
  ```

- gn_close_text – same as gn_close, but displays multiple lines of text
- gn_yesno_text – same as gn_yesno, but displays multiple lines of text

  Example

  ```
  call textinit()
  call textput(STR.warn_1)
  call textput(STR.warn_2)
  call textput(STR.warn_3)
  call textput(STR.warn_4)
  let prompt_response = gn_yesno_text("Message",4,"Y")
  call textinit()
  if upshift(prompt_response) = upshift(STR.no_val)then
      return
  end if
  ```

- gn_yesnoquit_text – sames as gn_yesnoquit, but displays multiple lines of text
- gn_progress_open, gn_progress_close – displays progress bar, and updates progress of an executing code stream

Build an Extension (*.ext) File

Block commands are created and stored in extension (*.ext) files. In a general sense, extension files are a lot like trigger files. Extension files hold block commands whereas trigger files hold triggers. The major difference is how extension files are merged into base code. As you recall triggers get merged automatically by the VDT Application Code Generator. For extension files, however, you must specify in a feature set file, called `base.set`, which extension files to use.

1. **Use vi to create a new file called warning.ext.**

2. **Add the following block command to warning.ext:**

```
start file "header.4gl"

function_define llh_a_input
    dialog_response char(1) # for response to dialog box
;

before block llh_a_input after_input
    if p_stomer.fname is null
    then
        call textinit()
        call textput("Warning: You have not entered a first “)
        call textput(“name for the contact.")
        call textput("Would you like to enter a contact now?")
        let dialog_response = gn_yesno_text("Warning",3,"Y")
        call textinit()
        if upshift(dialog_response) = "Y"
        then
            let nxt_fld = "fname"
            return
        end if
    end if
;
```

As you can see, this code modifies `header.4gl`. It adds code to add a yes/no dialog box with multiple lines of text. If the user decides to enter the contact first name they select yes and they are returned to the Contact first name field. If they select No to not enter a contact, the customer entry is saved without a first name for the contact.

3. **Save and quit from warning.ext.**

Create a base.set File

In order to incorporate your new block command into base code, you must add warning.ext to the base.set file.

1. **Use vi to create a new file called base.set.**

2. **Add the following line to the base. set file:**

   ```
   warning
   ```

   This is all you need to add. You do not need to include the `.ext` file extension. If you had more extension (*.ext) files to include, you would list them in the same way.

3. **Save and quit from base.set.**

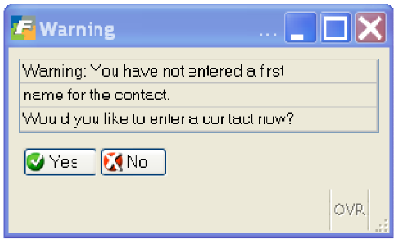
Compile the Code

1. **Run the compilation utility (fg .make).**

Run Your Customer Entry Program

2. **Start your customer entry program.**

3. **Try saving an entry without a contact's first name. You following dialog box is displayed.**



Warning: You have not entered a first
name for the contact.
Would you like to enter a contact now?

4. **Experiment with entering Yes and No.**

5. **Quit out of your Customer Entry program.**

6. **Use vi to look at the llh_after_input function in header.4gl. Notice where the before block command is putting your code. Also notice what the function_define block command does. Exit vi.**

Exercise 17B

Objective: To demonstrate the pluggable feature set concept, you will "unplug" your warning.ext file.

Unplug warning.ext

1. **Use vi to open your base. set file.**

2. **Place a # before the word warning:**

   ```
   # warning
   ```

3. **Save and quit base.set.**

Compile the Code

1. **Run fg.make to compile the code.**

Run Your Customer Entry Program

1. **Start your customer entry program.**

2. **Try saving an entry without a contact's first name.**

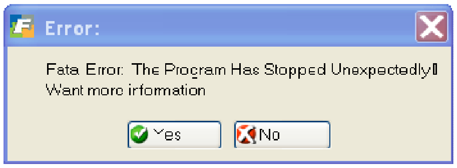
   As you can see, your warning message is now gone. You can add it back by simply removing the comment mark (#) from your `base.set` file.

3. **Quit from your Customer Entry program.**

Exercise 17C

Objective: Add a Required Field with nonull in the .per file.

Have you noticed what happens when you try to add a customer without a contact last name?   You get the error below and the program stops.



Fata Error: The Program Has Stopped Unexpectedly
Want more information

This is because the trcustomer.lname column is defined as not null and you are trying to insert a null into that column. You can automatically require a field with out a lot of custom coding by using the nonull=<field name> statement in the input section of the .per file. You need to add the nonull=<field name> directly to the .per file with vi.

Perform the following steps to add a required field using nonull.

1. **Use vi to add the nonull statement to the input 1 section of the cust.per file as indicated below.**

```
input 1
    table     = trcustomer
    key       = customer_num
    filter    = 1=1
    lookup    = name=cred_lk, key=credit_code, table=trcredit,
      filter=trcredit.credit_code = $credit_code
    zoom      = key=credit_code, screen=cred_zm, table=trcredit,
      from=credit_code
    nonull=lname
```

Note:

You can add multiple fields to nonull by separating the fields with commas like so:

nonull=lname, phone, city, state, zipcode

2. **Save cust.per and exit out of vi**

3. **Regenerate the i_cust.4gs program using fg.screen**

4. **recompile the program using fg.make**

5. **Run and test your changes.**

Chapter 18

Program Events and the Custom Toolbar

Main topics:

- Program Event Overview
- Program Event and Hot Key Tables
- The ON ACTION statement
- The new file command for extension files
- The Custom Toolbar
- The at_eof trigger

Program Event Overview

Program events are either internal or external actions that you can execute from within an input program. You can suspend your input program at any moment and run a program event.

You can add program events to your Navigation menu, which you activate with [CTRL]-[g]. You can also map program events to hot keys and add them to a custom toolbar.

External and Internal Events

As mentioned above, program events are classified either as external events or internal events.

Events that contain LINUX operating system commands constitute external events.

Events that issue Genero 4gl commands are internal. Internal events can be further classified into local and global events or actions.

Local and Global Events

A local event is an internal event that is executable only on one portion of the screen. An event that is "local to the header" can only be executed on the header portion of the screen. Whereas "local to the detail" specifies an event that only takes place on the detail portion of the screen.

A global event is an internal event that is executable from anywhere on the screen. A global event can be executed on the header or detail portions of the screen, from the ring menu, from a zoom screen, an add-on screen, etc.

Program Event and Hot Key Tables

All events and hot keys that you set up are kept in reference tables in the database.

Navigation Event Reference Table

Program events are kept in the Navigation Event Reference table, which goes by the name `stxactnr`.

`language` Holds the language variable for the event, such as [ENG] for English.

`act_key` Holds the event name. When you define events, you specify a value in an Action Code field. Whatever value you specify gets placed in this column.

`description` Holds a description of your program event.

`os_command` Holds the operating system command associated with your program event (for external events only).

`press_enter` Holds a *YIN* value. When your event completes, you can set a prompt to appear before returning to the input program. The "Press Enter to Continue" prompt gives you an opportunity to check error messages if an error occurred during event execution.

`Buttontext` Holds the text that appears when the mouse pointer hovers over a button.

An internal event does not contain a value in the `os_command` column and it sets the press_enter column to N.

Navigation Event Detail Table

All the program events that you set up are also kept in the Navigation Event Detail table, which uses the name `stxnvgtd`. This table specifies what program and user the event is associated with.

`act_key`	Holds the event name. When you define events, you specify a value in an Action Code field. Whatever value you specify gets placed in this column.
`line_no`	Holds the line number value for the program event.
`nav_module`	Holds the module name for the event.
`nav_program`	Holds the program name for the event.
`nav_user`	Holds the user name for the event. This value can be set to all or specify a single user.

Hot Key Definitions Reference Table

The Hot Keys Reference table assigns a unique number to most control keys and function keys. Control keys correspond with the order the letters appear in the alphabet. Function key number start with 101. This table has two fields: `key_code` and `key_desc`. The following list shows some default hot key entries:

key_code	**key_desc**
[2]	[[CTRL]-[b]
[5]	[CTRL]-[e]
[6]	[CTRL]-[f]
[101]	[F1]
[102]	[F2]
[103]	[F3]

Hot Key Definitions Detail Table

The `stxhotkd` table maps program events to control or function keys. It contains the following columns.

`hot_key` Holds the numeric hot key value.

`act_key` Holds the event name. When you define events, you specify a value in an Action Code field. Whatever value you specify gets placed in this column.

`hot_module` Holds the module name for the hot key.

`hot_program` Holds the program name for the hot key.

`hot__user` Holds the user name for the hot key.

`hot_visable` Holds R when the event is visible on the custom toolbar, N when not visable on the custom toolbar.

`hot_bmp` Holds Y if there is a custom bitmap.

`hot_bmpname` Holds the name of bitmap in the ‘pics’ directory under the FourJs product install directory on the workstation. Example: C:\Program Files\FourJs\gdc-fitrix\pics.

on_custom_action code blocks

There are on_custom_action block tags in the following 4GL stement groups:

- INPUT and INPUT ARRAY
- DISPLAY ARRAY
- MENU

They are reserved you to add your custom events. The convention for adding custom events or actions to programs is to add them on a custom toolbar. By adding a custom toolbar to a program you make your custom events/actions more visible to the user. By sticking to conventions used throughout Fitrix, you make your programs more intuitive to users.

The steps to add a customer toolbar are:

1. **Add the event to the event tables.**

2. **Use the `new file` command in a program directory extension file to copy $fglib-dir/lib/scr.4gs/ringMenu.4gl to the local program directory.**

3. **Use block commands in extension files to activate the event on the custom toolbar in local areas (header, detail, add, update) or globally. Below are examples of each circumstance. Substitute your event name for 'custom_action' below.**

- Globally (at the menu outside of add, update find etc)

```
start file "ringMenu.4gl"

after block ringMenu_menu on_custom_action
    ON ACTION custom_action
        <code to run event here>
;
```

- Header Input (add and update)

```
start file "header.4gl"

after block llh_input on_custom_action
    ON ACTION custom_action
        <code to run event here>
;
```

- Header Find

```
start file “midlevel.4gl”

after bock llh_construct on_custom_action
    ON ACTION custom_action
        <code to run event here>
;
```

- Detail Display (View Detail Button)

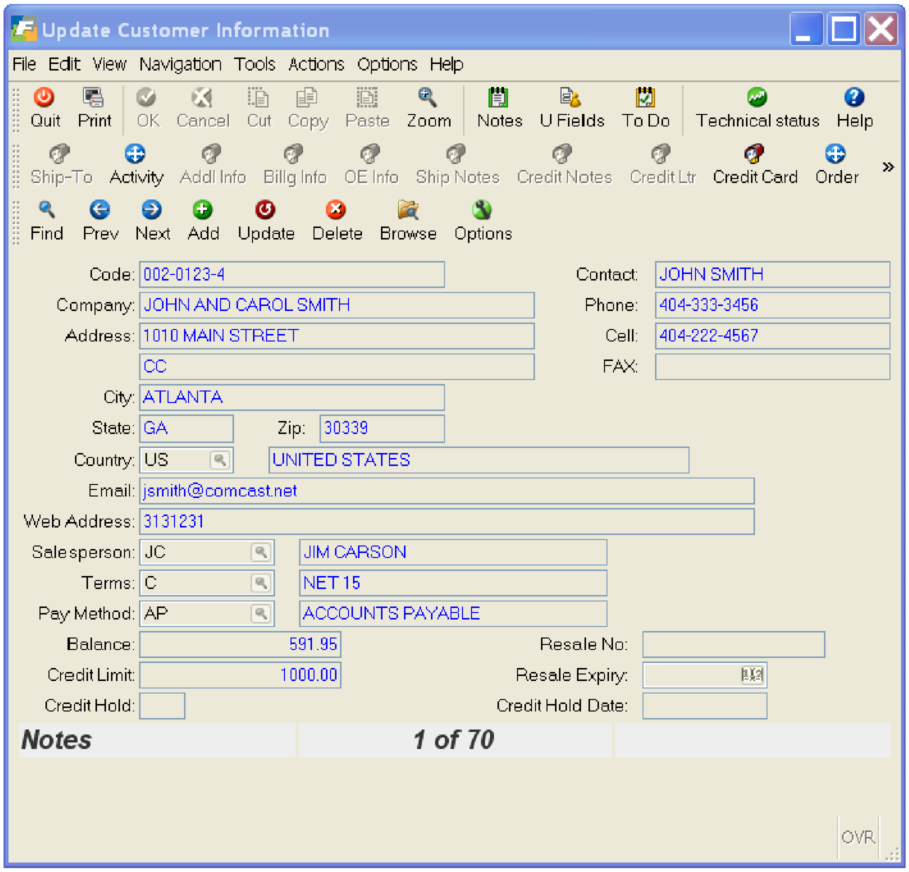
```
start file “midlevel.4gl”

after block mld_scroll on_custom_action
    ON ACTION custom_action
        <code to run event here>
;
```

- Detail Input (Add/Update Detail Button)

```
start file “detail.4gl”
after block lld_input on_custom_action
```

```
    ON ACTION custom_action
        <code to run event here>
;
```

Below is an example of a custom toolbar.



Code: 002-0123-4

Contact: JOHN SMITH

Company: JOHN AND CAROL SMITH

Phone: 404-333-3456

Address: 1010 MAIN STREET

Cell: 404-222-4567

CC

FAX:

City: ATLANTA

State: GA Zip: 30339

Country: US UNITED STATES

Email: jsmith@comcast.net

Web Address: 3131231

Salesperson: JC JIM CARSON

Terms: C NET 15

Pay Method: AP ACCOUNTS PAYABLE

Balance: 591.95

Resale No:

Credit Limit: 1000.00

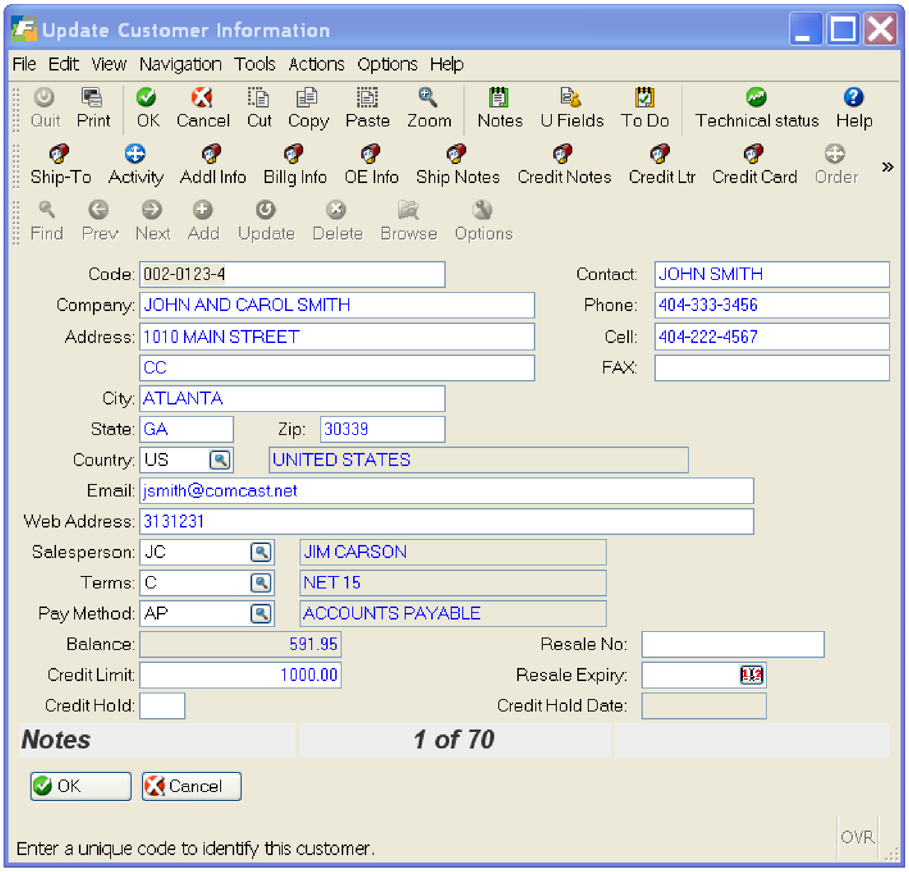
Resale Expiry:

Credit Hold:

Credit Hold Date:

Notes 1 of 70

Notice that some of the buttons are grayed out. Once you go into update mode, you see more buttons activated. The illustration on the following page shows the same program in updated mode. The reason that some buttons are active only during update is because of the placement of ON ACTION code. You will learn more about this during the exercises.


Update Customer Information
File Edit View Navigation Tools Actions Options Help
Quit Print OK Cancel Cut Copy Paste Zoom Notes U Fields To Do Technical status Help
Ship-To Activity Addl Info Billg Info OE Info Ship Notes Credit Notes Credit Ltr Credit Card Order »
Find Prev Next Add Update Delete Browse Options
OK
Cancel
OVR

Code: 002-0123-4
Contact: JOHN SMITH
Company: JOHN AND CAROL SMITH
Phone: 404-333-3456
Address: 1010 MAIN STREET
Cell: 404-222-4567
CC
FAX:
City: ATLANTA
State: GA
Zip: 30339
Country: US UNITED STATES
Email: jsmith@comcast.net
Web Address: 3131231
Salesperson: JC JIM CARSON
Terms: C NET 15
Pay Method: AP ACCOUNTS PAYABLE
Balance: 591.95
Resale No:
Credit Limit: 1000.00
Resale Expiry:
Credit Hold:
Credit Hold Date:

Notes **1 of 70**

Enter a unique code to identify this customer.

The at_eof Trigger

The `at_eof` trigger places whatever you put in it at the end of a file. It is commonly used for putting in functions that your write or library functions that you customize. The following `at_eof` trigger illustrates a custom function:

```
at_eof
######################################################
function IDY_funct()
######################################################
   CALL gn_close(“This is my function.”)
end function
# my_funct
   ;
```

There are three common uses for the `at_eof` trigger:

1. **Adding custom functions.**

2. **Modifying library functions**.

   Library functions exist outside your program directory. They are shared by many programs. H you modify a library functions in the library, you change how it works throughout all your applications. It is much safer to alter library functions in your local directory using the at_eof trigger. Even though this creates two functions with the same name, the function in the local directory takes precedence.

3. **Modifying a locally generated function**.

   By placing a locally generated function into your trigger file you can modify it, but the original function still exists in source code. You must use a `do_not_generate` trigger to keep the original function from generating. For example if you alter the `mlh_clear` function in your trigger file, add the following `do_not_ generate` trigger as well.

```
do_not_generate
   mlh_clear;
```

Section Summary

- Events have instructions attached to them. You can execute events at any time within a program. All events may be viewed, setup, and executed via the Navigation Menu, which you access by typing [CTRL]-[g].
- Any event can be mapped to a hot key. Once mapped to a hot key, the user can press the hot key to execute the event.
- External events have operating system instructions attached to them.
- Internal events have 4GL instructions attached to them.
- Local events are internal events that can only be executed on one portion of the screen (either the header or detail).
- Global events are internal events that can be executed on any portion of the screen: header, detail, or ring menu.
- All events are set up as rows in two tables: the Navigation Event Reference table (`stxactnr`) and the Navigation Event Detail table (`stxnvgtd`).
- All hot key mappings are set up as rows in two tables: the Hot Key Definition Reference table (`stxkeysr`) and the Hot Key Definitions Detail table (`stxhotkd`).
- Local events should be placed on a custom toolbar for easy user access. You add an event to the custom toolbar in 3 steps:

  1. Add the event to the event tables.
  2. Use new file in a program directory extension file to copy $fglibdir/lib/scr.4gs/ringMenu.4gl to the local program directory.
  3. Use block commands in extension files to activate the event on the custom toolbar

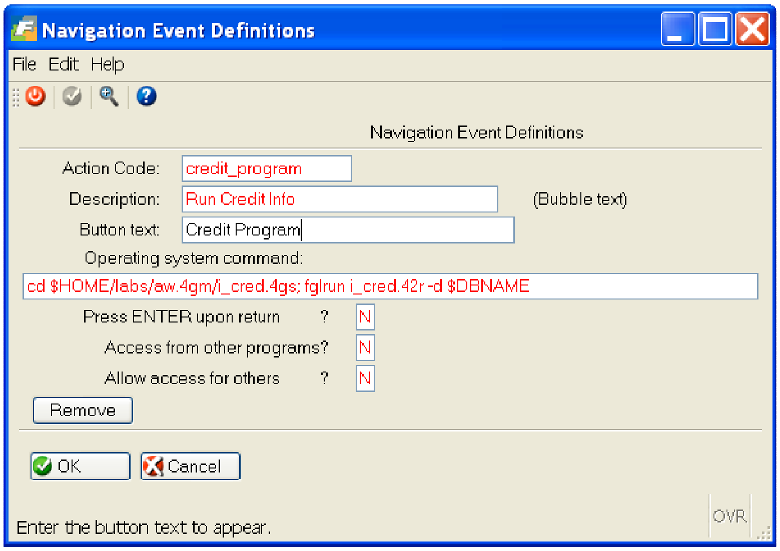
- The `at_eof` trigger is used to add code at the end of a file. You can modify library functions so that they behave a certain way just for the program you are running. You can even use the `at_eof` trigger to modify a function generated in local code, but if you do, you must use the `do_not_generate` trigger to prevent the original function from being created.

Exercise 18A

Objective: To add a simple internal event to your Customer Entry program and add it to a custom toolbar. The event created in chapter 7 is used.

Change a Navigation Event

1. **Start your Customer Entry program. Press [CTRL]-[g] to bring up the Navigation Events. Highlight credit_program and press [CTRL]-[z]. Change Button text to ‘Credit Program’ and save your entry.**



Action Code: credit_program

Description: Run Credit Info (Bubble text)

Button text: Credit Program

Operating system command:

cd $HOME/labs/aw.4gm/i_cred.4gs; fglrun i_cred.42r -d $DBNAME

Press ENTER upon return ? N

Access from other programs? N

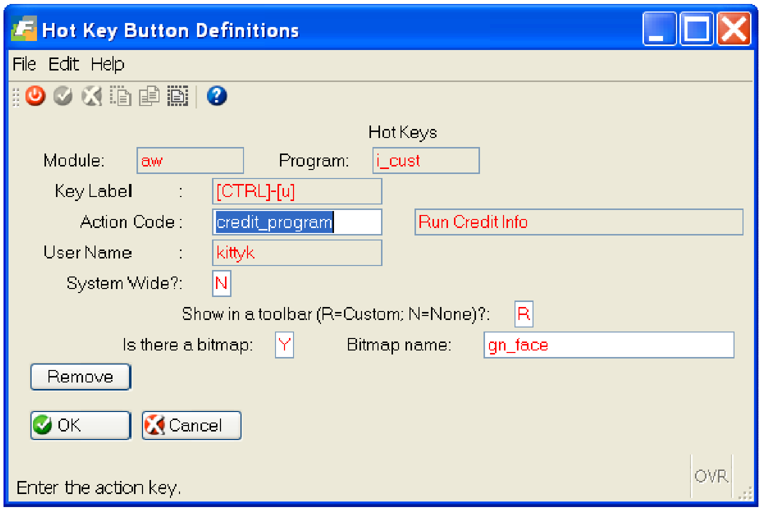
Allow access for others ? N

Enter the button text to appear.

2. **Press [CTRL]-[e] to display Hotkeys.**

3. **Scroll down to find and select [CTRL]-[u] Run Credit Info Hotkey definition. Press [CTRL]-[z] to edit the hotkey definition.**

   The Hotkey Button Definition window appears. Change Show in a toolbar (R=Custom; N=None)? field to ‘R’. Change Is there a bitmap to ‘Y’ and set Bitmap name to ‘gn_face’.



Hot Keys

Module: aw Program: i_cust

Key Label : [CTRL]-[u]

Action Code : credit_program Run Credit Info

User Name : kittyk

System Wide?: N

Show in a toolbar (R=Custom; N=None)?: R

Is there a bitmap: Y Bitmap name: gn_face

Enter the action key.

Save your entry. You now have 3 rows in your database for this custom event. You need to save these rows to the $fg/data/sql.4gc directory for future reference. The best way to organize the $fg/data/sql.4gc directory is by table name. Create the following files in $fg/data/sql.4gc:

stxactnr.sql:

insert into stxactnr values

("ENG","credit_program","Run Credit Info","cd $HOME/labs/aw.4gm/i_cred.4gs; fglrun i_cred.42r -d $DBNAME","N","Credit Program");

stxnvgtd.sql:

insert into stxnvgtd

values ("credit_program","","aw","i_cust","");

stxhotkd.sql:

insert into stxhotkd

values (21,"credit_program","aw","i_cust","","R","Y","gn_face");

Add Code for ringMenu.4gl to an Extension File

1. **Add a new extension file to labs/aw.4gm/i_cust.4gs. Call it ringMenu.ext. The easiest way to do this is to copy $fglibdir/lib/scr.4gs/ringMenu.4gl to ringMenu.ext (cp `$fglibdir/lib/scr.4gs/ringMenu.4gl ringMenu.ext`). Then modify ringMenu.ext. At the very top of the file add the following lines:**

```
new file “ringMenu.4gl”
at_eof
```

At the very bottom of the file add a semicolon (;) to terminate the block.

2. **Add ringMenu to the base.set file at the bottom.**

Add Code to Add Action to Custom Toolbar

3. **Add a new extension file called toolbar.ext. Add the following code to toolbar.ext**

```
start file "header.4gl"

after block llh_input on_custom_action
    ON ACTION credit_program
       let scratch = "cd ../i_cred.4gs; fglrun i_cred.42r -d $DBNAME"
       run scratch
;
```

4. **Add toolbar to base.set at the bottom.**

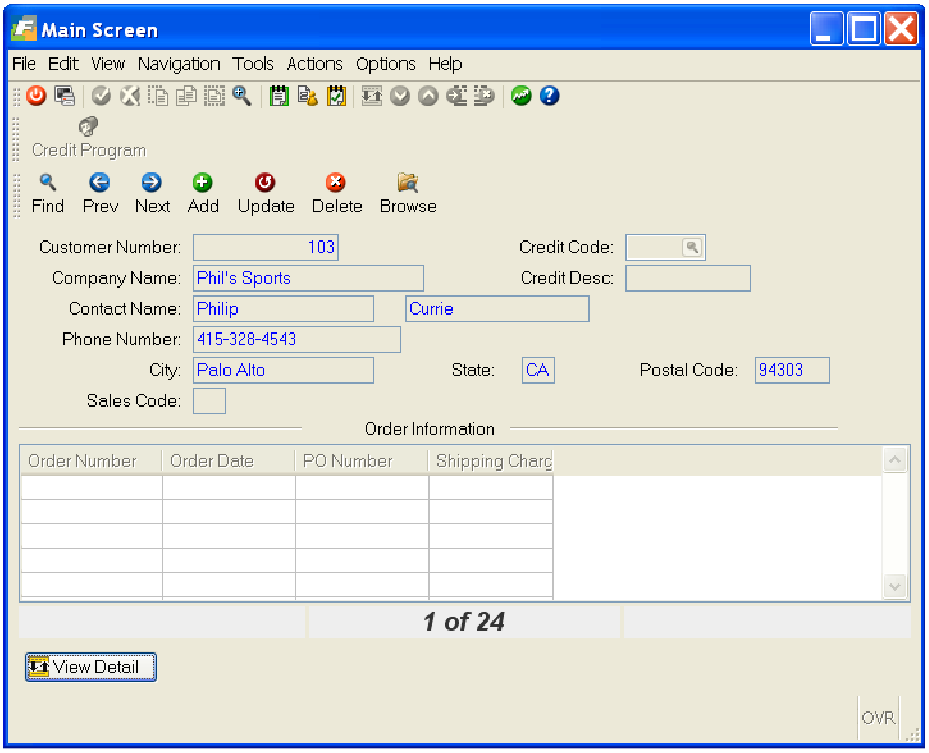
Compile and Run Customer Entry Program

1. **Merge and compile the code with the fg.make utility.**

2. **Run your program with fglrun.**

Update a Record

1. **Use find to select a customer record. Notice that the Credit Program button is inactive.**


Main Screen
File Edit View Navigation Tools Actions Options Help
Credit Program
Find Prev Next Add Update Delete Browse
View Detail
OVR

Customer Number: 103
Company Name: Phil's Sports
Contact Name: Philip Currie
Phone Number: 415-328-4543
City: Palo Alto State: CA Postal Code: 94303
Sales Code:
Credit Code:
Credit Desc:

Order Information

Order Number	Order Date	PO Number	Shipping Charg

1 of 24

2. **Update the customer. Notice that the Credit Program button is now active. Give it try.**

3. **Click the Detail Button to move to the detail portion of the window. Notice that the Credit Program button is now inactive. This is because of where you put the local event code.**

4. **Exit the customer entry program.**

Exercise 18B

Objective: Add a global Credit Program event to the customer entry program.

1. **Edit the toolbar.ext file adding the following code.**

```
start file "ringMenu.4gl"

after block ringMenu_menu on_custom_action
    ON ACTION credit_program
       let scratch = "cd ../i_cred.4gs; fglrun i_cred.42r -d $DBNAME"
       run scratch
;
```

2. **Merge and compile the program using the fg.make utility.**

3. **Run Customer Entry. Find a customer and update the customer. Notice when the Credit Program button is active. When you are finished, quit from Customer Entry.**

Exercise 18C

Objective: To use the at_eof trigger to disable a ring menu option.

You will use the `at_eof` file trigger to add custom logic to the `ok_delete` function. This function is called when the user selects the Delete ring menu command.

Under normal conditions, the `ok_delete` function returns true. You are going to alter `ok_delete` so that it returns false.

Add an at_eof Trigger

4. **Use vi to open your cust.trg file.**

5. **In the defaults section add the following at_eof trigger.**

```
at_eof
       function ok_delete()
           CALL gn_close("Cannot Delete","You are unable to delete a
record")
           return false
       end function;
```

6. **Save your trigger and compile the code.**

7. **Run Customer Entry.**

8. **Select a record and then try to delete it.**

   What do you see? Your message should appear in a dialog box with an OK button.

Chapter 19

Getting Started with FitrixVisual Menus

Main topics:

- Benefits of Fitrix VisualMenus
- Tables Used by Menus
- Menus Structure
- Starting a Menus Program

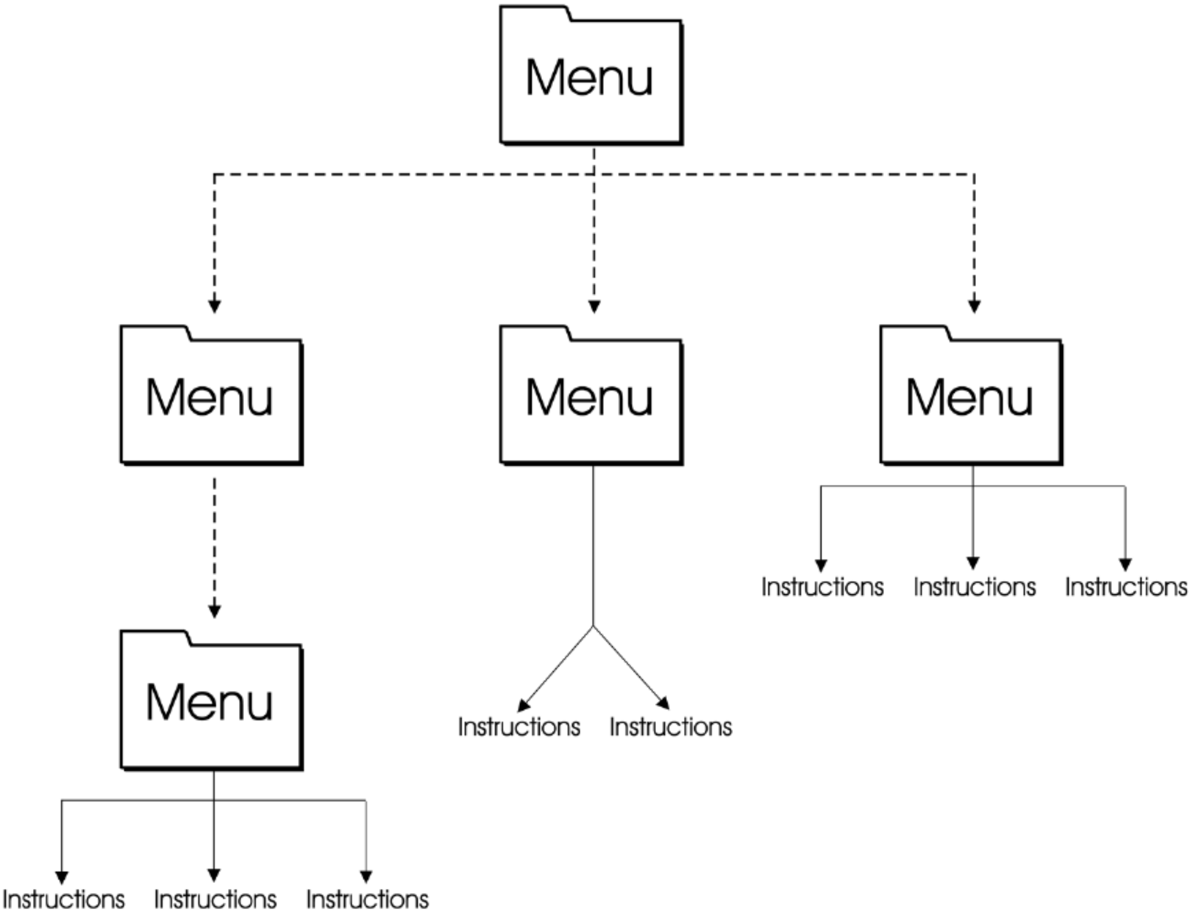
Benefits of Fitrix Visual Menus

Fitrix Visual Menus provides an attractive environment for users to run programs you create with the Fitrix CASE Tools. Fitrix Visual Menus:

- Graphical in nature
- Simple to create and modify.
- Displayable as layers of menus or as a tree structure
- Options can be selected via keyboard or mouse
- Supports concurrent execution of multiple menu options
- Menu definitions are stored in the database

Menu Structure

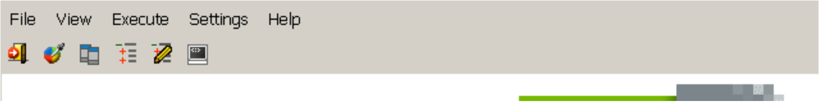
Fitrix Visual Menus are hierarchical in nature and uses the file folder metaphor familiar to Windows users. The file folder corresponds to a menu and provides a way of branching. Consider the following diagram.



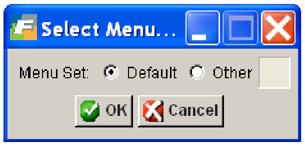
Running Visual Menus in Edit Mode

Fitrix Visual Menus can be run in Classic mode and Explorer modes. Both of these modes execute menu options. Fitrix Visual Menus can also be run in edit mode. Your logged in user name must be in the ‘root’ user group in order to enter edit mode. You can use

Edit mode to add, update and delete menu options. To switch to edit mode, you choose the Edit option of the View pull down menu. You can also enter edit mode by clicking on the ✎ button.



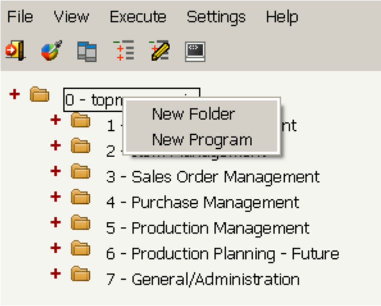
When you enter Edit mode, you are presented with the following dialog.



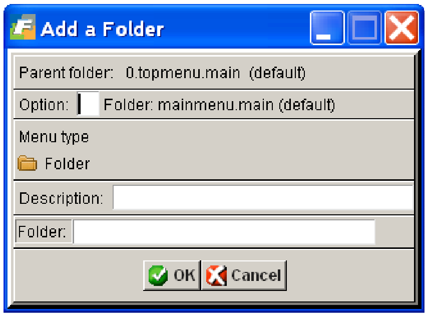
Menu Set: ● Default ○ Other

This dialog box is asking you for the menu set to use. Choose the default menu set when modifying Fitrix ERP visual menus. Choose other and enter a 3 character menu set designation if you are creating a new menu structure.

In Edit mode, to add a menu item to a menu, you right click the menu's folder or text. When you right click you are asked if you want to add a New Folder or a New Program.


File
View
Execute
Settings
Help
0 - topn
New Folder
New Program
1 -
nt
2 -
3 - Sales Order Management
4 - Purchase Management
5 - Production Management
6 - Production Planning - Future
7 - General/Administration

If you choose a new folder, you get the following dialog box.



Parent folder: 0.topmenu.main (default)

Option: Folder: mainmenu.main (default)

Menu type

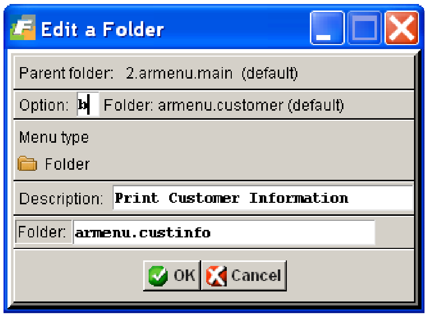
Folder

Description:

Folder:

For a new folder, enter a unique accelerator key. The description is what appears on the menu next to the accelerator key. You should follow conventions when entering the

Folder name. The convention is “parentmenu.childmenu”. Consider the definition for the Print Customer Information Folder/Menu in Accounts Receivable.



Parent folder: 2.armenu.main (default)

Option: b Folder: armenu.customer (default)

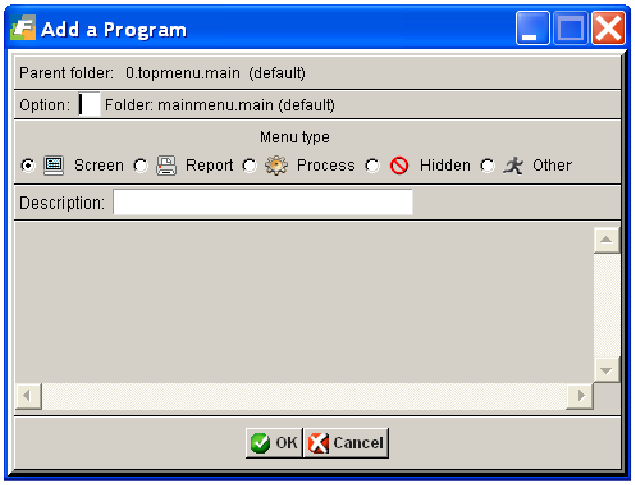
Menu type

Folder

Description: Print Customer Information

Folder: armenu.custinfo

If you choose New Program you get the following dialog box.



Parent folder: 0.topmenu.main (default)

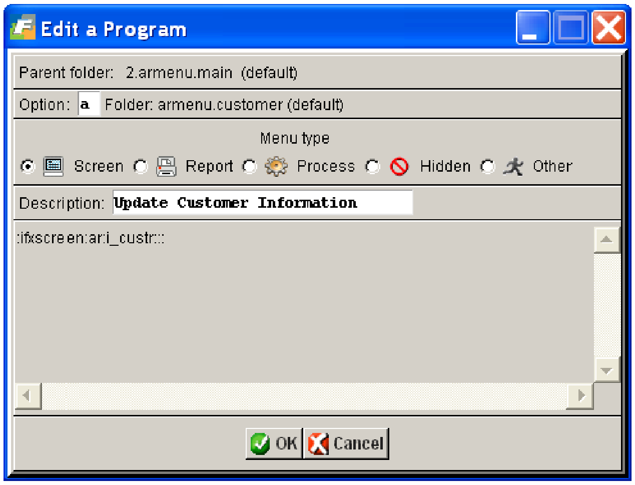
Option: Folder: mainmenu.main (default)

Menu type

● Screen ○ Report ○ Process ○ Hidden ○ Other

Description:

Here you also need to choose character which can be entered via the keyboard to execute the menu option. This character must be unique for each entry within a menu. You also choose the icon that is displayed to the user, as well as the description that appears on the menu. The text area below the Description is for the menu item instructions. The menu item instructions determine what happens when the user chooses the option. You will learn more about these menu item instructions later in this chapter. Below is an example of a program definition.



Parent folder: 2.armenu.main (default)

Option: a Folder: armenu.customer (default)

Menu type

● Screen ○ Report ○ Process ○ Hidden ○ Other

Description: Update Customer Information

:ifxscreen:ar:i_custr:::

All menuing information is kept in reference tables in the database. When you update menus in edit mode, you are also changing data in the database. The Fitrix Visual Menus tables are cgsmnitm, cgsmncmd and cgsmnsec.

Menu Item Table - cgsmnitm

The `cgsmnitm` table contains display information about a menu item. You can consider `cgsmnitm` the header table and `cgsmncmd` the detail table. The `cgsmncmd` table contains the following columns.

`ckey` Custom Key. Blank (“”) if not custom. Only populate ckey if you wish to have an entirely custom menu.

`mname` Contains the menu name. The menu name contains hierarchy information for the menu item. Format is parent-menu.childmenu.

`opt` Contains the accelerator key/keyoard key for the menu option. For example, the accelorator key for the menu option ‘a Update General Journal’ is ‘a’. The opt column must be unique within a menu.

`mtype` Determines the icon type for the menu option.

FL = folder = 📁

SC = screen = 🖥

RP = report = 🖨

OT = other = 🏃

HD = hidden =

PR = process = ⚙

`txt`

Holds the text displayed for the menu item. This appears after the accelerator key.

Menu Command Table - cgsmncmd

The `cgsmncmd` table contains menu item command information about a menu item. It contains the following columns.

`ckey`

Custom Key. Blank (“”) if not custom. Only populate ckey if you wish to have an entirely custom menu.

`mname`

Contains the menu name. The menu name contains hierarchy information for the menu item. Convention is parentmenu.menu. This is a join column to cgsmnitm

`opt`

Contains the accelerator key/keyoard key for the menu option. For example, the accelerator key for the menu option ‘a Update General Journal’ is ‘a’. The opt column must be unique within a menu. This is a join column to cgsmnitm.

`ilevel`

Sequence to run the commands (cmd).

`cmd`

Menu item command. Tells visual menus what to do when this menu item is selected.

Menu Security Table – cgsmnsec

The `cgsmnsec` table contains menu item security information about a menu item. You learn more about security in the next chapter. The `cgsmnsec` table contains the following columns.

`ckey` Custom Key. Blank (“”) if not custom. Only populate ckey if you wish to have an entirely custom menu.

`mname` Contains the menu name. The menu name contains hierarchy information for the menu item. Convention is parentmenu.menu. This is a join column to cgsmnitm

`opt` Contains the accelerator key/keyoard key for the menu option. For example, the accelerator key for the menu option ‘a Update General Journal’ is ‘a’. The opt column must be unique within a menu. This is a join column to cgsmnitm.

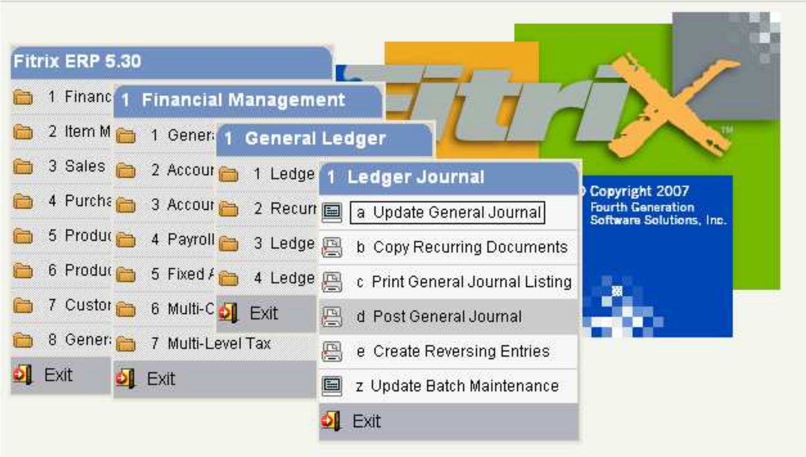
`either_id` Either a group id or a user id.

`allow_flag` Allow execution or deny execution.

Menu item instruction commands – cgsmncmd.cmd

The cmd column of `cgsmncmd` contains the menu item instruction command. Menu item instruction commands are a series of commands specific to a menu item. When the user selects a choice off of the menu, the item instruction(s) in the respective `cgsmncmd` table are executed.

Consider what happens when you choose ‘a Update General Journal’ below.



Corresponding rows in cgsmncmd are selected by the menu item. Consider the `cgsmncmd` row shown below.

```
ckey
mname    glmenu.journal
opt      a
ilevel   1
cmd      :ifxscreen:gl:i_genjrn:::
```

The `cmd` column determines the commands executed.

The `ifxscreen` item instruction command instructs Fitrix Visual Menus to run the input program gl.4gm/i_genjrn. The commands fields are delimited with colons. The first field determines the instruction. The subsequent fields are different and depend on the first field. The first field in the command above indicates that the program to execute is an input program. The next field is the module and the third field is the program name (without the extension). Item instruction commands and their arguments are always delimited by colons.

The following list shows some commonly used item instruction commands and their meaning:

`:ifxscreen:` Runs an input program.

`:item:` Shows menu arguments to user.

`:menu:` Calls up a subordinate menu.

`:env:` Sets a LINUX environment variable.

`:pause:` Displays a window to redirect report output or to continue with menu item.

`:show:` Displays text to the user when they select a menu option.

:ifxreport: Runs a report program.

The two most common program menu items are input programs and reports.

The command for starting an input program is `ifxscreen.` An example of the command to start the Customer Information program is below.

```
:ifxscreen:ar:i_custr:::
```

Notice that the command is delimited by colons and is in all lower case. The format of the command is:

```
:ifxscreen:module:program:flags:x:
```

An explanation of the fields in the ifxscreen command are below.

ifxscreen Indicates that an input program is to be run

module The module directory without the .4gm beneath the accounting directory.

program Name of the program to be run with out the extension. Example, i_custr

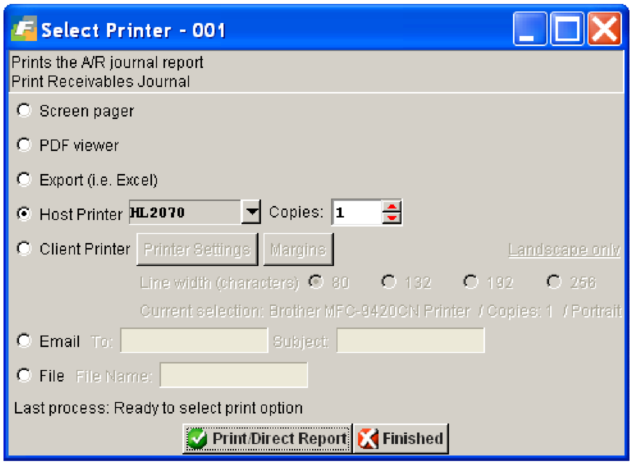
flags If the program should be run with any additional parameters, you add them in the flags field. This field is optional.

x The ‘x’ field is optional. If you have an ‘x’ in the x field, if the input program exits abnormally, subsequent menu item instructions are not executed.

There are typically several menu item instruction commands for report programs. These are :show:, :pause:, and :ifxreport:. Below are item instructions for printing Receivable Journal.

```
:show:Print Receivables Journal:
:show:Prints the A/R journal report:
:pause:p:
:ifxreport:ar:o_arjrnl::default:::
```

These instructions result in the following redirect output window.



Prints the A/R journal report
Print Receivables Journal

○ Screen pager
○ PDF viewer
○ Export (i.e. Excel)
● Host Printer HL2070 Copies: 1
○ Client Printer
Line width (characters) ● 80 ○ 132 ○ 192 ○ 256
Current selection: Brother MFC-9420CN Printer / Copies: 1 / Portrait
○ Email To: Subject:
○ File File Name:

Last process: Ready to select print option

The format of the show command is: `:show:text:`

`show` Use the show command to display information to the user before they run a report.

`text` This is the text to show to the user. You can have multiple show lines for a report. Notice the text of the show command appears at the top of the redirect output window.

The format of the pause command is: `:pause:flag:`

`pause` The pause command displays the ‘Select Printer’ window. The window displayed is determined by the flag field.

`flag` p = Show the select printer window where you can redirect the output of the report.

x = Show the continue window. Displays the show text and gives the user the opportunity to exit. No redirection of output.

The format of the ifxreport command is:

```
:ifxreport:module:program:flags:destination:x:
```

`ifxreport` Indicates a report program is to be run.

`module` The module directory without the .4gm beneath the accounting directory.

`program` Name of the program to be run with out the extension. Example, i_custr

`flags` If the program should be run with any additional parameters, you add them in the flags field. This field is optional.

x

The ‘x’ field is optional. If you have an ‘x’ in the x field, if the report program exits abnormally, subsequent menu item instructions are not executed.

Menu Structure in `cgsmnitm` and `cgsmncmd`

The table cgsmnitm table can be viewed as a header menu table and the cgsmncmd table as the detail table. Together they determine the menu structure. The join for the 2 tables is as follows:

```
cgsmnitm.mname = cgsmncmd.mname and
cgsmnitm.opt = cgsmncmd.opt and
cgsmnitm.ckey = cgsmncmd.ckey
```

Consider the following example. The bold items show the menu path to the Update General Journal program.

cgsmnitm.mname	opt	mtype	cgsmnitm.txt	cgsmncmd.cmd
mainmenu.main	**1**	**FL**	**Financial Management**	**:menu:v530_fmmenu.main:**
mainmenu.main	2	FL	Item Management	:menu:v530_immenu.main:
mainmenu.main	3	FL	Sales Order Management	:menu:v530_somenu.main:
mainmenu.main	4	FL	Purchase Management	:menu:v530_pmmenu.main:
mainmenu.main	5	FL	Production Management	:menu:v530_prmmenu.main:
mainmenu.main	6	FL	Production Planning - Future	:menu:v530_plnmenu.main:
mainmenu.main	7	FL	General/Administration	:menu:v530_gamenu.main:
v530_fmmenu.main	**1**	**FL**	**General Ledger**	**:menu:glmenu.main:**
v530_fmmenu.main	2	FL	Accounts Receivable	:menu:armenu.main:
v530_fmmenu.main	3	FL	Accounts Payable	:menu:apmenu.main:
v530_fmmenu.main	4	FL	Payroll	:menu:pymenu.main:
v530_fmmenu.main	5	FL	Fixed Assets	:menu:famenu.main:
v530_fmmenu.main	6	FL	Multi-Currency	:menu:mcmenu.main:
v530_fmmenu.main	7	FL	Multi-Level Tax	:menu:mainmenu.mtax:
glmenu.main	**1**	**FL**	**Ledger Journal**	**:menu:glmenu.journal:**
glmenu.main	2	FL	Recurring Documents	:menu:glmenu.recur:
glmenu.main	3	FL	Ledger End of Period	:menu:glmenu.eom:
glmenu.main	4	FL	Ledger Setup	:menu:glmenu.glsetup:
glmenu.journal	**a**	**SC**	**Update General Journal**	**:ifxscreen:gl:i_genjrn:::**
glmenu.journal	b	RP	Copy Recurring Documents	:show:Copy Recurring Documents: :show:selects those recurring documents that have been marked for copying: :show:copies marked recurring documents to the general journal: :pause:x: :ifxreport:gl:p_stdent:filter 'stgstder.post_type = "A" or stgstder.post_type = "Y"':default:::
glmenu.journal	c	RP	Print General Journal Listing	:show:Print General Journal Listing: :show:selects unposted documents from the general journal: :show:prints the preposting list: :pause:p: :ifxreport:gl:p_genjrn::default:::
glmenu.journal	d	RP	Post General Journal	:show:Post General Journal: :show:selects all unposted general journal entries:

				:show:posts selected entries to the general ledger activity file:
				:pause:p:
				:ifxreport:gl:p_genjrn:-p:default:::
glmenu.journal	e	RP	Create Reversing Entries	:show:Create Reversing Entries:
				:show:selects all posted general journal entries:
				:show:create a reverse entry in the journal:
				:show:entry file for every document where:
				:show:EOP Reverse field is equal to Y:
				:pause:p:
				:ifxreport:gl:p_newpr2::default:::
glmenu.journal	z	SC	Update Batch Maintenance	:ifxscreen:all:i_batch:-gl::::

Saving Customized Menu Data

You must save any menu data you customize to the $fg/data/sq1.4gc directory. This documents your changes so that you can reapply them after an update. You write the sql as insert statements. For example, if you added customized General Journal Summary Listing program (o_jrnsum.4gc) to the Ledger Journal menu you might have the following sql an a vm.sql file. Notice that the ckey column is not null but set to a space.

```
insert into cgsmnitm (ckey,mname,opt,mtype,txt)
   values (" ","glmenu.journal","f","RP",
     "Print General Journal Summary");
insert into cgsmncmd (ckey,mname,opt,ilevel,cmd)
  values (" ","glmenu.journal","f","1",
       ":show:Print General Journal Summary:");
insert into cgsmncmd (ckey,mname,opt,ilevel,cmd)
  values (" ","glmenu.journal","f","2",
      ":show:Prints the summary page of GL entries.:");
insert into cgsmncmd (ckey,mname,opt,ilevel,cmd)
  values (" ","glmenu.journal","f","3",
      ":ifxreport:gl:o_jrnsum::default:::"");
```

Section Summary

- Fitrix Visual Menus provides an attractive graphical environment for users to run programs.
- You can run Fitrix Visual Menus in edit mode to add and modify menu options.
- For each item on a menu, there are corresponding rows in the cgsmnitm and cgsmncmd tables. The join columns to these tables are mname, opt and ckey.
- When the user selects a menu item, the corresponding item instructions in the cgsmncmd table are executed.
- The item instructions in the cgsmncmd table are simple instructions that tell Fitrix Visual Menus what to do.
- The ifxscreen item instruction runs an input program.
- The :pause:p: item instruction displays the redirect report window
- The ifxreport item instruction runs a report program.
- You must save customized menu options in the $fg/data/sql.4gc directory.

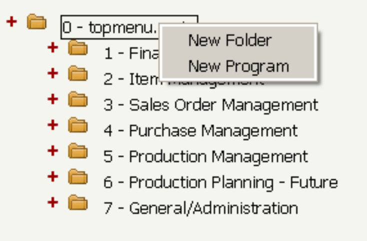
Exercise 19

Objective: To add a simple menuing front-end to the standard menus that starts your Customer Entry and credit entry programs.

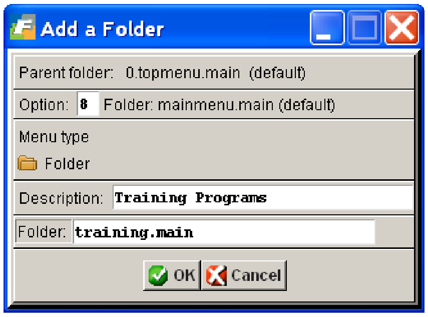
1. **Switch to Edit mode in Fitrix Visual Menus. You can do this by clicking on the on the toolbar.**

2. **Right click on the topmost folder (0 – topmenu.main)**

3. **Choose New Folder.**


0 - topmenu.
1 - Fina
2 - Item
3 - Sales Order Management
4 - Purchase Management
5 - Production Management
6 - Production Planning - Future
7 - General/Administration
New Folder
New Program

4. **Enter the following and click OK to save your work.**



Parent folder: 0.topmenu.main (default)

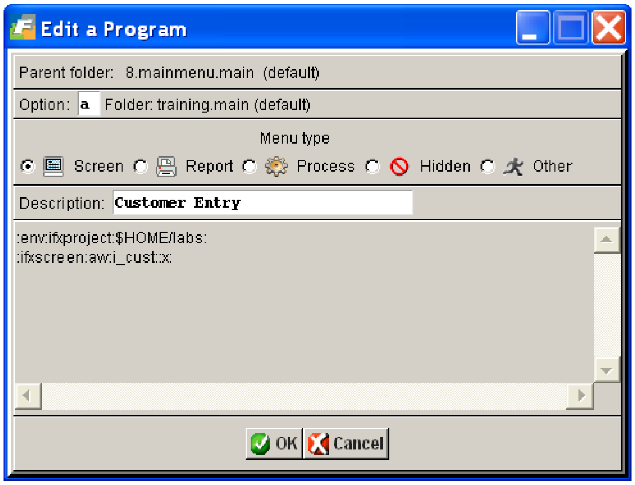
Option: 8 Folder: mainmenu.main (default)

Menu type: Folder

Description: Training Programs

Folder: training.main

5. **You now have a new Training Programs menu option at the bottom of the main or root menu. Right click on Training Programs and choose New Program. Fill out the Add a Program Window as displayed below. Notice the env command below. When you set `ifxproject` variable, you are determining the root directory structure that Fitrix Visual Menus uses to run programs. Usually `ifxproject` is set to $fg/accounting. But you have created the labs under your home directory in $HOME/labs. Setting the `ifxproject` variable using the env command temporarily sets ifxproject to $HOMe/labs for this menu option.**



Parent folder: 8.mainmenu.main (default)

Option: a Folder: training.main (default)

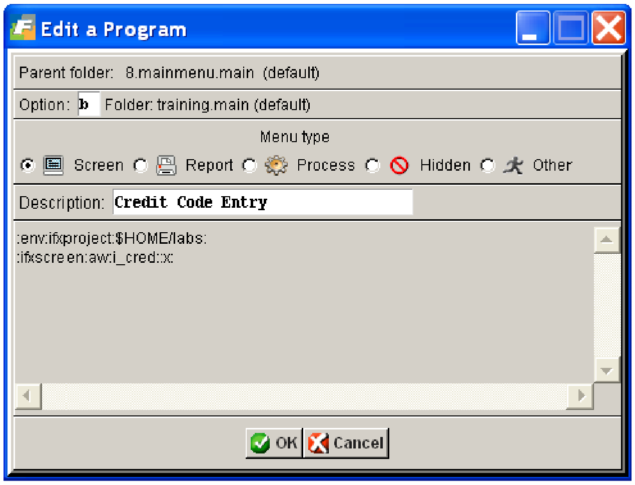
Menu type

● Screen ○ Report ○ Process ○ Hidden ○ Other

Description: Customer Entry

```
:env:ifxproject:$HOME/labs:
:ifxscreen:aw:i_cust:x:
```

6. Right click on the new Training Programs menu again and choose New Program. Fill out the Add a Program as displayed below. Click OK to save your work.



Parent folder: 8.mainmenu.main (default)

Option: b Folder: training.main (default)

Menu type

● Screen ○ Report ○ Process ○ Hidden ○ Other

Description: Credit Code Entry

```
:env:ifxproject:$HOME/labs:
:ifxscreen:aw:i_cred:x:
```

Run Your Menu Options

1. Switch back to classic menu mode by clicking the button.

2. Select Training Programs off of the main menu.

3. Run both your menu options to make sure they work correctly.

Save Your Custom Menu Options in $fg/data/sql.4gc

1. **Recall that you named the new folder `training.main` and the option was `8`. Use `dbaccess` to select see your new menu definition from the `cgsmnitm` and `cgsmncmd` tables using the `mname`, `opt` and `ckey` columns using the following sql.**

```
select * from cgsmnitm where
   mname = "mainmenu.main" and opt = "8" and ckey = " ";
select * from cgsmncmd where
   mname = "mainmenu.main" and opt = "8" and ckey = " ";
```

The result:

```
ckey
mname  mainmenu.main
opt    8
mtype  FL
txt    Training Programs

ckey
mname   mainmenu.main
opt     8
ilevel  1
cmd     :menu:training.main:
```

2. **Recall that *all* your new menu options have a mname = “training.main”. Use `dbaccess` to select see your new menu definition from the `cgsmnitm` and `cgsmncmd` tables using the `mname` and `ckey` columns using the following sql.**

```
select * from cgsmnitm where
   mname = "training.main"  and ckey = " ";
select * from cgsmncmd where
   mname = "training.main" and ckey = " ";
```

The result:

```
ckey
mname  training.main
opt    a
mtype  SC
txt    Customer Entry

ckey
mname  training.main
opt    b
mtype  SC
txt    Credit Code Entry
```

```
ckey
mname   training.main
opt     a
ilevel  1
cmd     :env:ifxproject:$HOME/labs:

ckey
mname   training.main
opt     a
ilevel  2
cmd     :ifxscreen:aw:i_cust::x:

ckey
mname   training.main
opt     b
ilevel  1
cmd     :env:ifxproject:$HOME/labs:

ckey
mname   training.main
opt     b
ilevel  2
cmd     :ifxscreen:aw:i_cred::x:
```

3. **Write sql to insert these rows into the database. Put them in a vm.sql file and save it in the $fg/data/sql.4gc directory. Below is the sql.**

```
insert into cgsmnitm (ckey,mname,opt,mtype,txt)
   values (" ","mainmenu.main","8","FL",
     "Training Programs");
insert into cgsmncmd (ckey,mname,opt,ilevel,cmd)
  values (" ","mainmenu.main","8","1",
       ":menu:training.main:");

insert into cgsmnitm (ckey,mname,opt,mtype,txt)
   values (" ","training.main","a","SC",
     "Customer Entry");
insert into cgsmncmd (ckey,mname,opt,ilevel,cmd)
  values (" ","training.main","a","1",
       ":env:ifxproject:$HOME/labs:");
insert into cgsmncmd (ckey,mname,opt,ilevel,cmd)
  values (" ","training.main","a","2",
       ":ifxscreen:aw:i_cust::x:");

insert into cgsmnitm (ckey,mname,opt,mtype,txt)
   values (" ","training.main","b","SC",
     "Credit Code Entry");
insert into cgsmncmd (ckey,mname,opt,ilevel,cmd)
  values (" ","training.main","b","1",
       ":env:ifxproject:$HOME/labs:");
insert into cgsmncmd (ckey,mname,opt,ilevel,cmd)
  values (" ","training.main","b","2",
       ":ifxscreen:aw:i_cred::x:");
```

Chapter 20

Security

- Security Overview
- The Security Programs
- Fitrix Visual Menus security

Security Overview

Security is based on a hierarchy. You design your security system around three levels of users. In addition, applications are divided into three levels. The key to setting up a quality security system depends on your understanding of these levels and how they relate to each other.

User Level	**Description**
Individual User	This level defines system users on a unique or individual basis. All system users, in other words anyone able to log in to the system, are considered individual users. You can grant individual users explicit allow or deny permission settings.
User Group	This level is made up of a subset of system users. You define and determine the types of groups and the members of each group on your system. When you set permissions for a group, all members of the group are given that permission.
Defaults	This level is made up of all system users. It uses defaults as a keyword that signifies a user group containing every individual user. When you set permissions for defaults, you are setting permissions for all users who do not receive more specific group or individual permissions.

Application Level	**Description**
Module	A collection of input and output programs that compose a product, such as General Ledger.
Program	A single program within a module. For instance, General Ledger Setup is an input program within the General Ledger module.
Event	An activity or command within a program. For example, many input programs let you Update current information. The Update command, then, is considered an event.

Security Programs

Security is a collection of programs that let you define security permissions for each level of user and application. Security consists of five input programs. These programs work interactively. In other words, information defined in one program is used to provide information for another program.

Program Name	**Description**
Module and Program Information	This program lists the modules and Information programs on your system. By default, this information comes preloaded in Security.
Security Events	This program lists the events used by the modules and programs on your system. Like modules and programs, event information is pre-loaded.
Security Groups	This program lets you define which individual users belong to which user group.
User & Group Permissions	This program provides a complete method for identifying users and groups on your system. In addition, it links information in the Module and Event programs with user and group definitions, and it allows you to set explicit user and group permissions. Most of the work you do with Security is done in this program.
Group Security Control	This program provides an easy-to-use interface for setting up group permissions on events. It does not contain all the features and flexibility of the User & Group Permissions program, but it is a simplistic alternative.

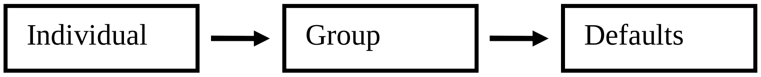
In later sections of this chapter, each program is described in more detail. This section concentrates on how Security takes and uses information supplied to the Security programs and which permission settings take precedence.

Determining Precedence

Security determines precedence in an inverted or "bottom up" manner. In other words, the most specific settings (the individual user settings and the event settings) take precedence over the more general settings.

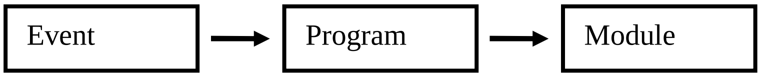
In terms of user levels, Security searches for an allow or deny permission first on the individual level, then on the group level, and finally on the global or defaults group level

User Level Search Order



In terms of application levels, Security looks first at the event level, then the program level, and finally the module level.

Application Level Search Order



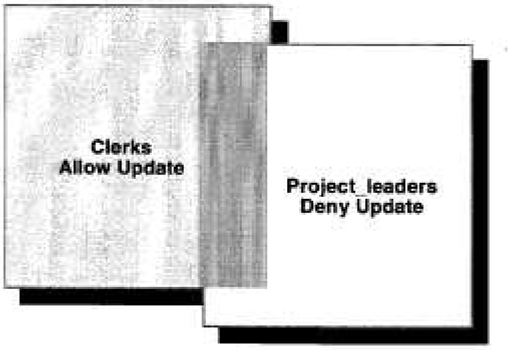
Overlapping Group Permissions

Security is designed to meet as many custom security setups as possible. For this reason, you can place individual users into more than one user group. Sometimes, however, users belong to groups that contain conflicting permission settings otherwise known as overlapping user groups. Users that belong to overlapping groups are given allow permission.

For instance a clerk might belong to a group called clerks and a group called `project_leaders`. At times, `clerks` and `project_leaders` might have conflicting permission settings. For instance, `clerks` might allow the Update event and `project_leaders` might deny it.

You can place users into more than one group.

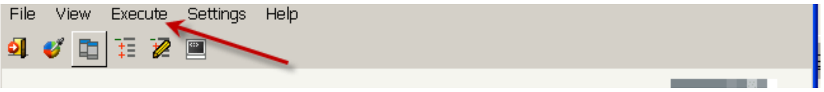
If a user is in two groups that have conflicting security permissions, allow permission is granted.



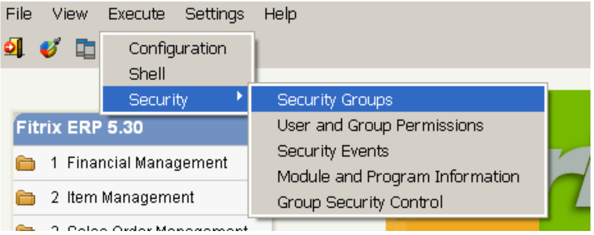
In this situation, the clerk who belongs to both groups is able to use the Update event.

Running the Security Programs

Security is a collection of five input programs. You use a11 of these programs to define Security on each level of user and application. You access the security programs from the Execute pull-down menu in Fitrix Visual Menus.



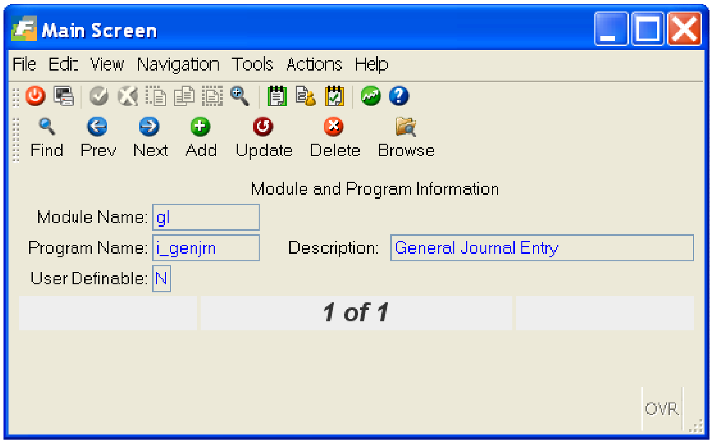
You choose the Security option to run the Security programs.



Module and Program Information

This input program lets you enter the modules and programs eligible to secure. All Fitrix modules and programs come pre-loaded. You only need to use Module and Program Information when you create custom programs or modules. The following figure shows the input screen for Module and Program Information:

Module and Program Information Program


Main Screen
File Edit View Navigation Tools Actions Help
Find Prev Next Add Update Delete Browse
OVR

Module and Program Information

Module Name: gl

Program Name: i_genjrn Description: General Journal Entry

User Definable: N

1 of 1

Adding Custom Programs to Module and Program Information

When you create a custom application, the Fitrix VDT Application Code Generator automatically builds logic that Security recognizes. But you must add the module and table information for that security logic to work. This module and program information is stored in the stxprogr table. The Module and program information program is a handy front end for the stxprogr table.

To add a custom report to Module and Program Information (stxprogr):

1. **Choose Add.**

2. **In the Module Name field, enter the module directory of the custom program.**

   For example, if your custom report is in **`sales.4gm`**, enter **`sales`** in the Module Name field.

3. **In the Program Name field, enter the program directory that contains your custom report.**

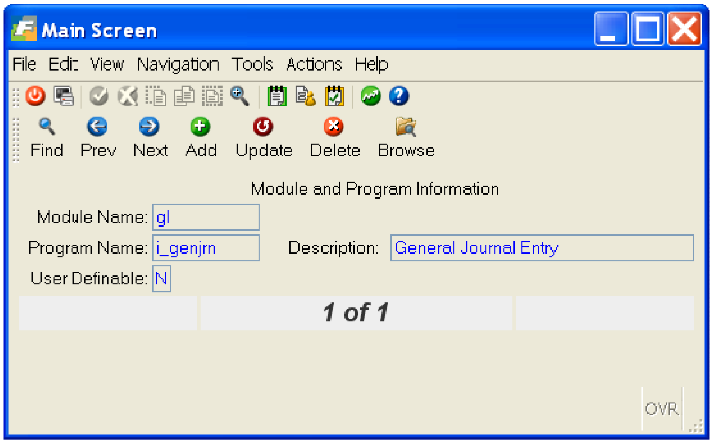
   For example, if your custom report is in **`o_sales.4gs`**, enter **`o_sales`** in the Program Name field.

4. **Describe your custom program in the Description field.**

   The User Definable field is a non-entry field.

5. **Press [ENTER] to store your entry.**

The Module and Program Information program lets you make custom programs “eligible” to secure.



Module and Program Information

Module Name: gl

Program Name: i_genjrn   Description: General Journal Entry

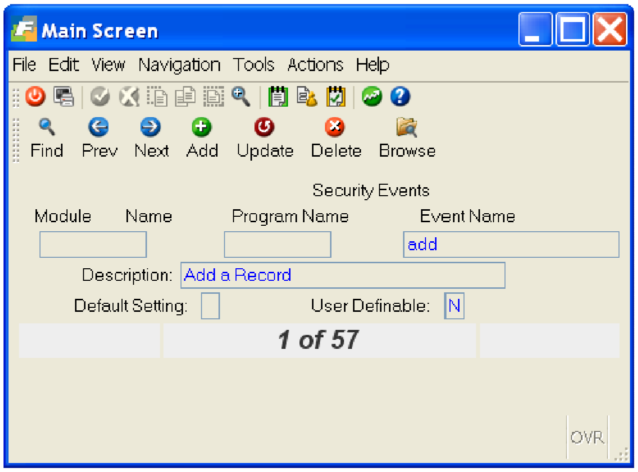
User Definable: N

1 of 1

Security Events

The Security Events program is similar to Module and Program Information. It too comes pre-loaded with events used in Fitrix programs, such as add, delete, and update. As well, Security Events lets you define custom events in custom programs. Similar to Module and Program Information, Security Events just lets you define events that are eligible to secure.

The Security Events program



Security Events

Module Name	Program Name	Event Name
		add

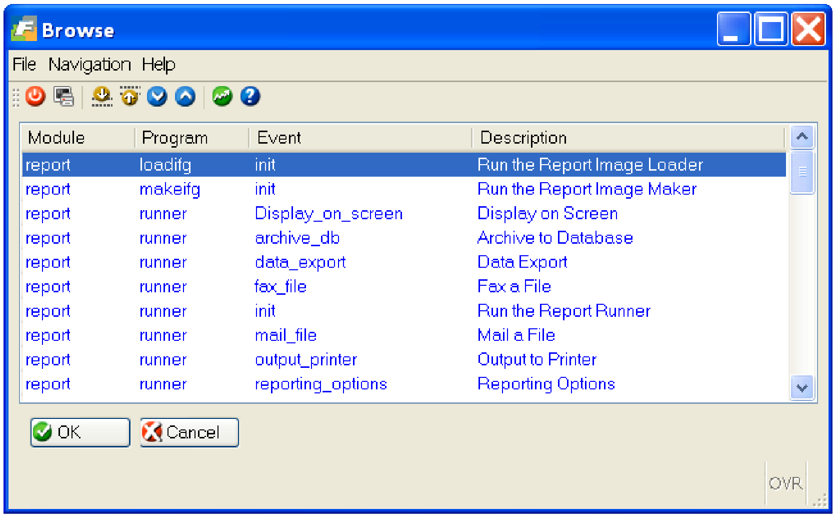
Description: Add a Record

Default Setting: □   User Definable: N

1 of 57

The following shows some of the 35 events associated with Report Writer.

There are 35 events associated with the *Report* Writer.



Module	Program	Event	Description
report	loadifg	init	Run the Report Image Loader
report	makeifg	init	Run the Report Image Maker
report	runner	Display_on_screen	Display on Screen
report	runner	archive_db	Archive to Database
report	runner	data_export	Data Export
report	runner	fax_file	Fax a File
report	runner	init	Run the Report Runner
report	runner	mail_file	Mail a File
report	runner	output_printer	Output to Printer
report	runner	reporting_options	Reporting Options

Adding Custom Events to Security Events

If your application contains custom events, you can add these events to the Security Events program. Once added, you can use the User and Group Permissions program to place individual and group permissions on your custom event.

Unlike custom programs, where Security logic gets generated automatically, you must add a few lines of code at the start of your custom events for Security to be able to recognize it.

For example, suppose you create a `o_sales` program. In `o_sales`, you create a custom event that allows users to fax report output to company headquarters. At the start of your custom fax event, add the following lines of code:

```
# Inserted for program level security.
# Check for permission
if not security_chk ("fax")
then
    call security_msg("fax")
else
    call fax(p_stomer.phone)
end if
```

After you add this code to your custom event, making that event eligible to secure requires the following steps:

1. **Select Add in the Security Events program to add your custom event.**

2. **In the Module Name field, enter the module directory of your custom program.**

   For example, if the module directory is **`sales.4gm`**, enter **`sales`**.

3. **In the Program Name field, enter the program directory of your custom program.**

   For example, if the program directory is **`o_sales.4gs`** enter **`o_sales`**.

4. **In the Event Name field, enter the name of your custom event.**

   For example, if the event name is **`fax`** enter **`fax`**.

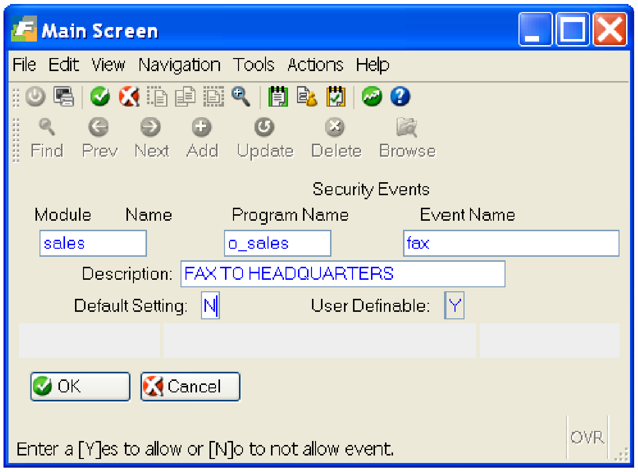
5. **In the Description field, enter a description of your event.**

6. **In the Default Setting field, enter the default permission for the event.**

   The User Definable field is a non-entry field.

7. **Press [ENTER] to store your entry.**

Use the Security Events program to make custom events securable.



Security Events

Module Name: sales

Program Name: o_sales

Event Name: fax

Description: FAX TO HEADQUARTERS

Default Setting: N

User Definable: Y

Enter a [Y]es to allow or [N]o to not allow event.

Note

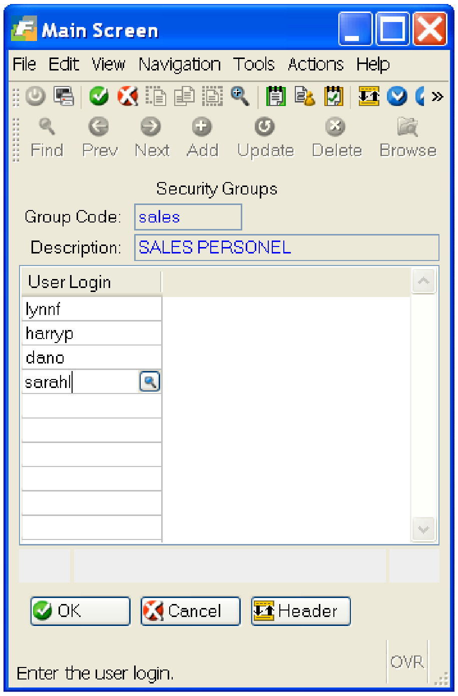
If you want to set permissions for your event in all the programs in a module, leave the Program Name field blank.

Security Groups

This program lets you assign individual users to groups. By creating groups of users, from individual users who require similar system access, you can simplify your security configuration.

For example, you might want to assign your entire sales force to a group called sales. Your definition of the sales group might look as follows:

The Security Groups program lets you define groups of users who share the same permission settings.



Main Screen
File Edit View Navigation Tools Actions Help
Find Prev Next Add Update Delete Browse

Security Groups

Group Code: sales

Description: SALES PERSONEL

User Login
lynnf
harryp
dano
sarahl

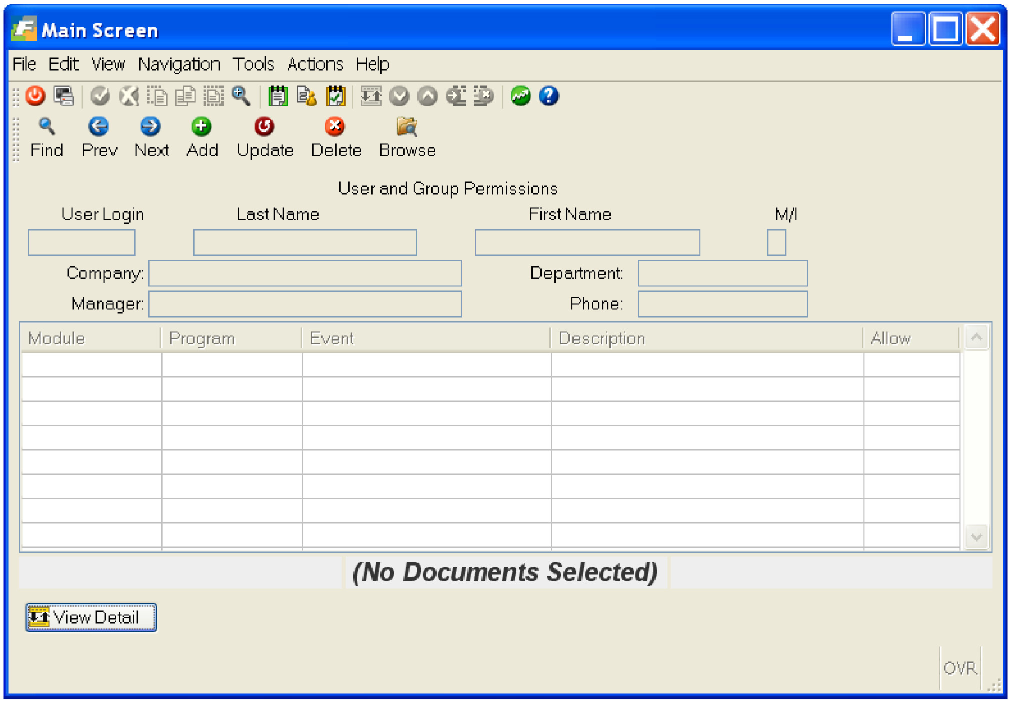
Enter the user login.

Once you define a security group, you can set permissions for that group in the User and Group Permissions program or in Group Security Control.

User and Group Permissions

This input program is where most of your security work gets done. It is this program that relates the information set in Module and Program Information, Security Events, and Security Groups with actual permission settings.

The User and Group Permissions program.



Setting Individual User Permissions

The most basic task of the User and Group Permissions program is setting permissions for an individual user.

To set permission for an individual user:

1. **Select Add.**

2. **Enter values for the User Login and Last Name fields.**

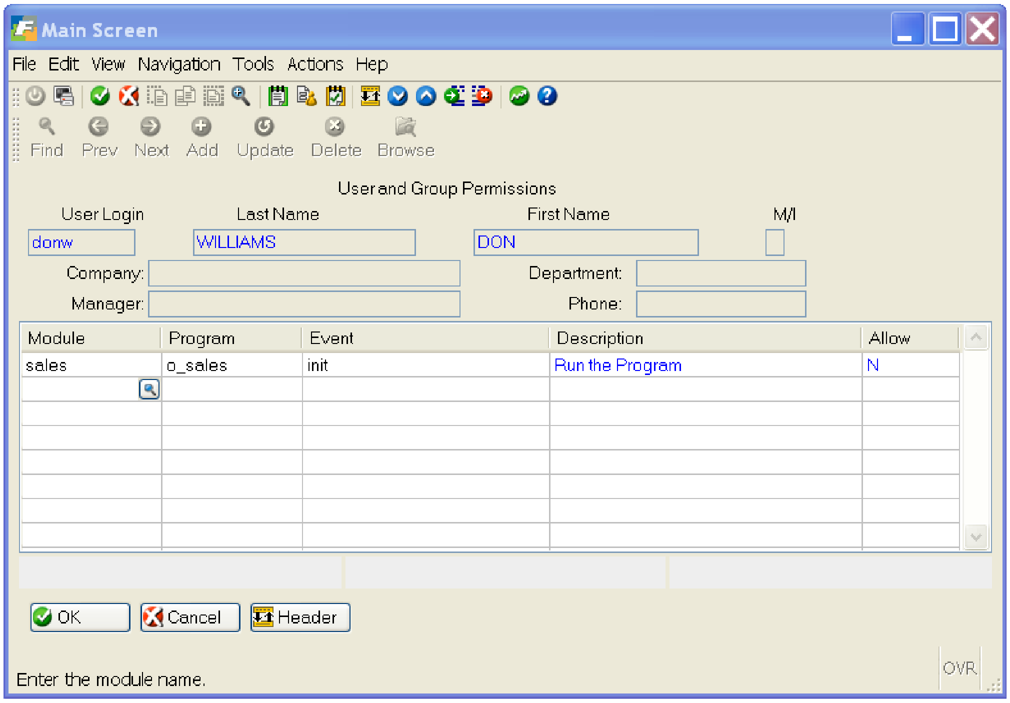
   For example, if you are setting permissions for **donw**, enter **donw** in the User Login field and donw's last name (for instance Williams) in the Last Name field.

   The User Login and Last Name fields are the only required fields. The other fields in the header section are optional, such as the Department and Phone fields.

3. **Press Detail to move to the detail section of the program.**

   In the detail section you can enter the module, program, and event you want to set permissions on. You can also press [CTRL][z] to pick from a list of defined modules, programs, and events.

   For example, suppose you want to deny **donw** the ability to run the sales report.


Main Screen
File Edit View Navigation Tools Actions Hep
Find Prev Next Add Update Delete Browse
OK
Cancel
Header
OVR

User and Group Permissions

User Login	Last Name	First Name	M/I
donw	WILLIAMS	DON	

Company:
Department:
Manager:
Phone:

Module	Program	Event	Description	Allow
sales	o_sales	init	Run the Program	N

Enter the module name.

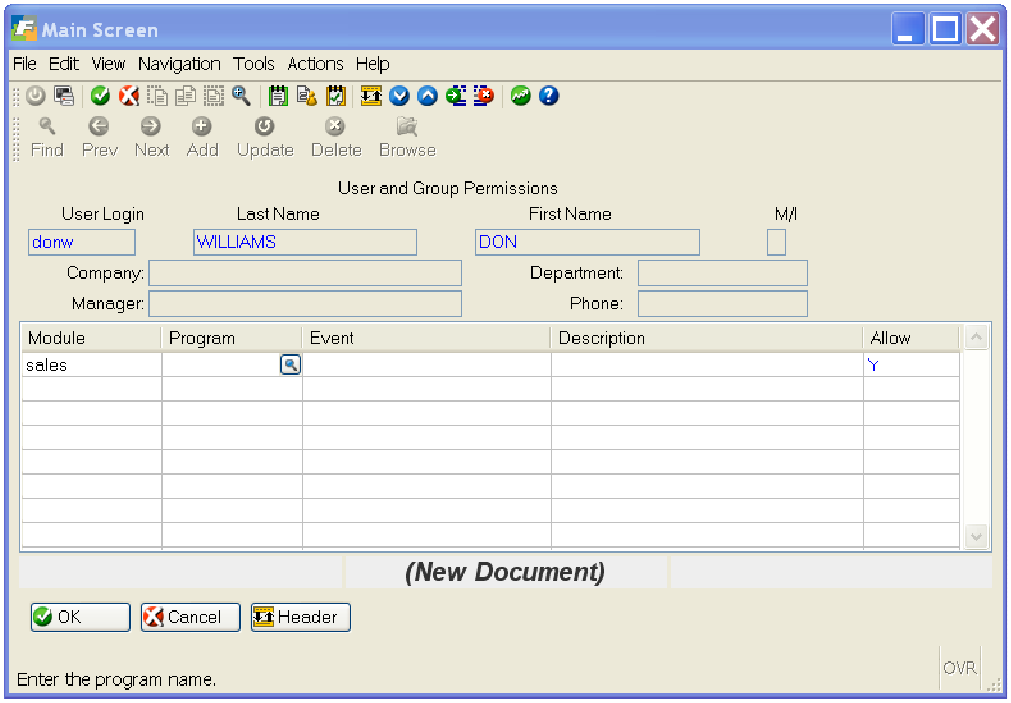
4. **Once you finish entering permission data, press [ENTER] to store your entry.**

Setting Permission for an Entire Module

To set permissions for an entire module, only specify the module name in the detail portion of User and' Group Permissions.

For example, to deny `donw` access to all programs in the sales module, make the following entry:

This entry denies donw access to all the programs in the report module.


Main Screen
File Edit View Navigation Tools Actions Help
Find Prev Next Add Update Delete Browse
OK
Cancel
Header
OVR

User and Group Permissions

User Login	Last Name	First Name	M/I
donw	WILLIAMS	DON	

Company:
Department:
Manager:
Phone:

Module	Program	Event	Description	Allow
sales				Y

(New Document)

Enter the program name.

In a similar sense, you can set permissions for all events in a program: specify both the module and program and leave the Event field blank.

Setting Group Permissions

You can also set permissions for groups that you have defined in the Security Group program (see "Security Groups" on page 307). In the same way you set permissions for individual users, you also set permissions for groups.

To set permissions for a group:

1. **From the User and Group Permssions program, choose Add.**

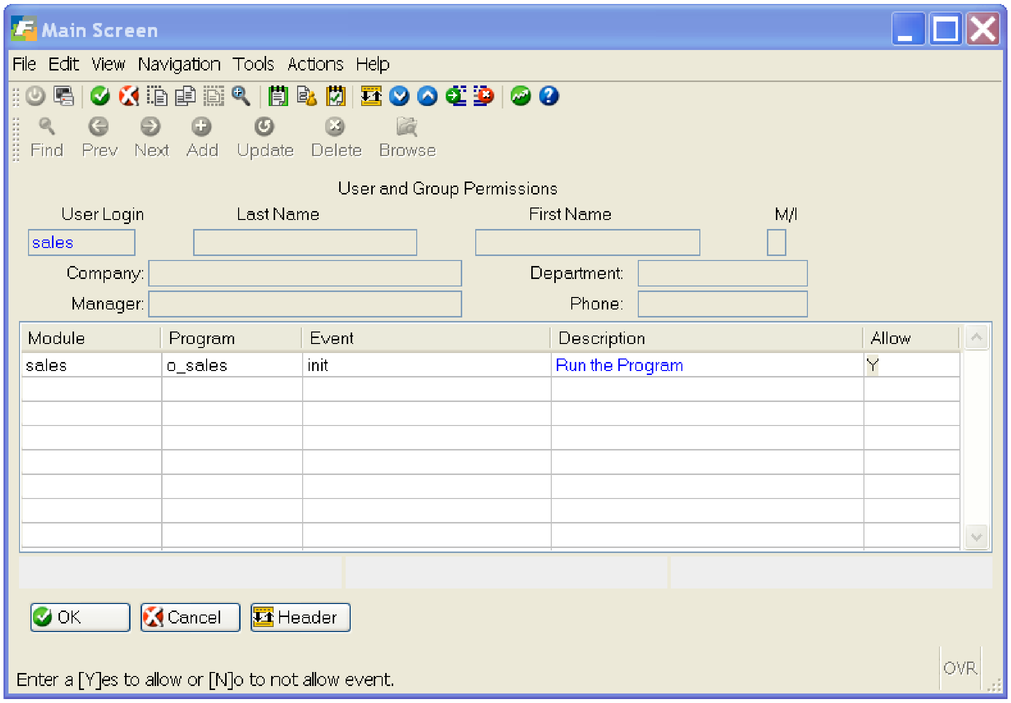
2. **Enter the group code (i.e., group name) in the User Login field and enter a description of the group in the Last Name field.**

3. **Press** Detail **to move to the detail portion of the program.**

   In the detail section you can enter the module, program, and event you want to set permissions on. You can also press [CTRL][z] to pick from a list of defined modules, programs, and events.

   For example, to set permissions of the sa1es group for init (running the program) sales.4gm/o_sales program event:

This entry sets permissions for the sales group.


Main Screen
File Edit View Navigation Tools Actions Help
Find Prev Next Add Update Delete Browse
OK
Cancel
Header
OVR

User and Group Permissions

User Login	Last Name	First Name	M/I
sales			

Company:
Manager:
Department:
Phone:

Module	Program	Event	Description	Allow
sales	o_sales	init	Run the Program	Y

Enter a [Y]es to allow or [N]o to not allow event.

4. **Once you finish entering permission data, press [ENTER] to store your entry.**

Setting Defaults Permission

The Defaults permission is a reserved permission setting. The values set for Defaults are passed to all users and groups not otherwise defined. For instance, if the user `robertc` does not belong to any groups and does not have an individual user entry, he receives the permissions set in defaults.

To set Defaults permission:

1. **Select Add in the User and Groups Permissions program.**

2. **Enter `defaults` in the User Login field and `DEFAULTS` in the Last Name field.**

3. **Press Detail to move to the detail section of the screen.**

   In the detail section, enter the module, program, and event you want to set permissions on. You can also press [CTRL]-[z] to pick from a list of defined modules, programs, and events.

4. **Press [ENTER] to store your settings.**

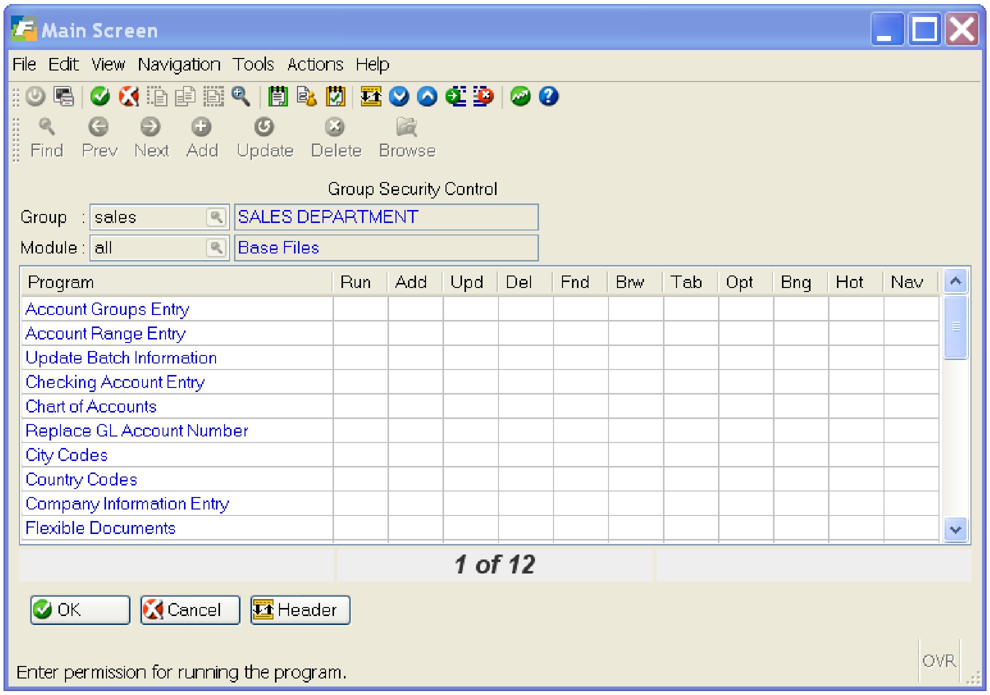
Caution

The Defaults permission affects all users on the system.

Group Security Control

Group Security Control is a simplified version of the User and Group Permissions program. With Group Security Control, common program events are already listed. Group Security Control has a matrix type interface, which helps you assign permission settings.

This entry sets permissions for the account group on the report programs.


Main Screen
File Edit View Navigation Tools Actions Help
Find Prev Next Add Update Delete Browse
OK
Cancel
Header
OVR

Group Security Control

Group : sales SALES DEPARTMENT

Module : all Base Files

Program	Run	Add	Upd	Del	Fnd	Brw	Tab	Opt	Bng	Hot	Nav
Account Groups Entry											
Account Range Entry											
Update Batch Information											
Checking Account Entry											
Chart of Accounts											
Replace GL Account Number											
City Codes											
Country Codes											
Company Information Entry											
Flexible Documents											

1 of 12

Enter permission for running the program.

The following describes the events available in Security Control.

Event	**Description**
Run	The Run event controls the use of the listed program. When the Run permission field is set to Y, members of the group can start the listed program. When set to N, the group cannot start the listed program.
Add	The Add event controls the ability to add or create new program documents. When Add is set to Y, documents can be added. When set to N, the group cannot add a document.
Upt	The Upt event specifies a group's ability to update a document. A Y in this field lets group members update a document, an N denies update permission.
Del	The Del event controls document deletion. Many times only specific users are allowed delete permission. When you set the Del event to Y, the group can delete documents. When set to N, documents cannot be deleted.
Fnd	The Fnd event controls a program's Find capabilities. When you set the Fnd event to Y, group members can conduct Query-ByExample searches for specific documents. When set to N, users cannot use the Find feature.
Brw	The Brw event controls the Browse capabilities. When you set Brw to Y, the group can use the Browse command. When set to N, browse privileges are denied.
Tab	The Tab event coincides with the Tab command. When you set the Tab field to Y, the group can use the Tab command. When set to N, group members cannot use the Tab command.
Opt	The Opt event controls access to the Options command. A Y in the Opt field grants access to the Options command, an N denies access.
Bng	The Bng event controls access to the operating system. In most cases, users are able to bang out (also called shell out or escape) to the operating system. When the Bng event is set to Y, the group can bang out of the program. When set to N, the group cannot escape to the operating system.
Hot	The Hot event corresponds to a program's HotKeys. Inmany programs, users can define Hot Keys that serve as keyboard shortcuts to common program commands. When you set the Hot event to Y, users can alter the default Hot Key definitions. When set to N, users

cannot edit the default Hot Key definitions.

Nav

The Nav event relates to a program's Navigate feature. In many programs, users can press [CTRL]-[g] to view the Navigate pop-up menu. When you set the Nav event to Y, users gain the ability to use this menu. When set to N, users cannot use the Navigate menu.

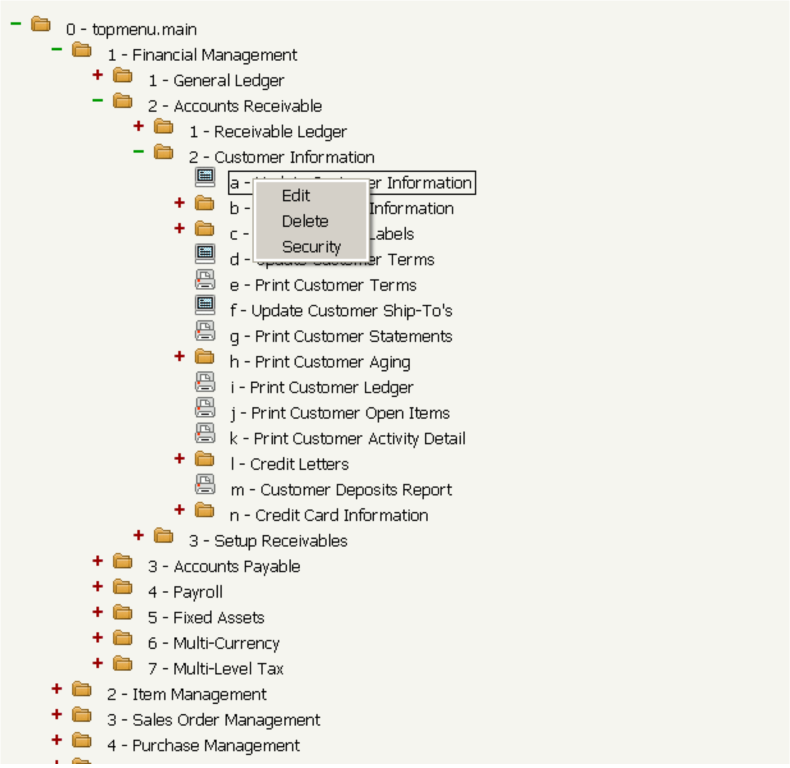
Fitrix Visual Menus Security

When you add users and groups with the User and Groups Permissions program, you are making those users and groups available to Fitrix Visual Menus Security. With Fitrix Visual Menus Security you can disallow execution of menu options and grey out items in Fitrix Visual Menus.

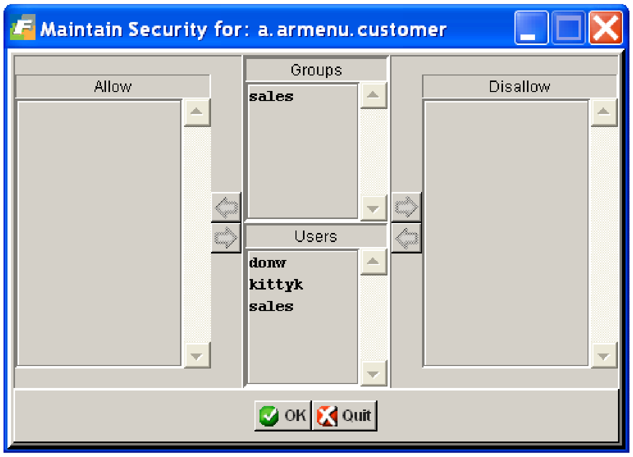
You need not set security at the Visual Menus level but it may be more convenient if your requirements allow you to set access at the Folder/Menu level. For example, if you need to allow only a small group of people into the options on the Financial Management menu, it would save time and perhaps some confusion to set permissions using the Fitrix Visual Menus level. Setting permissions using Fitrix Visual Menus on the Financial Management menu option would grey out the Financial Management options for those who are not allowed and they would not be able to see the individual menu options.

Please note that if you setup new users and groups, you must exit Fitrix Visual Menus and reenter so that the menu tables are reloaded from the database. If you do not reload the menu data, you do not see the new users and groups in Fitrix Visual Menus.

You setup Fitrix Visual Menus security by switching to Edit mode. You can switch to edit mode by clicking the button. Then you right click on the menu option and choose Security. Example:



The Fitrix Visual Menus Security Window appears.



You add groups and users to the Allow and Disallow lists by clicking the arrow keys. It is logical to use one list only. For example, you use the Allow list if there are only a few groups or users that are allowed to use a menu option. You should use the Disallow option if there are only a few groups and users you disallow to use a menu option. In addition, if your security needs allow you to set permissions at the Folder/Menu level, that is the place to do it. Setting security at the folder/menu level can cut down on maintenance.

When you add information to Fitrix Visual Menu security, that data is stored in the `cgsmnsec` table. The columns of `cgsmnsec` ar below.

ckey

Custom Key. Blank (“ ”) if not custom. Only populate ckey if you wish to have an entirely custom menu.

mname

Contains the menu name. The menu name contains hierarchy information for the menu item. Convention is parentmenu.menu. This is a join column to cgsmnitm

opt

Contains the accelerator key/keyoard key for the menu option. For example, the accelerator key for the menu option ‘a Update General Journal’ is ‘a’. The opt column must be unique within a menu. This is a join column to cgsmnitm.

either_id

Either a group id or a user id.

allow_flag

A = Allow execution

D = deny execution.

Section Summary

- Security is based on a hierarchy. You design your security system around three levels of users. In addition, applications are divided into three levels. The key to setting up a quality security system depends on your understanding of these levels and how they relate to each other.

- Security is a collection of five input programs. You use all of these programs to define Security on each level of user and application.

- The Module and Program Information program lets you enter the modules and programs eligible to secure. All modules and programs come pre-loaded. You only need to use Module and Program Information when you create custom programs or modules.

- Security Events is similar to Module and Program Information. It too comes pre-loaded with events used in programs, such as add, delete, and update. As well, Security Events lets you define custom events in custom programs. Similar to Module and Program Information, Security Events just lets you define events that are eligible to secure.

- Security Groups lets you assign individual users to groups. By creating groups of users, from individual users who require similar system access, you can simplify your security configuration.

- User and Group Permissions is where most of your security work gets done. It is the program that relates the information set in Module and Program Information, Security Events, and Security Groups with actual permission settings.

- Group Security Control is a simplified version of the User and Group Permissions program. With Group Security Control, common program events are already listed. Group Security Control has a matrix type interface, which helps you assign permission settings.

- Fitrix Visual Menus security can disallow or allow menu and program execution as well. Fitrix Visual Menus security is most useful if you need to control security at the Folder/Menu level.

Exercise 20A

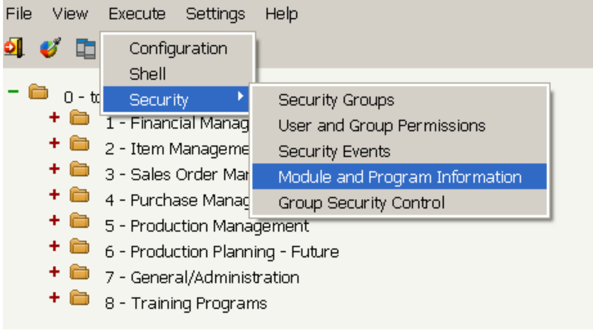
Objective: To use: Security to deny yourself the ability to update records in your Customer Entry program.

Security lets you control how a program is used and by whom. In this exercise, you will set a security restriction on yourself. You will deny yourself access to the Update command in your Customer Entry program.

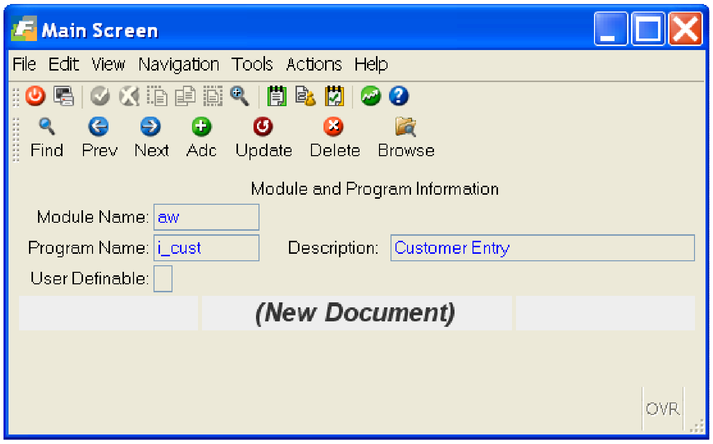
Start the Module Information Program

This program adds your Customer Entry program to a "roster" in the database. The roster is simply a listing of all the modules and programs that are "securable" or eligible to secure. The table that stores this “roster” is `stxprogr`.

1. **From the Execute pull-down menu choose Security then Module and Program Information.**



2. **Select Add and enter the following information and save it.**


Main Screen
File Edit View Navigation Tools Actions Help
Find Prev Next Adc Update Delete Browse
OVR

Module and Program Information

Module Name: aw

Program Name: i_cust Description: Customer Entry

User Definable: □

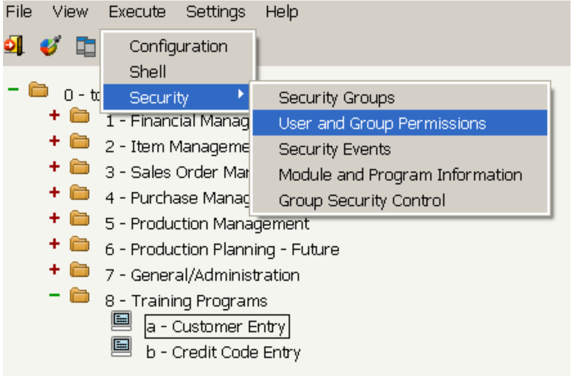
(New Document)

3. **Quit out of the program.**

Start the User Permissions Program

User Permissions assigns different security permission values to individual users or groups of users. You will use this program on yourself.

1. **From the Execute pull-down menu choose Security then Module and User and Group Permissions:**



The User Information program appears. This program contains both a header and a detail section. The header section contains information about the user, which in this case will be you. The detail section contains information about the module, program, event, and permission setting.

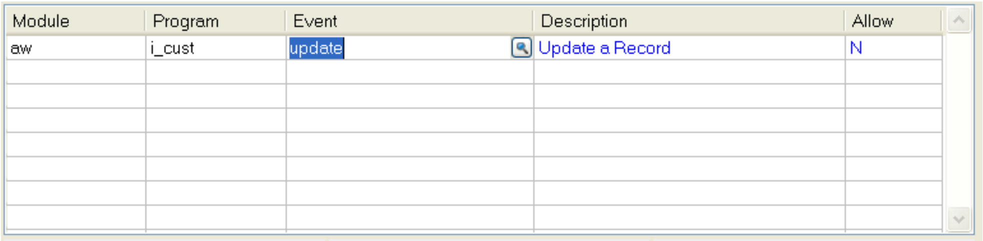
2. **Select Add to create a new user record.**

3. **Place your user login in the User Login field.**

4. **Enter your first and last name in the Name fields.**

5. **Click** Detail **to move to the detail section:**

6. **Fill in the detail fields as follows and press [ENTER] to save this record:**



Module	Program	Event	Description	Allow
aw	i_cust	update	Update a Record	N

7. **Quit from User and Group Permissions.**

Start Your Customer Entry Program

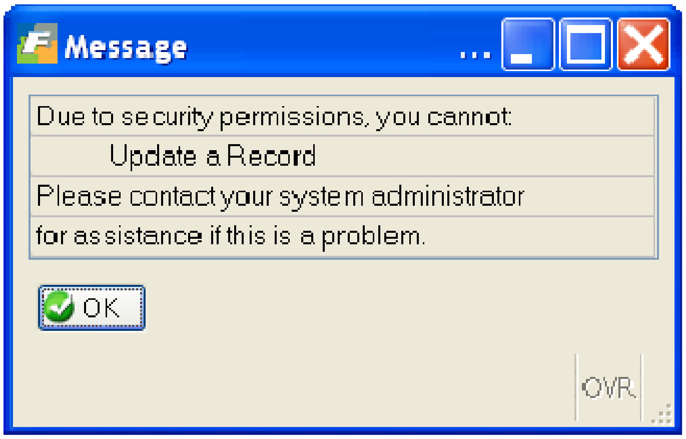
1. **Use cd to move to your i_cust 4gs directory and start your Customer Entry program.**

   If you would rather, you can also start it from the *Menus* program you created in Exercise 19.

2. **Use Find to select a record or group of records.**

3. **Press Update to alter the record.**

   A message appears denying you access to update:



Due to security permissions, you cannot:
Update a Record
Please contact your system administrator
for assistance if this is a problem.

4. **Press OK.**

   Notice that you can still use the other commands, you only restricted access to the Update command.

5. **Quit from Customer Entry.**

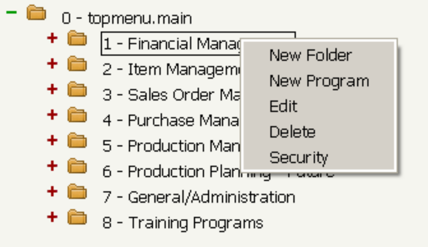
Exercise 20B

Objective: To use Fitirx Visual Menus Security to deny you the access to all the menu options on the Financial Management menu.

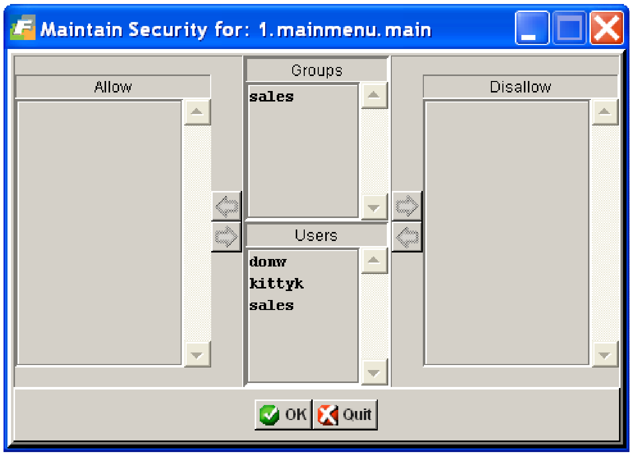
With Fitrix Visual Menus you can deny access to programs at the Folder/Menu level. This security feature denies users access to entire menus. Specifically users cannot even see the items on these menus.

Enter Edit Mode

1. **Click the on the toolbar**.
2. Right click on the 1- Financial Management menu. Choose Security.



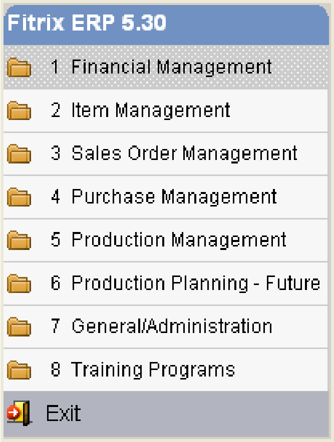
3. A window similar to following window appears. Groups and Users may be different.



4. **Choose your user id from the Users list by clicking on it. Click the right arrow to add it to the Disallow list. Click OK to save your changes.**

5. **Exit out of Fitrix Visual Menus. You must reload the menus to update security on your workstation. Start Fitrix Visual Menus again. If you switch to classic mode you see that**

the Financial Management menu is grayed out   Try to enter the Financial Management menu.


Fitrix ERP 5.30
1 Financial Management
2 Item Management
3 Sales Order Management
4 Purchase Management
5 Production Management
6 Production Planning - Future
7 General/Administration
8 Training Programs
Exit