CREDIT CARD PURCHASING SYSTEM
SHORE INTERMEDIATE MAINTENANCE ACTIVITY (SIMA)
UNITED STATES NAVY

GRADUATE PROJECT

BY
RUSSELL LOWENTHAL
April 26, 1999

COMMITTEE MEMBERS

Dr. Mario Andre Mayerhofer Guimaraes, Chairman

Dr. Robert Stephen Dannelly, Member

Dr. Chandrika Rao, Member
Abstract

This project is the development of an interactive, Oracle-based, credit-card purchasing system for the Shore Intermediate Maintenance Activity (SIMA) at Naval Station Ingleside, Texas. The credit-card purchasing system is now being used to accept all requests for purchases less than or equal to $2,500.00 scheduled to be made by government credit card from the existing Oracle-based enterprise management system. Requests are being assigned to a buyer for action and tracked throughout the life of the purchase from the original order through pickup, receipt, and delivery. The system provides an easily understood, user-friendly interface with the database. All necessary reports are available to the user both on-line and via printed copy.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>iii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>iv</td>
</tr>
<tr>
<td>List of Code Listings</td>
<td>v</td>
</tr>
<tr>
<td>Background and Rationale</td>
<td>1</td>
</tr>
<tr>
<td>Narrative</td>
<td>5</td>
</tr>
<tr>
<td>Environment of Project</td>
<td>19</td>
</tr>
<tr>
<td>Table Descriptions</td>
<td>21</td>
</tr>
<tr>
<td>Entity Relationship Diagram</td>
<td>23</td>
</tr>
<tr>
<td>User Screens - Database Matrix</td>
<td>24</td>
</tr>
<tr>
<td>Procedure</td>
<td>25</td>
</tr>
<tr>
<td>Client Installation</td>
<td>25</td>
</tr>
<tr>
<td>CompassCONTRACT Menu Integration</td>
<td>27</td>
</tr>
<tr>
<td>Initializing the Program</td>
<td>32</td>
</tr>
<tr>
<td>Preparing to Update the Purchase Request</td>
<td>45</td>
</tr>
<tr>
<td>Selecting the Credit Card Buyer</td>
<td>66</td>
</tr>
<tr>
<td>Credit Card Buyer Query and Update</td>
<td>69</td>
</tr>
<tr>
<td>Procedure Summary</td>
<td>70</td>
</tr>
<tr>
<td>Detailed Table Descriptions</td>
<td>71</td>
</tr>
<tr>
<td>Results</td>
<td>95</td>
</tr>
<tr>
<td>References</td>
<td>96</td>
</tr>
</tbody>
</table>
**List of Figures**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>External Interfaces</td>
<td>6</td>
</tr>
<tr>
<td>2.</td>
<td>Graduate Project in relation to overall system</td>
<td>6</td>
</tr>
<tr>
<td>3.</td>
<td>Data Flow</td>
<td>7</td>
</tr>
<tr>
<td>4.</td>
<td>Program Selection Screen</td>
<td>8</td>
</tr>
<tr>
<td>5.</td>
<td>Login Screen</td>
<td>8</td>
</tr>
<tr>
<td>6.</td>
<td>Original Purchase Approval Screen</td>
<td>9</td>
</tr>
<tr>
<td>7.</td>
<td>Purchase Requisition Approval Queue</td>
<td>10</td>
</tr>
<tr>
<td>8.</td>
<td>Purchase Request Approval Screen</td>
<td>11</td>
</tr>
<tr>
<td>9.</td>
<td>Requisition Type Selection Screen</td>
<td>13</td>
</tr>
<tr>
<td>10.</td>
<td>Assign a fund code</td>
<td>13</td>
</tr>
<tr>
<td>11.</td>
<td>Credit Card Buyer’s Screening Queue</td>
<td>16</td>
</tr>
<tr>
<td>12.</td>
<td>Credit Card Buyer’s Detail Screen</td>
<td>17</td>
</tr>
<tr>
<td>13.</td>
<td>Purchase Order Receiving Screen</td>
<td>18</td>
</tr>
<tr>
<td>14.</td>
<td>Granting users access to the new screens</td>
<td>25</td>
</tr>
<tr>
<td>15.</td>
<td>Purchase Approval Queue Canvas 2</td>
<td>27</td>
</tr>
<tr>
<td>16.</td>
<td>Purchase Approval Queue Canvas 3</td>
<td>35</td>
</tr>
<tr>
<td>17.</td>
<td>Buyer Selection Screen</td>
<td>65</td>
</tr>
</tbody>
</table>
CODE LISTINGS

1: PRAV WHEN_NEW_FORM_INSTANCE trigger
   24
2: PRAV Query button WHEN BUTTON_PRESSED trigger
   26
3: PRAV PRD_BLOCK Post Query Trigger
   28
4: PRAV PRD Block When_New_Record trigger
   35
5: PRAV Req_Type Drop Down Box trigger
   35
6: PRAV Continuing Services WHEN_VALIDATE_ITEM trigger
   37
7: PRAV Fund Code list-of-values selection statement
   38
8: PRAV Disapprove Button WHEN BUTTON_PRESSED trigger
   40
9: PRAV Close Window WHEN BUTTON_PRESSED trigger
   41
10: PRAV Administrative Cancel When_Button_Pressed trigger
    42
11: PRAV PRD_Table Pre-Update Trigger
    46
12: Database PRD_Table Pre-Update Trigger
    48
13: Database Create Milstrip Stored Procedure
    52
14: Database Calculate PO serial number stored procedure
    54
15: Database Create_PO_POD Stored Procedure
    61
16: PRAV PRD_Table On-Update trigger
    61
17: Database PRD_Table Pre-Insert Trigger
    62
18: PRAV PRD_table Post-Update trigger
    62
19: Select Buyer When_Button_Pressed trigger
    64
20: Credit Card Buyer When New Form Instance trigger
    64
21: Credit Card POD summary WHEN_MOUSE_CLICKED trigger
    65
22: Credit Card POD_BLOCK ON_POPULATEDETAILS trigger 66
23: Database POD Notes table creation statement 66
24: Credit Card Return to Queue When Button Pressed trigger 67
Background and Rationale

The United States Navy was one of the first government activities to dedicate computer resources to the task of managing and tracking logistics actions. Starting with the late 1960’s, the Navy developed standardized eighty-column punch card format for use in requesting material, reporting the status of material requests, shipping dates, cancellations, and hundreds of other logistics actions. As computing hardware and software advanced over the years, a vast array of government-owned computer systems evolved around this standardized format (known as the MILSTRIP format). Even after mass data storage replaced the punch card, the Navy (as well as the Department of Defense as a whole) was forced to remain committed to the MILSTRIP format for communicating supply requests, because that was the one standard that all of the various systems recognized and could communicate with (NAVSUP 1997).

Continuing development, modernization, and maintenance of these various government-owned software and hardware systems is extremely costly (on the order of several billion dollars per year). In an effort to conserve dwindling defense dollars, as well as in response to the rapidly approaching turn-of-the-century and its attendant Y2K problems, the Department of Defense initiated several projects aimed at using commercially available hardware and software to replace the existing systems. In January of 1997, SIMA Ingleside was selected as the Navy’s pilot site for a Commercially-developed, Off-The-Shelf (COTS) enterprise management system called Compass CONTRACT. Simultaneous projects are being conducted by the U.S. Air Force at the Aviation Maintenance and Refit Center (AMARC) in Tucson, Arizona, by the U.S.
Marine Corps at Marine Corps Logistics Center in Albany, Georgia, and by the United States Army at Tobyhanna Army Depot, Tucson Arizona

After approximately eight months of test and development, SIMA Ingleside began using the CompassCONTRACT program (February 1998) to manage the repair and upkeep of the 29-vessel minesweeping fleet. During that eight-month development period, interfaces with the existing Defense Supply System were developed and put into place (following the standard MILSTRIP format discussed earlier).

Concurrent with the effort to change the computer system at SIMA, the Navy also expanded an existing initiative for the supply system to streamline the process of purchasing parts used in the repair and maintenance of government systems.

Under the original purchasing system, parts were purchased for the entire Department of Defense by several Inventory Control Points (ICP) that were used to manage the lifecycle of parts assigned to them. When a spare part was required, the organization that needed the spare part would generate a MILSTRIP document and send it (via several intermediate levels of bureaucracy) to the ICP (NAVSUP 1996). The ICP personnel would either:

a. ship the part -or-

b. order the part and send another MILSTRIP document indicating when the requester might expect delivery.

As you might imagine, this was a slow, cumbersome process that was quite costly to maintain. By the time each level of bureaucracy had finished adding “value” to the purchase request, it often took as much as a month for a routine request to even reach the ICP. In addition to the time added by all of this bureaucracy, each person and
organization that handled the MILSTRIP also cost the government money. For that reason, a simple claw-hammer (available from the local hardware store for under $10.00) might cost the requesting activity several hundred dollars!

Recognizing that the existing system was inefficient and costly, the Navy began issuing VISA credit cards to each organization. An arbitrary cut-off limit of $2,500 was established and organizations were given permission to “open-purchase” items that were available commercially. Now that $10.00 hammer only cost the requesting organization $10.00.

Unfortunately, up until this project there was no formal method developed to track these credit-card purchases. In the past, CompassCONTRACT generated purchase requests and the resulting MILSTRIP documents were simply deleted from the outgoing files sent to the ICP. During a typical month, several hundred credit-card requests are made and filled. Under the old system, there was no way for credit-card requests to be electronically assigned to a buyer (they are using paper requests and a manual tickler system). There was also no way for the requester to find out the status of the purchase request short of making a phone call or visiting the supply department to ask about the status. There was also no way for CompassCONTRACT to determine that parts had been received so that work that had been delayed awaiting parts could be released for completion.

Development of a solution to this problem not only presented an excellent opportunity to improve efficiency at SIMA (thus saving tax-dollars), but it also offered an excellent candidate for a graduate project.
The problem was a good candidate for a graduate project because it involved application of principles and techniques learned during graduate studies in a real-world environment. Because there was no existing system, user requirements had to be identified and documented using the structured methodology learned in Systems Analysis and Design (COSC 5338). The solution is heavily database dependent (CompassCONTRACT is built around an ORACLE database), thus reinforcing material learned in Introduction to DBMS (COSC 3436), Data Structures (COSC 5321), and Database Management Systems (COSC 5335). Developing the user interface and linking the new system to the existing system relied on programming techniques developed during Object Oriented Programming (COSC 4424) and Foundations in Programming and Problem Solving (COSC 5311) classes. Finally, this project related directly to my area of specialization, Government/Defense Systems.
This program adds increased functionality to an existing enterprise resource planning program, CompassCONTRACT. CompassCONTRACT is a product of the Los Angeles based Western Data Systems (WDS) company. Western Data Systems provides enterprise resource planning software to a worldwide customer base consisting primarily of aerospace and defense related industries.

CompassCONTRACT was installed at the Shore Intermediate Maintenance Activity, Ingleside, Texas as part of a pilot program to replace government developed software with commercial off-the-shelf (COTS) software. Because the system must work with existing external applications to receive requests for maintenance, order supplies and materials, and report financial transactions, several interfaces to those external applications had to be developed.
The project adds another layer between Compass CONTRACT and two of the external applications as seen below:

Overall, data flow in the portion of the system affected by this project is show below:
This system is being used by two functional groups of people. One group is the 
*comptroller*, who screens and approves/disapproves requests for purchases to verify that 
purchases are for a legitimate purpose and that the prices are reasonable. The other group 
is the *buyer*, who takes the approved purchase request and actually procures the material 
being ordered. There are multiple users within each of the two groups.

The comptroller’s function is to:

- review the purchase request for technical accuracy
  - price
  - urgency of need
  - unit of issue
- assign the purchase request to a buying agency
- assign the purchase request to a fund code
- approve or disapprove the purchase request

**Review the purchase request.** The comptroller is using the new Purchase 
Approval Screen. In order to get to this screen, the comptroller enters the parent program 
and selects program group “CA” option “21” from the menu (see figure 1). The 
comptroller enters the database number (01 for production, 10 for training) and the 
appropriate keyword and then presses the “Launch” button. This brings up a standard 
login screen (see figure 2) where the comptroller enters the appropriate login name and 
password. The Server and Port field are automatically filled in from the user’s preset 
environment variables.
Figure 4: Program Selection Screen

Figure 1: Login Screen
This takes the comptroller to the new purchase request approval screen, which replicates the functionality of the original screen (see figure 3), with some added features.

![Image of Purchase Approval Screen]

**Figure 2: Original Purchase Approval Screen**

If the OK button is selected, the next screen the comptroller sees is the Purchase Requisition Approval Queue, which allows the user to select from a list of purchase requests meeting a set of specified criteria. The user can choose to see any or all of the following:

- all new requests
- all requests that have been assigned to a buyer but not yet approved

9
- all approved requests
- a 'custom' query which the user can initiate by entering query information in one of the screen's columns.

![Figure 3: Purchase Requisition Approval Queue](image-url)

To use the queue, the comptroller will normally use one or more of the check boxes provided on the bottom of the screen to select new requisitions, and requisitions assigned to a buyer. After choosing the selection criteria for the query, the user presses the "query" button located in the lower left corner of the form. This will bring up a list of all requisitions meeting the specified criteria. This screen does not allow the user to edit any information. By placing the cursor in the desired record and pressing the "approval
screen” button in the lower right hand corner of the screen, the user is taken to the next screen, the Purchase Requisition Approval Screen (see figure 5).

In the Purchase Requisition Approval Screen, the comptroller will first verify the total cost of the purchase by checking technical publications, manufacturer’s catalogs, or other sources to verify that the unit price is correct. If the price is incorrect, the comptroller may correct it in the approval screen and that correction will update the database.

![Image of Approval Screen]

**Figure 4: Purchase Requisition Approval Screen**

Next, the comptroller will verify that the proper advice code (DLA, 1993), project code, and priority have been assigned to the purchase request. The advice and priority codes can be changed by using the drop-down boxes adjacent to the field. Each drop-down box will pull a list of values from the internal code of the form (alternate
method may be to have them pull the list from a look-up table). The project code is entered manually.

The comptroller then determines if the requisition falls into one of three special categories:

a. Casrep (Casualty Report). Does the requisition support repair to a ship that is covered by a casualty report (note: a casualty report indicates that the ability of the ship to perform its mission is degraded to some degree or another). If so, the requisition requires special handling to insure the material is received as quickly as possible. The comptroller will indicate this is the case by checking the CASREP check-box.

b. Continuing Service. Is the requisition intended to either contract for, or maintain a previously contracted continuing service (such as telephone support, copier maintenance, etc). If so, check the Continuing Service check-box.

c. Reimbursable. Will the money spent on this requisition be reimbursed from an outside activity. An example of this type of purchase might for repairs to a Coast Guard or Army piece of equipment where the amount spent on the repair will be returned by the agency that owns the equipment.

**Assign the purchase request to a buying agency.** Once all of these decisions have been made, the comptroller now has enough information to determine how the purchase will be made. Using the Req Type drop-down box, the comptroller will select from three choices:
a. NSN (National Stock Number) Purchase. The purchase request is for a standard stock item

![Valid Requisition Types](image)

**Figure 6: Requisition Types**

b. Credit Card Purchase. The purchase request is less than $2,500 and the material is not readily available in the standard stock system

c. Open Purchase. The purchase request is for more than $2,500 and is for material not readily available in the stock

If the material will be bought by credit card, the comptroller next assigns a buyer for the requisition by using the *Buyer* drop-down box. Pressing the down arrow will bring up a list of eligible buyers for the comptroller to select from.

![Valid Fund Codes](image)

**Figure 7: Assign a Fund Code**
**Assign the purchase request to a fund code.** Next, the comptroller assigns the purchase to a funding code. Funding codes are selected by using the *Fund Code* drop-down box in the lower center portion of the screen. Pressing the down arrow brings up a list of eligible fund codes (note: fund codes segregate purchases by appropriation and budget category as required by congressional oversight rules).

**Approve or disapprove the purchase request.** Once the comptroller has selected the fund code, the next step is to enter the approval code by using the *Approval Code* drop-down box located immediately beneath the *Fund Code* drop-down box.

Finally, the comptroller may choose to either close the window or proceed to the next request. To close the window, the comptroller will simply press the *Close Window* button at the bottom of the screen. To proceed to the next request, the comptroller may either use the arrow located at the top of the screen, or use the *Page Down* key on the keyboard.

Another choice the comptroller may make is disapproving the request (if it is not a legitimate purchase or if funds are not available). Pressing the *Disapprove* button located at the bottom left of the screen does this.

NSN-type purchase requests that have been previously approved, but not yet received may be administratively cancelled by pressing the *Admin Cancel* button at the bottom right of the screen.

This concludes the comptroller's function. If the request type was for NSN or open purchase, it will not be procured using the credit card purchasing system, and is of
no further interest. If the request was for a credit card purchase, it is next handed over to a buyer for action.

The buyer’s function is to:

- review the purchase request
- locate a vendor to supply the material
- contract for procurement and delivery of the material
- deliver the material to the ordering workcenter
- update the purchasing system to indicate the material has been received (can be waived).

**Review the Purchase Request.** The buyer uses the Credit Card Buyer’s screening queue to select purchase requests assigned for their action.

The Credit Card Buyer’s screen is reached by using the program selection screen (figure 1). The Buyer’s screen presents a list of all open purchase requests assigned to the buyer for action. It also allows the buyer to perform queries by using the *Query* menu option. Queries may select a particular purchase request, JCN (job control number), or range of dates. No editing of the purchase request may be performed from this screen.

The buyer selects the purchase request to be worked on by simply positioning the mouse cursor on the appropriate row, then pressing the left mouse button. This takes the buyer to the detail screen (see figure 9).
This screen provides the only on-line method for the buyer to view the details of the purchase request, as well as the instructions to the buyer located in the remarks section.

At the bottom of the screen are blocks for the buyer to indicate the purchase price of the material, as well as a free-form notes field. This notes field supports the following buyer functions:

- locate a vendor to supply the material

- contract for procurement and delivery of the material
Figure 9: Credit Card Buyer's Detail Screen

Update the purchasing system to indicate the material has been received. For this function the user returns to the standard program and indicates receipt in the PO Receiving screen (see figure 10).
Figure 10: PO Receiving Screen
Environment of the Project

The project environment can be divided into four categories: server, network, user, and client.

-Server-

Hardware: Hewlett Packard K420 server with dual 64-bit processors, 1 gigabyte RAM, 750 gigabyte of disk storage in a RAID 5 configuration

Operating System: HPUX version 10.20

Database: Oracle RDBMS version 7.3.4.4 providing row locking, contention management, and data integrity by session

-Network-

100BASET network with T1 (1.544 megabit) internet connection. Access to server controlled via INFOCRYPT digital encryption unit providing NSA certified protection up to and including FOOU and NOFORN level. (note: while there is no classified information stored in the database utilized by this application, there is another database resident on the same server that does require this protection.

-User-

Supports approximately 300 concurrent users. Average users has between eight and twelve years experience working with DOD logistics matters, two or more years experience working in a Microsoft Windows environment, and has received a two-day training class on CompassCONTRACT and related programs. All users are active duty
military personnel and consequently spend at most three years in a position before moving on. Because of this, extended training is impractical and the program must be as intuitive as possible.

-Client-

Hardware: PC compatible, Pentium 166 or better, minimum 32 megabyte RAM, minimum 1 gigabyte disk storage.

Operations System: Windows NT version 4.0 with service release 4.0

Oracle Form Runtime version 5.0
1. **JOB**. Interface table that connect CompassCONTRACT with the Regional Maintenance Automated Information System (RMAIS). Holds information about projects assigned to the organization for action.

2. **PB_TABLE**. The CompassCONTRACT project table. Accepts information from the JOB table (one-to-one relationship).

3. **SO_TABLE**. CompassCONTRACT's sales order table. In this environment, each project will normally generate a single sales order for completion and delivery, however there are some cases when there will be multiple sales orders for a single project.

4. **SOD_TABLE**. The detail table for the sales order. Each line item in the SOD table will relate to a sales order and will be for the delivery of some item in the IM table.

5. **ORD_TABLE**. CompassCONTRACT's work order table. Each work order in the table will normally be for the repair or manufacture of some item on a sales order line item. There may be a case when a work order will be for an internal repair with no corresponding sales order. Work orders also correspond to projects in the PB table.

6. **TRK_TABLE**. CompassCONTRACT's detail table for the work order. Each line item in the TRK table is a specific work order operation for a given number of man-hours.

7. **IM_TABLE**. CompassCONTRACT's item master table. All things that are manufactured or repaired must have an entry in this table, as must any part that is ordered to assist in the manufacture or repair of one of the items.

8. **PID_TABLE**. CompassCONTRACT's detail table with a one-to-one relationship for items in the IM table. Contains planning data, vendor names, reserved quantity, etc. for each part.

9. **PR_TABLE**. CompassCONTRACT's purchase request header file. Supplies the purchase request number, date created, date changed, etc. for purchase requests.

10. **PRD_TABLE**. CompassCONTRACT's detail table for purchase requests. Each record in the PRD table represents an individual part that has been placed on order. Parts must already exist in the IM and PID table, and be billable to a project located in the PB table.

11. **PR_INTERFACE**. Additional table added to CompassCONTRACT to record information specific to PRD records that could not be stored in the standard table. Has a one-to-one relationship with PRD_TABLE.

12. **PO_TABLE**. Purchase Order table. Each record in the Purchase order table represents a purchase request that has been approved.
13. **POD_TABLE**. Compass\textit{CONTRACT}'s Purchase Order detail table. Each record represents as specific part that has been requested, and the delivery schedule for that part. In this environment, there is normally a one-to-one relationship between PO and POD.

14. **POD_NOTES**. Additional table added to Compass\textit{CONTRACT} for the Credit Card module. Stores user notes and actual purchase costs.

15. **MILSTRIP**. Additional table added to Compass\textit{CONTRACT} to interface with the DOD logistics system. Stores the 80-column milstrip document along with information needed to link the milstrip to the purchase order in Compass\textit{CONTRACT}.

16. **FUND_TRANS**. Additional table added to Compass\textit{CONTRACT} to store information for interface with the DOD financial transaction reporting system, STARS-FL. Stores individual fund transactions for future extraction and reporting.

17. **FUND_CODES**. Additional table added to Compass\textit{CONTRACT} to store information for interface with the DOD financial transaction reporting system. Stores the account breakdown for each funding code with a congressional appropriation.

18. **BUDGET_CATS**. Additional table added to Compass\textit{CONTRACT} to store information for interface with the DOD financial transaction reporting system. Stores information about congressional appropriations.

19. **RCV_TABLE**. Compass\textit{CONTRACT}'s material receipt table. Stores information about each part received by the receiving dock, as well as the disposition of that part (who it was delivered to or what storage location the part was placed in). This will normally have a one-to-one relationship with the POD table, but there may be cases where a purchase order is received in two or more sections (vendor was only able to deliver a partial shipment one day, with the rest of the shipment following later).

20. **BT_TABLE**. Compass\textit{CONTRACT}'s buyer information table. Stores information about the buyers including name, code, phone number, and hierarchical information such as department and division.

21. **SNF_TABLE**. Compass\textit{CONTRACT}'s standard notes table. Stores notes for purchase requests, purchase orders, and work order operations. Each individual line of notes.

22. **BUDGET_CONTROL_TABLE**. Additional table added to Compass\textit{CONTRACT} to store information needed for interface with the MILSTRIP system. Contains information used to develop the PO serial number.
## User Screen-to-Database Table Matrix

<table>
<thead>
<tr>
<th>Table</th>
<th>Purchase Requisition Approval Queue</th>
<th>Purchase Requisition Approval Screen</th>
<th>Fund Code Assignment Screen</th>
<th>Credit Card Buyer Selection Screen</th>
<th>Credit Card Buyer's Screening Queue</th>
<th>Credit Card Buyer's Detail Screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT_TABLE</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>BUDGET CATS</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>FUND_CODES</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>FUND_TRANS</td>
<td>I,U</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>IM_TABLE</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>JOB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>MILSTRIP</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>ORD_TABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>PB_TABLE</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>PID_TABLE</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>PO_TABLE</td>
<td>I,U</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>POD_NOTES</td>
<td></td>
<td></td>
<td></td>
<td>S,U,I</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>POD_TABLE</td>
<td>I,U</td>
<td></td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>PR_TABLE</td>
<td>S</td>
<td>S,U</td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>PRD_TABLE</td>
<td>S</td>
<td>S,U</td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>PR_INTERFACES</td>
<td>S,U</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>RCV_TABLE</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>SNF_TABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>SO_TABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>SOD_TABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>TRK_TABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>

**Legend:**  
S = Select  
U = Update  
I = Insert  
D = Delete
Procedure

Client installation

The executable file F50RUN32.EXE (runtime version of Developer 2000) is installed on each workstation. The new forms (.FMX files) are installed on a Windows NT application server that maps to the client workstation’s “Y” drive. To facilitate user access to the new programs, the program selection screen has been modified.

CompassCONTRACT menu integration

The PGM2 table controls the program select screen (shown in figure 1). To modify the table and enable user access to the new programs, the following records were inserted into the PGM2 table:

<table>
<thead>
<tr>
<th>Mod</th>
<th>Funct</th>
<th>Description</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>21</td>
<td>Maintain Requisitions</td>
<td>F50RUN32.EXE Y:ORA10\PRAV.FMX</td>
</tr>
<tr>
<td>CA</td>
<td>25</td>
<td>Credit Card Buyers Queue</td>
<td>F50RUN32.EXE Y:ORA10\CREDIT.FMX</td>
</tr>
</tbody>
</table>

Using the SQL statements:

Insert into pgm2_table values ( 'CA', '21', 'Maintain Requisitions', , 'F50RUN32.EXE Y:ORA10\PRAV.FMX ', , '0', 'CS');

Insert into pgm2_table values ( 'CA', '21', 'Maintain Requisitions', , 'F50RUN32.EXE Y:ORA10\CREDIT.FMX ', , '0', 'CS');

Note: The existing database has two program schemas, WDS01 for production, WDS10 for testing and development. The contents of WDS01 are exported quarterly and imported to WDS10 to maintain a realistic development environment. Because SIMA uses version 7.3.3.6 of the Oracle database, all forms developed using Developer 2000 must be compiled specifically for the schema in which they will operate (Oracle version
7.3.4 and higher does not have this requirement). To decrease the administrative burden and make things easier for the system administrator and programming staff, the client side installations are set up so that all schema-specific files are stored in either a WDS01 or WDS10 directory. For production programs, there will always be two instances of the file, one in each directory.

Once the PGM2 table was been updated, the user menu began showing the two new programs. Access to the new programs is controlled via a built-in security program and must be granted by the system administrator (see figure 11 below).

![WDS Compass CONTRACT - Database 10](image)

**Figure 11: Granting user permission to access new programs**

Selecting the Cost Accounting Module, Function 21, displays the new requisition approval screen.
Initializing the program

As soon as the Purchase Requisition Approval form opens, a trigger fires (see listing 1). This trigger (WHEN_NEW_FORM_INSTANCE) gets the user's login name from the operating system, determines the current fiscal year (DOD's fiscal year starts October 1 and ends September 30), sets up the condition of the opening form, and then calls the opening form (Cinarkaya, B and Tushar Gadhia. 1998).

BEGIN
Set_Window_Property(FORMS_MDI_WINDOW, WINDOW_STATE, NORMAL);
Set_window_property(forms_mdi_window, title, 'Purchase Request Approval');
select upper(OSUSER) into :control.username
from v$session
where audsid = userenv('SESSIONID');
--get the fiscal year for the bc_grants block
select decode(to_char(sysdate,'MM'),
    '10',to_number(to_char(sysdate,'YYYY'))+1,
    '11',to_number(to_char(sysdate,'YYYY'))+1,
    '12',to_number(to_char(sysdate,'YYYY'))+1,
    to_char(sysdate,'YYYY'))
into :control.fy
from dual;
GO_ITEM('prd_Table.req_nbr');
:control.new := 'Y';
:control.assigned := 'Y';
:control.approved := 'Y';
DO_KEY('ENTER_QUERY');
END;

Listing 1: PRAV WHEN_NEW_FORM_INSTANCE trigger

The opening form displays on canvas 2 (see figure 12). The screen starts in query mode (triggered by the code (DO_KEY('ENTER_QUERY'))), waiting for user input to determine what records will display. The default is to display all records. However, in view of the size of the table (several hundred thousand records), the user will normally narrow the selection by use of the checkboxes in the bottom half of the screen. The user may also enter a query specification in one of the available blocks. If the user enters a
number, purchase order number, or buyer, the execution of the query is handled by the Developer 2000 runtime. Specifications entered by means of the check boxes feed a filter selection parameter attached to the prd_table block. This selection parameter decodes the checkboxes and (along with any selection criteria entered in the fields) determines what

---

**Figure 12: Purchase Approval Queue Canvas 2**

records will be retrieved when the *Query* button is pressed (the user may also press the F8 key to execute the query, or use the drop-down menu selections *Query*, *Execute*).
The Query button has a WHEN_BUTTON_PRESSED trigger attached to with the following code (Cinarkaya, B and Tushar Gadhia. 1998):

BEGIN
  GO_ITEM('prd_table.req_nbr');
  :global.rec_nav := 'TRUE';
  if (name_in('SYSTEM.MODE') != 'ENTER-QUERY') then
    do_key('ENTER_QUERY');
  else
    do_key('EXECUTE_QUERY');
  end if;
  :global.rec_nav := 'FALSE';
END;

Listing 2: PRAV Query button WHEN_BUTTON_PRESSED trigger

This trigger simply toggles the application back and forth between the enter query parameter mode and execute query mode. Because the trigger has to be executable regardless of the form mode (e.g. query or non-query) the function SYSTEM.MODE must be preceded by the keyword NAME_IN.

Once the query has been executed, all fields on the opening screen are filled in with the exception of the Description and Status fields. Those fields are filled by a POST_QUERY trigger attached to the PRD_TABLE block. The post_query trigger has the following code:

BEGIN
  -- message(prd_table.req_nbr);
  -- set the req type text
  --message('Status':|prd_table.status||'Note code_3':|prd_Table.note_code_3||'x');
  IF :prd_table.req_type = '1' THEN
    :prd_table.req_type_text := 'NSN Purchase';
  ELSIF :prd_table.req_type = '2' THEN
    :prd_table.req_type_text := 'Credit Card Purchase';
  ELSIF :prd_table.req_type = '4' THEN
    :prd_table.req_type_text := 'Open Purchase';
  ELSE
    :prd_table.req_type_text := 'Unknown req Type';
  END IF;

  -- See if this req's been received
  IF (:prd_table.status = 'C' and SUBSTR(prd_table.note_code_3,1,5) = 'ADCA') THEN
SELECT DECODE(COUNT(*),0,'N','Y')
INTO :prd_table.rcpt_flag
FROM rcv
WHERE rcv.po_nbr = RPAD(:prd_Table.po_nbr,10,'')
AND rcv.po_item_nbr = '0001';
END IF;
-- Determine if this req requires obligations:
-- If this req_type = 2 or the COG for the part starts w/ an even# then OBL_FLAG = N
-- The the req_type is 1 or 4 or the COG is odd or blank the OBL_FLAG=Y
:prd_table.obl_flag := 'Y';
BEGIN
-- force the first cog char to a number
-- if its a letter/blank fall to the exception and set obl flag
IF mod(to_number(substr(:prd_table.cog,1,1),'X'),2)=0 THEN
 :prd_table.obl_flag := 'N';
END IF;
EXCEPTION
WHEN value_error THEN
 :prd_table.obl_flag := 'Y';
END;

-- check the pr_intrefaces rec to see if it's a credit card purchase - no obl for credit cards
-- if the req_type = '2' then its a credit card purchase so NO OBL

IF :prd_table.req_type='2' THEN
 :prd_table.obl_flag := 'N';
END IF;

-- get ship name, hull nbr, and status
IF :prd_table.project != ' ' THEN
BEGIN
select nv(vdesc,''), nv(assy_desc,''),
decode(trim(wbs_type),'Y','ACTIVE',
'N','INACTIVE','OTHER')
into :prd_table.pb_vdesc,:prd_table.pb_assy_desc,:prd_table.ship_status
from pb_table
where substr(pb_table.project,1,5) = rpad(substr(:prd_table.project,1,5),5,'')
and rtrim(parnt_project) is null;
:prd_table.uic := substr(:prd_table.project,1,5);
EXCEPTION
WHEN NO_DATA_FOUND THEN
 :prd_table.pb_vdesc := '';
 :prd_table.pb_assy_desc := '';
 :PRD_TABLE.SHIP_STATUS := 'OTHER';
WHEN TOO_MANY_ROWS THEN
 MESSAGE('More than one record in PB_TABLE for Project'||:prd_table.project||'x');
 BELL;
END;
ELSE
 :prd_table.pb_vdesc := '';
 :prd_table.pb_assy_desc := '';
 :PRD_TABLE.SHIP_STATUS := 'OTHER';
END IF;
-- save the qty, price and calc values
:prd_Table.qty_hold := :prd_Table.qty;

30
:prd_Table.unit_prce_hold := :prd_Table.unit_prce;
:prd_Table.calvalue_hold := :prd_Table.calvalue;

IF (:prd_table.status is null and (:prd_table.buyer is null OR :prd_table.buyer = 'ZZ')) then
:prd_table.status_display := 'NEW';
:prd_table.status_abbr := 'NEW';

ELSIF (:prd_table.status is null and :prd_table.buyer is not null) then
:prd_table.status_display := 'ASSIGNED';
:prd_table.status_abbr := 'ASSG';

ELSIF (:prd_table.status = 'C' and SUBSTR(:prd_table.note_code_3,1,5) = 'ADCAN') then
:prd_table.status_display := 'ADMIN CAN';
:prd_table.status_abbr := 'ADCN';

ELSIF (:prd_table.status = 'C' and RTRIM(:prd_table.note_code_3) IS NULL) then
:prd_table.status_display := 'APPROVED';
:prd_table.status_abbr := 'APPR';

-- get the requisition type (type 1=A0A=NSN, type 2,4=A0E=NON-NSN)
begin

select NVL(SUBSTR(status,1,1),'S')
into :prd_table.edit_flag
from milstrip
where req_nbr = RPAD(:prd_table.req_nbr,10,'')
and req_item_nbr = RPAD(:prd_table.req_item_nbr,4,'')
and SUBSTR(doc_id,1,2) = 'A0'
and po_nbr = RPAD(:prd_table.po_nbr,10,'');

--message('Found milstrip...');

exception
when no_data_found then
  IF (SUBSTR(:prd_table.note_code_3,1,5) = 'ADCAN') THEN
    -- ok not to find one if the PR has been ADCAN'D.
    null;
  ELSE
    message('NO Milstrip for Req#/Item/Date:'||:prd_table.req_nbr||':||:prd_table.req_item_nbr;
    -- raise form_trigger_failure;
  END IF;
when Invalid Cursor State then
  message('More than One Milstrip Record found for Req_nbr='||:prd_table.req_nbr;
    ' Item='||:prd_table.req_item_nbr);
  raise form_trigger_failure;
end;

ELSE
  Message('Unknown Status Code for
Req#/Item/Date:'||:prd_table.req_nbr||':||:prd_table.req_item_nbr;
    '||:prd_table.date_reqd);
END IF;
End;

Listing 3: PRAV PRD_BLOCK Post Query Trigger
Preparing to update the purchase request

Once the POST_QUERY trigger completes, the user may now select the record that needs to be worked on by placing the mouse within the appropriate row and pressing the left mouse button. The user then uses the mouse to press the Approval button, which causes a WHEN_BUTTON_PRESSED trigger to fire. The trigger has the following code (Barnfield, L. 1997):

```
GO_ITEM('PRD_TABLE.REQ,NBR_MIRROR');
```

and will cause canvas 3 to display (see figure 13 below)

Canvas 3 is where the majority of the manipulation of records is performed. The top portion of the canvas displays information about the record being manipulated (the fields are presented to the user with a shaded background to indicate that they are not available for editing). The bottom portion of the form allows the user to determine what type the purchase will be by selecting from a list of values. The areas the user will be able to edit depends on a When_New_Record trigger (fires each time the cursor moves from one record to another) attached to the PRD block.

```plaintext
begin
  -- decode the pr status
  IF :system.current_item not in ('APP_REQD_01', 'PART', 'VDESC', 'PROJECT', 'REQ_NBR', 'REQ_ITEM_NBR') THEN
    GO_ITEM('CLOSE_WINDOW');
  END IF;

  IF (:prd_table.status is null and :prd_table.buyer is null) then
    -- status is new
    -- IF (:prd_table.req_nbr is not null) THEN
    --   Message('New requisition: Proceed by Assigning a Buyer or Selecting a Fund Code and Approval Code.);
    -- ELSE IF (:system.mode = 'ENTER-QUERY') THEN
    --   Message('Enter Query Criteria.....or Press CTRL-Q to Cancel Query');
    -- ELSE
    --   Null;
    -- END IF;
    set_item_property('FUND_CODE', enabled, PROPERTY_TRUE);
    set_item_property('FUND_CODE', updateable, PROPERTY_TRUE);
```
set_item_property('FUND_CODE', insert_allowed, PROPERTY_TRUE);
set_item_property('APP_01', enabled, PROPERTY_TRUE);
set_item_property('APP_01', updateable, PROPERTY_TRUE);
set_item_property('APP_01', insert_allowed, PROPERTY_TRUE);
set_item_property('QTY', enabled, PROPERTY_TRUE);
set_item_property('QTY', updateable, PROPERTY_TRUE);
set_item_property('QTY', insert_allowed, PROPERTY_TRUE);
set_item_property('UNIT_PRC', enabled, PROPERTY_TRUE);
set_item_property('UNIT_PRC', updateable, PROPERTY_TRUE);
set_item_property('UNIT_PRC', insert_allowed, PROPERTY_TRUE);
set_item_property('BUYER', enabled, PROPERTY_TRUE);
set_item_property('BUYER', updateable, PROPERTY_TRUE);
set_item_property('BUYER', insert_allowed, PROPERTY_TRUE);
set_item_property('DISAPPROVE', enabled, PROPERTY_TRUE);
set_item_property('DISAPPROVE', navigable, PROPERTY_TRUE);
set_item_property('DISAPPROVE', mouse_navigate, PROPERTY_TRUE);
set_item_property('ADMIN_CANCEL', enabled, PROPERTY_FALSE);
set_item_property('ADMINCANCEL', navigable, PROPERTY_FALSE);
set_item_property('ADMIN_CANCEL', mouse_navigate, PROPERTY_FALSE);
set_item_property('REQ_TYPE', enabled, PROPERTY_TRUE);
set_item_property('REQ_TYPE', updateable, PROPERTY_TRUE);
set_item_property('REQ_TYPE', insert_allowed, PROPERTY_TRUE);
set_item_property('ADVICE_CODE', enabled, PROPERTY_TRUE);
set_item_property('ADVICE_CODE', updateable, PROPERTY_TRUE);
set_item_property('ADVICE_CODE', insert_allowed, PROPERTY_TRUE);
set_item_property('CASREP', enabled, PROPERTY_TRUE);
set_item_property('CASREP', updateable, PROPERTY_TRUE);
set_item_property('CASREP', insert_allowed, PROPERTY_TRUE);
set_item_property('REIMBURSABLE', enabled, PROPERTY_TRUE);
set_item_property('REIMBURSABLE', updateable, PROPERTY_TRUE);
set_item_property('REIMBURSABLE', insert_allowed, PROPERTY_TRUE);
set_item_property('CONT_SERVICE', enabled, PROPERTY_TRUE);
set_item_property('CONT_SERVICE', updateable, PROPERTY_TRUE);
set_item_property('CONT_SERVICE', insert_allowed, PROPERTY_TRUE);
set_item_property('ADVICE_CODE', enabled, PROPERTY_TRUE);
set_item_property('ADVICE_CODE', updateable, PROPERTY_TRUE);
set_item_property('ADVICE_CODE', insert_allowed, PROPERTY_TRUE);
set_item_property('DELIVER_TO', enabled, PROPERTY_TRUE);
set_item_property('DELIVER_TO', updateable, PROPERTY_TRUE);
set_item_property('DELIVER_TO', insert_allowed, PROPERTY_TRUE);
ELSIF (:prd_table.status is null and :prd_table.buyer is not null) then

-- status is assigned to a buyer

message('Assigned Requisition: Proceed by Selecting a Fund Code and Approval Code.');

set_item_property('BUYER', enabled, PROPERTY_TRUE);
set_item_property('BUYER', updateable, PROPERTY_TRUE);
set_item_property('BUYER', insert_allowed, PROPERTY_TRUE);

set_item_property('FUND_CODE', enabled, PROPERTY_TRUE);
set_item_property('FUND_CODE', updateable, PROPERTY_TRUE);
set_item_property('FUND_CODE', insert_allowed, PROPERTY_TRUE);

set_item_property('APP_01', enabled, PROPERTY_TRUE);
set_item_property('APP_01', updateable, PROPERTY_TRUE);
set_item_property('APP_01', insert_allowed, PROPERTY_TRUE);

set_item_property('QTY', enabled, PROPERTY_TRUE);
set_item_property('QTY', updateable, PROPERTY_TRUE);
set_item_property('QTY', insert_allowed, PROPERTY_TRUE);

set_item_property('UNIT_PRC', enabled, PROPERTY_TRUE);
set_item_property('UNIT_PRC', updateable, PROPERTY_TRUE);
set_item_property('UNIT_PRC', insert_allowed, PROPERTY_TRUE);

set_item_property('DISAPPROVE', enabled, PROPERTY_TRUE);
set_item_property('DISAPPROVE', navigable, PROPERTY_TRUE);
set_item_property('DISAPPROVE', mouse_navigate, PROPERTY_TRUE);

set_item_property('ADMIN_CANCEL', enabled, PROPERTY_FALSE);
set_item_property('ADMIN_CANCEL', navigable, PROPERTY_FALSE);
set_item_property('ADMIN_CANCEL', mouse_navigate, PROPERTY_FALSE);

set_item_property('REQ_TYPE', enabled, PROPERTY_TRUE);
set_item_property('REQ_TYPE', updateable, PROPERTY_TRUE);
set_item_property('REQ_TYPE', insert_allowed, PROPERTY_TRUE);

set_item_property('ADVICE_CODE', enabled, PROPERTY_TRUE);
set_item_property('ADVICE_CODE', updateable, PROPERTY_TRUE);
set_item_property('ADVICE_CODE', insert_allowed, PROPERTY_TRUE);

set_item_property('CASREP', enabled, PROPERTY_TRUE);
set_item_property('CASREP', updateable, PROPERTY_TRUE);
set_item_property('CASREP', insert_allowed, PROPERTY_TRUE);

set_item_property('REIMBURSABLE', enabled, PROPERTY_TRUE);
set_item_property('REIMBURSABLE', updateable, PROPERTY_TRUE);
set_item_property('REIMBURSABLE', insert_allowed, PROPERTY_TRUE);

set_item_property('CONT_SERVICE', enabled, PROPERTY_TRUE);
set_item_property('CONT_SERVICE', updateable, PROPERTY_TRUE);
set_item_property('CONT_SERVICE', insert_allowed, PROPERTY_TRUE);
set_item_property('ADVICE_CODE', enabled, PROPERTY_TRUE);
set_item_property('ADVICE_CODE', updateable, PROPERTY_TRUE);
set_item_property('ADVICE_CODE', insert_allowed, PROPERTY_TRUE);

set_item_property('DELIVER_TO', enabled, PROPERTY_TRUE);
set_item_property('DELIVER_TO', updateable, PROPERTY_TRUE);
set_item_property('DELIVER_TO', insert_allowed, PROPERTY_TRUE);

set_item_property('REQUEST_DEPT', enabled, PROPERTY_TRUE);
set_item_property('REQUEST_DEPT', updateable, PROPERTY_TRUE);
set_item_property('REQUEST_DEPT', insert_allowed, PROPERTY_TRUE);

ELSIF (:prd_table.status = 'C') and (SUBSTR(:prd_table.note_code,3,1,5) = 'ADCAN') THEN
  MESSAGE('This Requisition has been ADMIN Cancelled, so you can only view this Requisition.');
  SET_VIEW_ONLY;

ELSIF (:prd_table.status = 'C') and (:prd_table.req_type = '1' or :prd_table.req_type = '4') THEN
  MESSAGE('Approved Type 1 or 4 Requisition: You can Edit the Unit Price, but not the Qty.');
  set_item_property('REQ_TYPE', enabled, PROPERTY_FALSE);
  set_item_property('REQ_TYPE', updateable, PROPERTY_FALSE);
  set_item_property('REQ_TYPE', insert_allowed, PROPERTY_FALSE);

  set_item_property('ADVICE_CODE', enabled, PROPERTY_TRUE);
  set_item_property('ADVICE_CODE', updateable, PROPERTY_TRUE);
  set_item_property('ADVICE_CODE', insert_allowed, PROPERTY_TRUE);

  set_item_property('CASREP', enabled, PROPERTY_FALSE);
  set_item_property('CASREP', updateable, PROPERTY_FALSE);
  set_item_property('CASREP', insert_allowed, PROPERTY_FALSE);

  set_item_property('REIMBURSABLE', enabled, PROPERTY_FALSE);
  set_item_property('REIMBURSABLE', updateable, PROPERTY_FALSE);
  set_item_property('REIMBURSABLE', insert_allowed, PROPERTY_FALSE);

  set_item_property('CONT_SERVICE', enabled, PROPERTY_FALSE);
  set_item_property('CONT_SERVICE', updateable, PROPERTY_FALSE);
  set_item_property('CONT_SERVICE', insert_allowed, PROPERTY_FALSE);

  set_item_property('QTY', enabled, PROPERTY_FALSE);
  set_item_property('QTY', updateable, PROPERTY_FALSE);
  set_item_property('QTY', insert_allowed, PROPERTY_FALSE);

  set_item_property('UNIT_PRC', enabled, PROPERTY_TRUE);
  set_item_property('UNIT_PRC', updateable, PROPERTY_TRUE);
  set_item_property('UNIT_PRC', insert_allowed, PROPERTY_TRUE);

  set_item_property('BUYER', enabled, PROPERTY_FALSE);
  set_item_property('BUYER', updateable, PROPERTY_FALSE);
  set_item_property('BUYER', insert_allowed, PROPERTY_FALSE);

  set_item_property('FUND_CODE', enabled, PROPERTY_FALSE);

set_item_property('FUND_CODE', updateable, PROPERTY_FALSE);
set_item_property('FUND_CODE', insert_allowed, PROPERTY_FALSE);

set_item_property('APP_01', enabled, PROPERTY_FALSE);
set_item_property('APP_01', updateable, PROPERTY_FALSE);
set_item_property('APP_01', insert_allowed, PROPERTY_FALSE);

IF (SUBSTR(:prd_table.edit_flag,1,1) = 'S' or :prd_table.rcpt_flag = 'Y') THEN
set_item_property('DISAPPROVE', enabled, PROPERTY_FALSE);
set_item_property('DISAPPROVE', navigable, PROPERTY_FALSE);
set_item_property('DISAPPROVE', mouse_navigate, PROPERTY_FALSE);
ELSE
set_item_property('DISAPPROVE', enabled, PROPERTY_TRUE);
set_item_property('DISAPPROVE', navigable, PROPERTY_TRUE);
set_item_property('DISAPPROVE', mouse_navigate, PROPERTY_TRUE);
END IF;

set_item_property('ADMIN_CANCEL', enabled, PROPERTY_TRUE);
set_item_property('ADMIN_CANCEL', navigable, PROPERTY_TRUE);
set_item_property('ADMIN_CANCEL', mouse_navigate, PROPERTY_TRUE);

set_item_property('ADVICE_CODE', enabled, PROPERTY_TRUE);
set_item_property('ADVICE_CODE', updateable, PROPERTY_TRUE);
set_item_property('ADVICE_CODE', insert_allowed, PROPERTY_TRUE);

set_item_property('DELIVER_TO', enabled, PROPERTY_TRUE);
set_item_property('DELIVER_TO', updateable, PROPERTY_TRUE);
set_item_property('DELIVER_TO', insert_allowed, PROPERTY_TRUE);

set_item_property('REQUEST_DEPT', enabled, PROPERTY_TRUE);
set_item_property('REQUEST_DEPT', updateable, PROPERTY_TRUE);
set_item_property('REQUEST_DEPT', insert_allowed, PROPERTY_TRUE);

ELSIF (:prd_table.status = 'C') and (:prd_table.req_type = '2') then
message('Approved Type 2 Requisition: You can Edit the Price and Qty.);
BEGIN
IF :prd_table.rcpt_flag = 'Y' THEN
set_item_property('DISAPPROVE', enabled, PROPERTY_FALSE);
set_item_property('DISAPPROVE', navigable, PROPERTY_FALSE);
set_item_property('DISAPPROVE', mouse_navigate, PROPERTY_FALSE);
ELSE
set_item_property('DISAPPROVE', enabled, PROPERTY_TRUE);
set_item_property('DISAPPROVE', navigable, PROPERTY_TRUE);
set_item_property('DISAPPROVE', mouse_navigate, PROPERTY_TRUE);
END IF;
END;

set_item_property('ADMIN_CANCEL', enabled, PROPERTY_TRUE);
set_item_property('ADMIN_CANCEL', navigable, PROPERTY_TRUE);
set_item_property('ADMIN_CANCEL', mouse_navigate, PROPERTY_TRUE);

set_item_property('ADVICE_CODE', enabled, PROPERTY_FALSE);
set_item_property('ADVICE_CODE', updateable, PROPERTY_FALSE);
set_item_property('ADVICE_CODE', insert_allowed, PROPERTY_FALSE);
ELSIF (:prd_table.status = 'D') THEN

MESSAGE('This Requisition has been Closed, so you can only view this Requisition.');
SET_VIEW_ONLY;

ELSE
-- record fails all edit rules so set edit capability to none to prevent invalid edits

MESSAGE('This Requisition does NOT meet any of the EDIT criteria, so you can only view this Requisition. ');
SET_VIEW_ONLY;
END IF;

IF :prd_table.cont_service = 'Y' THEN
  set_item_property('QTY', enabled, PROPERTY_FALSE);
  set_item_property('QTY', updateable, PROPERTY_FALSE);
  set_item_property('QTY', insert_allowed, PROPERTY_FALSE);
END IF;

END;

Listing 4: PRAV PRD Block When_New_Record trigger

Once canvas 3 displays, the user can edit the record, correcting any information that has been entered incorrectly by the person placing the order. The user can update the type of purchase order that will be generated from the request, set parameters such as continuing services, CASREP, buyer, price, quantity, various codes, priority, etc. When the record meets all requirements, the user selects a fund code to deduct the purchase price from, then selects an approval code that identifies the record as ready for transmission.

The req_type drop down box has the following trigger attached to it:

begin
  go_item('prd_table.req_type');
  do_key('list_values');
end;

Listing 5: PRAV Req_Type Drop Down Box trigger

38
Figure 13: Purchase Approval Queue Canavas 3

This trigger causes a list of values to be displayed. The contents of the list are static with a value of 1 indicating standard NSN purchase, 2 indicating credit card purchase, and 4 indicating open purchase.

The Casrep checkbox has no code attached to it, but is simply used to place a “W” in the fifth character of the PO_NBR field. This indicates to the purchasing system that the requisition has a higher than normal priority because the part is needed to restore a mission critical system to operation.

If the continuing services block is selected, the requisition is transmitted with “C999” in the quantity block. This indicates to the purchasing system that the request will be billed for over a set time period (such as for telephone services or copier repair).
In order for a part to qualify as a continuing services, the requisition type must be set to
"4" and the COG (cognizance code) of the part must be blank. This condition is verified
when the box is checked by a WHEN_VALIDATE_ITEM trigger attached to the check
box. The code is as follows:

```
DECLARE
    alert_id ALERT := Find_Alert('generic_alert');
    dummy_var NUMBER;
    alert_id_FAIL ALERT := FIND_ALERT('alert_fail');
BEGIN
    IF :cont_service = 'Y' and :prd_table.req_type != '4' THEN
        Set_Alert_Property(alert_id, ALERT_MESSAGE_TEXT,
                           'The Req Type must be a type 4 - Open Purchase to be a Continuing Service.');
        -- now show the alert
        :cont_service := 'N';
        dummy_var := Show_Alert(alert_id);
        RAISE FORM_TRIGGER_FAILURE;
    ELSIF (:cont_service = 'Y' and RTRIM(:prd_table.cog) IS NOT NULL) THEN
        Set_Alert_Property(alert_id_FAIL, ALERT_MESSAGE_TEXT,
                           'The COG must be BLANK to be a Continuing Service.');
        -- now show the alert
        :prd_table.cont_service := 'N';
        dummy_var := Show_Alert(alert_id_FAIL);
        RAISE FORM_TRIGGER_FAILURE;
    ELSE
        :prd_table.qty := 1;
        set_item_property('QTY', NAVIGABLE, PROPERTY_FALSE);
    END IF;
    IF :cont_service = 'N' THEN
        set_item_property('QTY', enabled, PROPERTY_TRUE);
        set_item_property('QTY', updateable, PROPERTY_TRUE);
        set_item_property('QTY', insert_allowed, PROPERTY_TRUE);
    END IF;
END;
```

**Listing 6: PRAV Continuing Services WHEN_VALIDATE_ITEM trigger**

Like the Casrep checkbox, the next checkbox, the Reimbursable checkbox, also
alters the PO_NBR. There are no triggers attached to the Reimbursable checkbox.

The Buyer drop-down box allows the user to select from a list of values. This is a
dynamic list populated with values returned by the following SQL statement:

```
select buyer, name from bt;
```
Unit Price and quantity (QTY) fields will both be updateable, and the extended price (Ext. Cost) field calculates the product of the two.

The advice code (Adv code) drop-down box presents another static list of values. The user is not limited to the items in the list (the advice code is a 2-character alphanumeric code with over a thousand possible entries, only the five most common are displayed in the list).

The project code (Proj Cd) field is user updateable. It defaults to a standard value (LK5) but may be changed if the part is being ordered to support something other than the repair of minesweeping vessels.

The Priority drop-down box displays another static list of values. With this field, the user is limited to only those values presented in the list.

The Fund Code drop-down box presents a dynamic list of values. The list is generated by the SQL statement:

```sql
SELECT appn, subhead, bud_cat, fund_code, fund_code_desc,
       LPAD(TO_CHAR(balance,'$999,999,999.09'),16,' ') Balance,
       fund_code_seq
FROM fund_codes, budget_cats
WHERE fund_codes.bc_seq = budget_cats.bc_seq
  AND (:prd_table.ship_status||bud_cat||substr(appn,4,4) =
    'ACTIVE'||'3BIMA'||'1804' OR
    :prd_table.ship_status||bud_cat||substr(appn,4,4) =
    'INACTIVE'||'3BIMA'||'1806' or :prd_table.ship_status = 'OTHER')
  AND substr(appn,3,1) = substr(:control.fy,4,1)
ORDER BY appn, subhead, bud_cat, fund_code
```

**Listing 7: PRAV Fund Code list-of-values selection statement**

This list responds to the customer’s status (active, reserve, or other) and only allows the user to select fund codes appropriate for the customer. The customer status is determined by a section of the POST_QUERY trigger (see listing 3).

41
The Approval Code drop down box displays a dynamic list of values based on the SQL statement:

```
select app,name from am_table
```

The user has three command buttons available which appear at the bottom of the screen. The Disapprove button has a WHEN_BUTTON_PRESSED trigger associated with it (see listing 8 below)

```
DECLARE
  hold_alert number;
  adj_flag varchar2(3);
BEGIN
  IF (:prd_table.status = 'D') THEN
    hold_alert := show_alert('ALREADY_DISAPP_CANNED');
    raise form_trigger_failure;
  END IF;

  -- reset the quantity and unit price back to the original value
  -- because you can not change the unit price or qty on a disapproval

  :prd_table.unit_prc := :prd_table.unit_prc_hold;
  :prd_table.qty := :prd_table.qty_hold;
  :prd_table.calcvalue := :prd_table.unit_prc_hold * :prd_table.qty_hold;

  --
  -- check to see if the po has already been received
  -- create adj obligation record if hasn't been received
  -- or if it is a Continuing Service req which aren't received
  -- show alert if it has been received

  --
  IF (:prd_table.status = 'C') THEN
    SELECT DECODE(count(*),0,'YES','NO')
    INTO adj_flag
    FROM rcv
    WHERE po_nbr = RPAD(NVL(:prd_table.po_nbr,'X'),10,'')
      AND po_item_nbr = '0001';

    -- message('after select from rcv');
    -- fail if adj_flag = NO

    IF (adj_flag = 'YES' and :prd_table.obl_flag = 'Y' and :prd_table.req_type != '2') THEN
      INSERT INTO fund_trans
      (tran_type,tran_amt,tran_date,created_by,fund_code_seq,req_nbr,req_item_nbr,fy,date_reqd)
      VALUES
      ('ADJ',:prd_table.calcvalue_hold*-1,sysdate,control.username,:prd_table.fund_code_seq,
       :prd_table.req_nbr,:prd_table.req_item_nbr,control.fy,:prd_table.date_reqd);
    END IF;
  END IF;
```

```
-- create an expense record for continuing services
IF (:prd_table.cont_service = 'Y') THEN
```

```
```
INSERT INTO fund_trans

(tran_type, tran_amt, tran_date, created_by, fund_code_seq, req_nbr, req_item_nbr, fy, date_reqd)
VALUES
('EXP', prd_table.calcvalue_hold*-1, sysdate, :control.username, :prd_table.fund_code_seq,
 :prd_table.req_nbr, :prd_Table.req_item_nbr, :control.fy, :prd_table.date_reqd);
END IF;
ELSEIF (adj_flag = 'NO') THEN
  hold_alert := show_alert('RECEIVED');
  raise form_trigger_failure;
END IF;
ELSE
  null;
END IF;
END IF;
:prd_table.status := 'D';
:prd_table.app_date_01 := null;
DO KEY('commit_form');
GO_ITEM('prd_table.close_window');
set_view_only;
END;

Listing 8: PRAV Disapprove Button WHEN_BUTTON_PRESSED trigger

This trigger checks to see if this request has been previously approved. If not, the request is flagged as deleted by changing the PRD_TABLE.STATUS to "D". If the request has been previously approved, but not yet placed on order, the funds obligated to the request are adjusted, and the request is deleted. If the request has been previously approved and the order has already been generated, an error message to inform the user the request can no longer be disapproved is displayed.

The Close Window button has a WHEN_BUTTON_PRESSED trigger associated with it that restores the form’s properties to read-only and hides canvas 3 by hiding its display window (window 2).

begin
  -- set_block_property('PRD_TABLE', 'CURRENT_RECORD_ATTRIBUTE', 'VIS_ATT_BLUE');
go_item('prd_table.app_reqd_01');

  set_item_property('FUND_CODE', enabled, PROPERTY_FALSE);
  set_item_property('FUND_CODE', updateable, PROPERTY_FALSE);
  set_item_property('FUND_CODE', insert_allowed, PROPERTY_FALSE);

  set_item_property('APP_01', enabled, PROPERTY_FALSE);
  set_item_property('APP_01', updateable, PROPERTY_FALSE);

end;
set_item_property('APP_01', insert_allowed, PROPERTY_FALSE);

set_item_property('APP_DATE_01', enabled, PROPERTY_FALSE);
set_item_property('APP_DATE_01', updateable, PROPERTY_FALSE);
set_item_property('APP_DATE_01', insert_allowed, PROPERTY_FALSE);

set_item_property('QTY', enabled, PROPERTY_FALSE);
set_item_property('QTY', updateable, PROPERTY_FALSE);
set_item_property('QTY', insert_allowed, PROPERTY_FALSE);

set_item_property('UNIT_PRC', enabled, PROPERTY_FALSE);
set_item_property('UNIT_PRC', updateable, PROPERTY_FALSE);
set_item_property('UNIT_PRC', insert_allowed, PROPERTY_FALSE);

set_item_property('BUYER', enabled, PROPERTY_FALSE);
set_item_property('BUYER', updateable, PROPERTY_FALSE);
set_item_property('BUYER', insert_allowed, PROPERTY_FALSE);

set_item_property('ADMIN_CANCEL', enabled, PROPERTY_FALSE);
set_item_property('ADMIN_CANCEL', navigable, PROPERTY_FALSE);
set_item_property('ADMIN_CANCEL', mouse_navigate, PROPERTY_FALSE);

hide_window('WINDOW2');
end;

Listing 9: PRAV Close Window WHEN_BUTTON_PRESSED trigger

The administrative cancel (Admin. Cancel) button has a WHEN_BUTTON_PRESSED trigger associated with it that will allow the user to cancel a request that has already been transmitted, and even received. The trigger will adjust the fund obligation to reverse any expense generated by the original requisition and set the request status to “D.” Administrative cancels can be differentiated from disapproved requests by the note code “ADCAN” inserted into the note_codes_3 field of the PRD table.

DECLARE
    hold_alert number;
    rcv_flag VARCHAR2(3);
BEGIN
    IF (:prd_table.status = 'C' and NVL(substr(:prd_table.note_code_3,1,5),'X') = 'ADCAN') THEN
        hold_alert := show_alert('ALREADY_DISAPP_CANNED');
        raise form_trigger_failure;
    END IF;
    -- reset the quantity and unit price back to the original value
    -- because you can not change the unit price or qty on a disapproval
    :prd_table.unit_prc := :prd_table.unit_prc_hold;
    :prd_table.qty := :prd_table.qty_hold;
    :prd_table.calcvalue := :prd_table.unit_prc_hold * :prd_table.qty_hold;

44
IF (:prd_table.obl_flag = 'Y' and :prd_table.req_type != '2') THEN
  INSERT INTO fund_trans
  (tran_type,tran_amt,tran_date,created_by,fund_code_seq,req_nbr,req_item_nbr,fy,date_reqd)
  VALUES
  ('ADJ',:prd_table.calcvalue_hold*-1,sysdate,control.username,:prd_table.fund_code_seq,
   :prd_table.req_nbr,:prd_Table.req_item_nbr,control.fy,:prd_table.date_reqd);

  SELECT DECODE(count(*),0,'NO','YES') INTO rcv_flag
  FROM rcv_table
  WHERE po_nbr = RPAD(NVL(:prd_table.po_nbr,'X'),10,'')
  AND po_item_nbr = '0001';

  IF (rcv_flag = 'YES') THEN
    INSERT INTO fund_trans
    (tran_type,tran_amt,tran_date,created_by,fund_code_seq,req_nbr,req_item_nbr,fy,date_reqd)
    VALUES
    ('EXP',:prd_table.calcvalue_hold*-1,sysdate,control.username,:prd_table.fund_code_seq,
     :prd_table.req_nbr,:prd_Table.req_item_nbr,control.fy,:prd_table.date_reqd);
  END IF;
END IF;

-- set the status to C
-- and set the note_code_2 to FORM_EVENT so the trigger on the PRD_PRE_UPD table will
-- create a milstrip
:prd_table.status := 'C';
:prd_table.note_code_3 := 'ADCAN';
:prd_table.note_code_2 := 'FORM_EVENT';

DO KEY('commit_form');
go_item('close_window');
sel_view_only;
END;

Listing 10: PRAV Administrative Cancel When_Button_Pressed trigger

Update the Purchase Request & create Purchase Order. MILSTRIP

Once the user has made any necessary changes to the record, the user presses the
"Save" menu button or uses the "Save" menu item. This fires three triggers in succession,
the pre_update, on_update, and post_update triggers for the prd_table block. The
pre_update trigger attached to the form updates the fund. Code for the pre_update
trigger:

DECLARE
dummy NUMBER := 0;
hold_alert NUMBER := 0;
adj_amt NUMBER := 0;
avail_balance NUMBER := 0;
v_tran_type VARCHAR2(3) := NULL;
alert_id_fail alert := FIND_ALERT('ALERT_FAIL');

--
-- Purpose:
-- 1) Create fund transactions during the approval process and for modifications to the requisition
-- after it has been approved.
-- 2) Update related tables affected by changes to the unit price of an item or the qty ordered
--
-- Transaction Notes:
--   Req Type 1 = NSN Purchase (ALWAYS create an obligation unless the COG starts w/ an EVEN#)
--   Req Type 2 = Credit Card Purchase (NEVER create obligation or expense transaction)
--   Req Type 4 = OPEN Purchase(ALWAYS create an obligation unless the COG starts w/anEVEN#)
--   req Type 4 = OPEN Purchase which is a "Continuing Service" (Always create an Expense record)
--
BEGIN

--
-- Process approved orders first: If the user changed the unit price or qty ordered create
-- additional fund transactions
--
IF (:prd_table.status = 'C' and NVL(SUBSTR(:prd_table.note_code_3,1,5),''),X') != 'ADCAN')
AND ((:prd_table.qty != :prd_table.qty_hold) or (:prd_table.unit_prc != :prd_table.unit_prc_hold))
THEN

adj_amt := :prd_table.calcvalue - :prd_table.calcvalue_hold;

IF (adj_amt != 0 and :prd_table.obl_flag = 'Y' and :prd_table.req_type != '2') THEN

-- Confirm that the budget category has enough $$ to increase the obligation

IF (adj_amt > 0 ) THEN
SELECT balance INTO avail_balance
FROM budget_cats bc, fund_codes fc
WHERE fc.fund_code_seq = :prd_table.fund_code_seq
AND bc.bc_seq = fc.bc_seq;
IF (avail_balance < adj_amt) THEN
message('Fund Code Balance is :||to_char(avail_balance,'$999,999,990.00')));
hold_alert := show_alert('NOT_ENUF_ADJ $$');
raise form_trigger_failure;
END IF;
END IF;

-- create OBL/ADJ transaction

v_tran_type := 'ADJ';
IF (:prd_table.cont_service = 'Y') THEN
  v_tran_type := 'OBL';
END IF;
INSERT INTO fund_trans
(tran_type, tran_amt, tran_date, created_by, fund_code_seq, req_nbr, req_item_nbr, fy, date_reqd, ref_nbr)
VALUES
(v_tran_type, adj_amt, sysdate, control.username, prd_table.fund_code_seq,
 :prd_table.req_nbr, :prd_table.req_item_nbr, control.fy, prd_table.date_reqd,
 :prd_table.po_nbr);

-- create an EXP transaction for Req Type 4 Continuing Services
IF (:prd_table.cont_service = 'Y') THEN
  INSERT INTO fund_trans
  (tran_type, tran_amt, tran_date, created_by, fund_code_seq, req_nbr, req_item_nbr, fy,
   date_reqd, ref_nbr)
  VALUES
  ('EXP', adj_amt, sysdate, control.username, prd_table.fund_code_seq,
   :prd_table.req_nbr, :prd_table.req_item_nbr, control.fy, prd_table.date_reqd,
   :prd_table.po_nbr);
  END IF;
END IF; -- adj_amt != 0

-- update the PO and POD if the value of the PRD changed
IF (:prd_table.unit_prp != :prd_table.unit_prp_hold or :prd_table.qty != :prd_table.qty_hold) THEN
  UPDATE pod_table
  SET unit_prp = :prd_table.unit_prp,
      qty_ord = :prd_table.qty
  WHERE po_nbr = RPAD(:prd_table.po_nbr,10,'')
  AND po_item_nbr = '0001';

  IF (:prd_table.calvalue != :prd_table.calvalue_hold) THEN
    UPDATE po_table
    SET value = value + adj_amt
    WHERE po_nbr = RPAD(:prd_table.po_nbr,10,'');
  END IF;
END IF; -- END process changes to unit_price or qty
END IF; -- END create new OBL/ADJ/EXP transactions for previously approved reqs

--
-- Process new approvals: Validate the fund code and approval code and
-- create new transactions
--

IF (:prd_table.fund_code||:prd_table.app_01 = NVL(:prd_table.fund_code,'X')||NVL(:prd_table.app_01,'X') AND
    NVL(:prd_table.status,'X') NOT IN ('D','C')) THEN
-- Create the obligation if the COG does not start w/ an even# and the Req Type = 1 or 4

IF (:prd_table.obl_flag = 'Y' and :prd_table.req_type != '2' and :prd_table.calcvalue != 0) then

-- check fund balance

SELECT balance
FROM budget_cats bc, fund_codes fc
WHERE fc.fund_code_seq = :prd_table.fund_code_seq
AND bc.bc_seq = fc.bc_seq;
IF (avail_balance < :prd_table.calcvalue) THEN
:prd_table.fund_code_seq := null;
:prd_table.fund_code := null;
:prd_table.subhead := null;
:prd_table.appn := null;
:prd_table.bud_cat := null;
:prd_table.app_date_01 := null;
:prd_table.app_01 := null;
hold_alert := show_alert('NOT_ENUF_$$');
raise form_trigger_failure;
END IF;

INSERT INTO fund_trans
(tran_type,tran_amt,tran_date,created_by,fund_code_seq,req_nbr,req_item_nbr,ty,req_date_reqd) VALUES
('OBL',:prd_table.calcvalue,sysdate,control.username,:prd_table.fund_code_seq,
:prd_table.req_nbr,:prd_table.req_item_nbr,control.ty,:prd_table.date_reqd);

-- create an EXP record for Req Type = 4 Continuing Service reqs

IF (:prd_table.cont_service = 'Y') THEN
INSERT INTO fund_trans
(tran_type,tran_amt,tran_date,created_by,fund_code_seq,req_nbr,req_item_nbr,ty,req_date_reqd) VALUES
('EXP',:prd_table.calcvalue,sysdate,control.username,:prd_table.fund_code_seq,
:prd_table.req_nbr,:prd_table.req_item_nbr,control.ty,:prd_table.date_reqd);
END IF;
END IF; -- create the OBL/EXP records for new approvals

-- set the PRD fields for approval

:prd_table.status := 'C';
:prd_table.date_1st_app := sysdate;
:prd_table.app_date_01 := sysdate;
:prd_table.app_user_01 := control.username;

ELSIF (:prd_table.fund_code || :prd_table.app_01 != NVL(:prd_table.fund_code,'X')||NVL(:prd_table.app_01,'X') AND NVL(:prd_table.status,'X') NOT IN ('D','C')) THEN

48
Set_Alert_Property(alert_id_fail, ALERT_MESSAGE_TEXT,
   'You must enter a Fund Code and Approval Code to Approve a Requistion.');
   
   -- now show the alert
dummy := Show_Alert(alert_id_fail);
RAISE FORM_TRIGGER_FAILURE;
ELSE
   null;
END IF;

--
-- PROCESS all reqs (approved or new): Update the PR and IM tables if the value or unit price of the PRD changed
--
--
-- update pr table

IF (:prd_table.calcvalue != :prd_table.calcvalue_hold) THEN

   UPDATE pr_table
   SET value = value + (:prd_table.calcvalue - :prd_table.calcvalue_hold)
   WHERE req_nbr = RPAD(:prd_table.req_nbr,10,' ');

   -- update the hold variables for prd value
   :prd_table.calcvalue_hold := :prd_table.calcvalue;

END IF;

-- update the item/part masters if the unit price for this item was modified

IF (:prd_table.unit_prc != :prd_table.unit_prc_hold or :prd_table.qty != :prd_table.qty_hold) THEN

   IF (:prd_table.unit_prc != :prd_table.unit_prc_hold AND :prd_table.cont_service != 'Y' AND RTRIM(:prd_table.part) IS NOT NULL) THEN

      UPDATE im_table
      SET std_mat = :prd_table.unit_prc,
         std_tot = :prd_table.unit_prc
      WHERE part = RPAD(:prd_table.part,32,' ');

      UPDATE pid_table
      SET cur_mat = :prd_table.unit_prc,
         cur_tot = :prd_table.unit_prc
      WHERE part = RPAD(:prd_table.part,32,' ');

   END IF;

   -- update the hold variables for the line item amount
   :prd_table.qty_hold := :prd_table.qty;
   :prd_table.unit_prc_hold := :prd_table.unit_prc;

END IF;

END;

Listing 11: PRAV PRD_Table Pre-Update Trigger