Implementing a Basic POS Solution within Oracle Applications

Ugur Evin
Cayman Islands Government

Introduction

How to implement a basic POS solution within Oracle Applications? This paper explains how to combine transaction entry, receipt entry and receipt printing processes in a single process based on a real world implementation. As part of this implementation you will see how to implement "bitmap printing" without using standard postscript printing method, create receipts on-the-fly using AR interface and use custom library to keep your customizations portable across different versions of Oracle Applications.

Company

This solution has been implemented in the Cayman Islands Government. The Cayman Islands comprises Grand Cayman, Cayman Brac and Little Cayman. The three islands are situated in the western Caribbean, about 150 miles south of Cuba, 480 miles south of Miami, Florida, and 180 miles northwest of Jamaica. George Town, the capital, is on the western shore of Grand Cayman. The Cayman Islands is a British Dependent Overseas Territory. It is a parliamentary democracy with judicial, executive and legislative branches. There is also a governor appointed from the UK over the Executive Council.

The government has portfolios and ministries. There are departments under ministries. This implementation covers all the departments in the government.

A Typical POS System

POS stands for Point Of Sale. This is a rather broad definition that can include merchandising aids, displays and the methods used to enable transactions. A PC point-of-sale system designed around a standard personal computer. Customers can add various peripherals to the system to meet their own particular needs. Basically a POS system consists of a receipt printer, a reader (check reader, magnetic stripe reader), keyboard, barcode scanner, applications software, etc. The main goal of a POS system is to complete the whole transaction in an acceptable amount of time.

Background

Cayman Islands Government (CIG) provides public services to people and businesses in the Cayman Islands. Customers come to government offices to pay their bills in return to services CIG provides them. In most cases CIG doesn't send any invoice or notification to people to pay their bills since payment amounts and schedules are not predictable. In other words in this implementation CIG doesn't create invoice transactions in the Oracle Applications before customers come to pay their bills. Payment types can be taxes, hotel licensing fees, vehicle licensing fees, driving license renewal fee, etc.

Basically, after a customer makes his payment, CIG needs to issue a receipt back to the customer. As we all know Oracle Applications software has not been designed as a POS system. In our case, in order to issue a receipt back to customer, transaction information must be entered using AR Transaction form (*arxtwmai.fmb*) since it has not been entered before, then completed, and finally a cash receipt must be entered using AR Receipt form (*arxrwmai.fmb*) as a separate process. After all, you need to print out a physical receipt, normally a character based receipt which doesn't work for us, back to customer. Waiting for all this work finishing is a nightmare for both customer and CIG officer.

The whole purpose of our implementation was to issue a bitmap receipt (CIG crest on it) to customer as quickly as possible. However there were some difficulties preventing us from finding a solution easily.

First of all, in order to print a bitmap report within Oracle Applications you need to have postscript or postscript emulating printers. Not all the CIG printers have been postscript or postscript printing capable. So, how would we print bitmap receipts in this environment?

Suggested Solutions and Drawbacks to Print Bitmap Reports

The first solution was to use a pre-printed paper with CIG crest on it. However, this solution requires you to change the paper whenever you need to print something else. They tried to use this for a while but didn't survive long since it wasn't practical at all.

The second solution was to buy a postscript or postscript-emulating printer for each department that needs to issue a bitmap receipt. This solution was not accepted since the cost of the solution was too high compared to the return of investment.

The final solution was to write down receipts manually by hand. They tried doing this for a while but it was not an accepted solution either since the whole process would need to finish in a decent amount of time.

How to Shorten the Whole Process

Other than the problem of printing a bitmap report, we haven't had decided yet how to shorten transaction entry, receipt entry and bitmap printing processes to an acceptable amount of time and effort.

I came up with a technical solution by the help of functional consultants and users. I combined these 3 processes into a single process from the end-user perspective.

Basically we have had two basic problems to be solved:

1. How do we create a receipt in the background?
2. How do we print bitmap reports without using postscript printers?

1. How to Create an AR Receipt in the Background

By resolving this issue, we will be answering all the questions below:

a. Will we create a new form or customize the existing *AR Transaction form*?
b. When will we create a receipt?
c. What is the *Zoom* button and what involves when pressing the *Zoom* button?
d. How will we create a receipt record in the database?
e. What customizations need to be done?
f. What setups need to be done to use *Process Lockbox (arlplb)* program?
g. What is going on behind the scene when we click on *Print Receipt* button?

a. A New Form or Customization of an AR Transaction Form

Some people suggested me to create a new form rather than using the existing AR Transaction form for printing *Quick Receipts*, which are the AR receipts created through the Transaction form, since they needed some customizations on the form. The advantage of this approach is that you have the total control of everything in the form and it could be easy to manage the form's behavior. However, there are some drawbacks of this approach as well. Firstly, you would need to reflect the changes in the database to your form whenever an upgrade occurs by installing a patch. Other than the database changes, functionality may change as well. Secondly, it isn't practical to create a new form for this purpose since customization of this form is much easier.

The easiest approach to the customization problem is to change the form behavior and appearance by modifying the existing code and GUI items on the form. The problem with that is that whenever you apply an AR family pack or a mega patch it is most likely that your customization will be overwritten by the new patch, and then you need to do

your whole customization all over again. It is not a practical solution especially in a patchy 11i environment. In addition, this solution is not supported by Oracle.

The right solution to the problem is to use *custom libraries*. Custom library (*custom.pll*) comes with Oracle applications and it resides in *$AU_TOP/resource* directory on your application server. You can put all your custom code into the *custom.pll* . In our case, I made a small change in the *custom.pll* and attached a new library, *xxcig_quick_receipts.pll* in order to keep my customization regarding Quick Receipts separate. It is a good practice to attach your own libraries to the *custom.pll* rather than making all your changes in the *custom.pll*. Using custom libraries is easy and supported way by Oracle. However, be aware that Oracle doesn't support your customization. You can always turn on and off the custom code within the applications if you have used custom libraries, then you can inform Oracle if there is a problem with their original form when the customization is turned off.

b. When to Create a Quick Receipt

In the section above, I decided to customize *arxtwmai.fmb* form by using *custom.pll* and *xxcig_quick_receipts.pll.* Now we need to decide when we should create a quick receipt.

The first suggestion was that I should create the receipt right after a user enters the transaction. In order to do that I could use the custom library or a database trigger. However, I didn't want to do this since users may not want to create a receipt right after entering a transaction at all times. There should be an indicator that tells the system this transaction should produce a Quick Receipt or this is a regular invoice transaction that is not supposed to create a receipt. I will explain in the following section if you want to create a Quick Receipt every time you enter a transaction for your interest.

I suggested using a button to kick off the receipt creation process. Since we are not able to change the forms by adding buttons in order to be supported by Oracle, we've had to use the custom library to accomplish this. On the *toolbar* of the Oracle Applications, there is a special button called *Zoom* button. You can enable or disable this button and run your custom code when pressing the button by using the custom library. You will see how I used the *Zoom* button in the following sections.

Figure 1. Oracle Applications Toolbar

In order to enable or disable the *Zoom* button the following custom library function *zoom_available* used along with our custom library *xxcig_quick_receipts.pll*:

```
…
…
FUNCTION zoom_available RETURN BOOLEAN IS

BEGIN
   IF xxcig_quick_receipts.zoom_available THEN
     RETURN true;
   ELSE
     RETURN false;
   END IF;

END zoom_available;
…
…
```

Source 1. Customization on zoom_available function in custom.pll


```
…
…
PACKAGE BODY xxcig_quick_receipts IS
--
--  04/24/2002  Created by Ugur Evin
--

FUNCTION zoom_available RETURN BOOLEAN IS

-- Use this procedure to enable/disable the zoom button

  v_block_name  VARCHAR2(30) := NAME_IN('SYSTEM.CURSOR_BLOCK');
  v_form_name    VARCHAR2(30) := NAME_IN('SYSTEM.CURRENT_FORM');

BEGIN

   IF (v_form_name = 'ARXTWMAI' and v_block_name = 'TGW_HEADER') OR  (v_form_name = 'ARXRWMAI') THEN
      RETURN true; -- zoom button is enabled
   END IF;

   RETURN false; -- zoom button is disabled

END;
…
…
```

Source 2. From our custom library xxcig_quick_receipts.pll

c. What Involves When Pressing the Zoom Button

Well, now we are on *arxtwmai.fmb* form and enabled the *Zoom* button as above. We can put our code into the custom library to manage the form as we wish. I want the *Zoom* button to invoke our custom form, *xxcig_transaction.fmb*, which lets you print either a receipt or an invoice and create a customer. We invoke and pass the parameters to the form within *xxcig_quick_receipts.pll* as below:

```
…
…
PROCEDURE event(event_name  VARCHAR2) IS
…
…
IF event_name = 'ZOOM' and v_form_name='ARXTWMAI' THEN

     param_to_pass1 := FND_PROFILE.VALUE('Printer'); -- get the default printer name for the user
     param_to_pass2 := NAME_IN('tgw_header.customer_trx_id'); -- get the transaction id
     param_to_pass3 := NAME_IN ('tgw_header.complete_flag'); -- transaction complete (Y or N)
     param_to_pass4 := NAME_IN ('tgw_header.primary_salesrep_id'); -- salesperson
          --
     FND_FUNCTION.EXECUTE( FUNCTION_NAME => 'XXCIG_TRANSACTION', -- our custom form to be opened
                                    OPEN_FLAG => 'Y',
                                    SESSION_FLAG => 'Y',
                                    OTHER_PARAMS => 'PRINTER='''||param_to_pass1||
                                    ''TRX_ID='''||param_to_pass2||
                                    ''COMPLETE_FLAG='''||param_to_pass3||
                                    ''REP_ID='''||param_to_pass4||
                                    ''P_ORG_ID='''||v_org_id||''''); -- multiorg environment
 END IF;
…
…
```

Source 3. Handling the Zoom event in xxcig_quick_receipts.pll

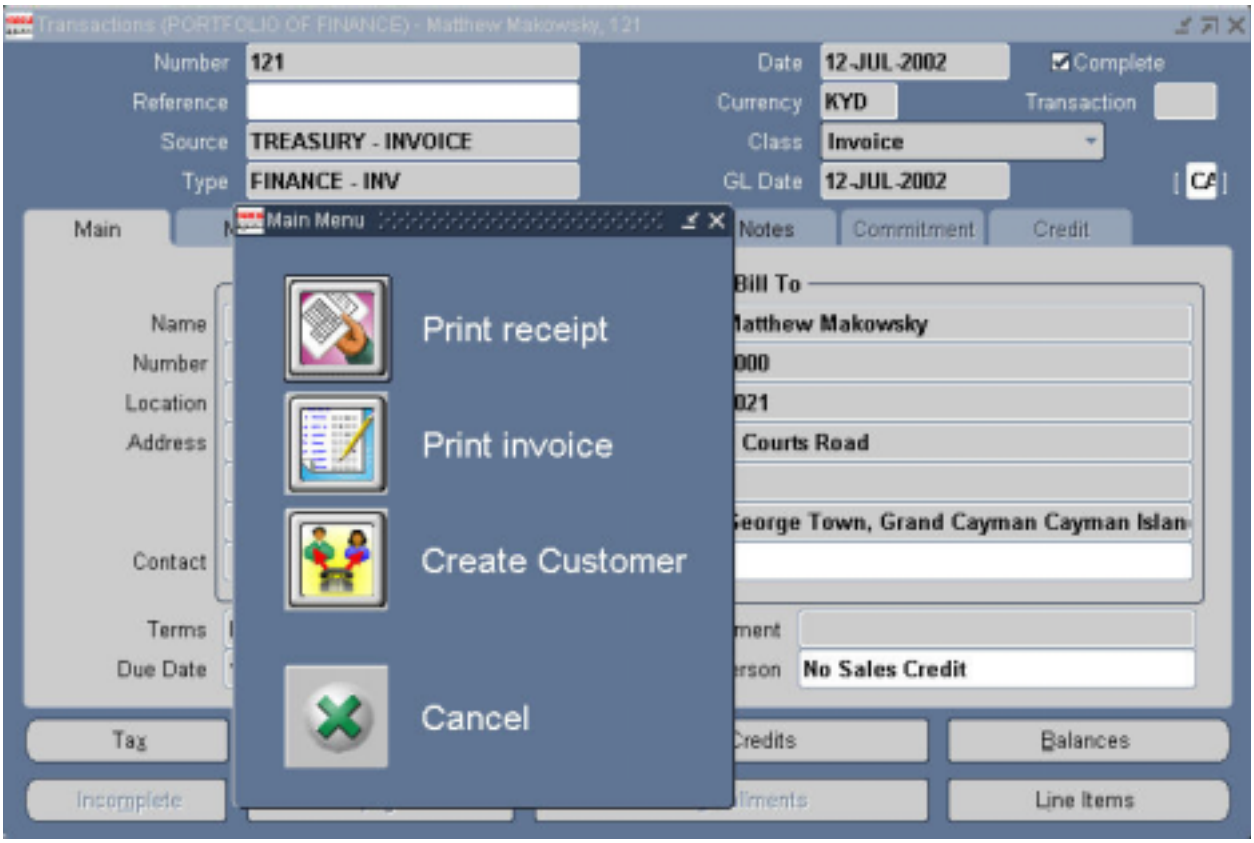

Figure 2. Custom form xxcig_transaction.fmb to print receipt/invoice and create customer

d. How to Create a Quick Receipt Record in the Database

Now we know that somehow after pressing the *Zoom* button and invoking our custom form we will create a Quick Receipt record in the database. The first method is to insert the record directly into the *ar_cash_receipts_all* table. This table stores all the receipt records. However, inserting a record directly into Oracle Applications tables is not supported by Oracle. The problem with that is that you can corrupt the database consistency since you are skipping the functional validation of a receipt record. In addition, in the AR Receipt form (*arxrwmai.fmb*) we don't know exactly what Oracle developers has done to insert a receipt record. You can open up the form and investigate further but there is an easier solution for that.

In a scenario like this Oracle recommends you to use Oracle Applications APIs (Application Program Interface). Using APIs is a supported way to enter and modify data. However, I cannot say that they are very well documented. It is not always easy to use them. In our case, I used standard AR interface program which is *Process Lockbox (arlplb)* to create a receipt record in the database. In the following sections you will see how I use *arlplb* to create a receipt.

e. Setup to be Done to Use Process Lockbox Program

1. Transmission formats specify how data in your lockbox bank file is organized so it can be successfully imported into the receivables interface tables. Receivables provides several standard transmission formats you can modify to meet your needs. In our case we create our own transmission format before using *arlplb* program. Using AR Manager responsibility you can navigate to Transmission Formats form (*arxdlktf.fmb*) through Setup > Receipts > Lockboxes > Transmission Formats from the main menu. Below is our custom transmission format, Cayman Department, and its attributes:

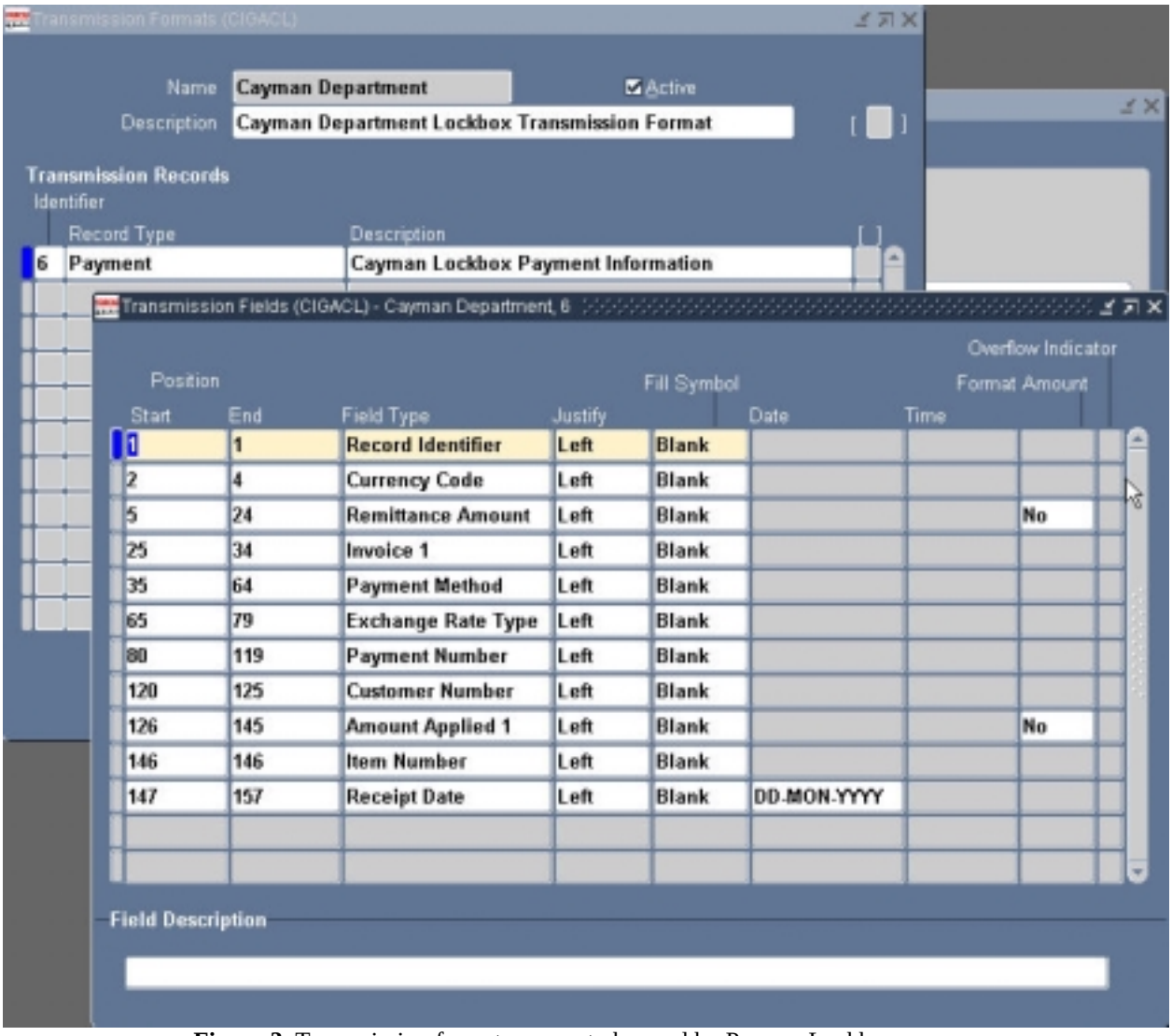

Figure 3. Transmission formats screen to be used by Process Lockbox process

2. Then we should create a text file which has the receipt information we want to create. This text file is one of the parameters of *arlplb* program. a*rlplb* loads the text file information using SQL*Loader into the *ar_payments_interface* table, finally the required table(s) is populated with the final receipt information. I have used the *utl_file* database package to create a text file. The directory in which you create the text file must be defined in the *utl_file_dir* initialization parameter in your database instance's *initSID.ora* file (it is *spfile* on Oracle 9i) before using the *utl_file* package. Let's say that if you want to store the text file under /u04/appl/ciprappl/xxcig/11.5.0/ar/lockbox directory then you should change your *utl_file_dir* parameter as follows:

```
…
utl_file_dir = /u04/appl/ciprappl/xxcig/11.5.0/ar/lockbox, /usr/tmp
…
```

Source 4. init.ora file

Please note that you separate multiple entries in the *utl_file_dir* parameter by a comma as shown above. In the following sections I will show how to create the text file.

3. If you define a different transmission format or edit the existing Default or Convert formats, you must edit the SQL*Loader control file before you can import data into Receivables. We create our control file to load text file data into *ar_payments_interface* by *arlplb* program. The format of the control file is the same as the Transmission Format we just defined above. I named the control file *caybox.ctl* and it is located under *$AR_TOP/bin* directory by default. Other control files for *arlplb* program reside in here too.

```
LOAD DATA
APPEND
INTO TABLE ar_payments_interface
WHEN RECORD_TYPE = '6' -- payment
(status                    CONSTANT 'AR_PLB_NEW_RECORD',
 record_type               POSITION(01:01)  CHAR,
 currency_code             POSITION(02:04)  CHAR,
 remittance_amount         POSITION(05:24)  DECIMAL EXTERNAL,
 invoice1                  POSITION(25:34) CHAR,
 receipt_method            POSITION(35:64) CHAR,
 exchange_rate_type        POSITION(65:79) CHAR,
 check_number              POSITION(80:119) CHAR,
 customer_number           POSITION(120:125) CHAR,
 amount_applied1           POSITION(126:145) DECIMAL EXTERNAL,
 item_number               POSITION(146:146) INTEGER EXTERNAL,
 receipt_date              POSITION(147:157) DATE 'DD-MON-YYYY')
```

Source 5. caybox.ctl control file to be used by Process Lockbox

f. What Customizations to be Done on the AR Transaction Form

1. I created a descriptive flexfield (*dff*) in order to enter payment type and other payment related information. Valid payment types can be cash, credit card, cheque, wire and none (default). If the payment type is *none*, it means that it is a basic transaction that doesn't create any quick receipt. When you navigate into the *dff* the following screen comes up:

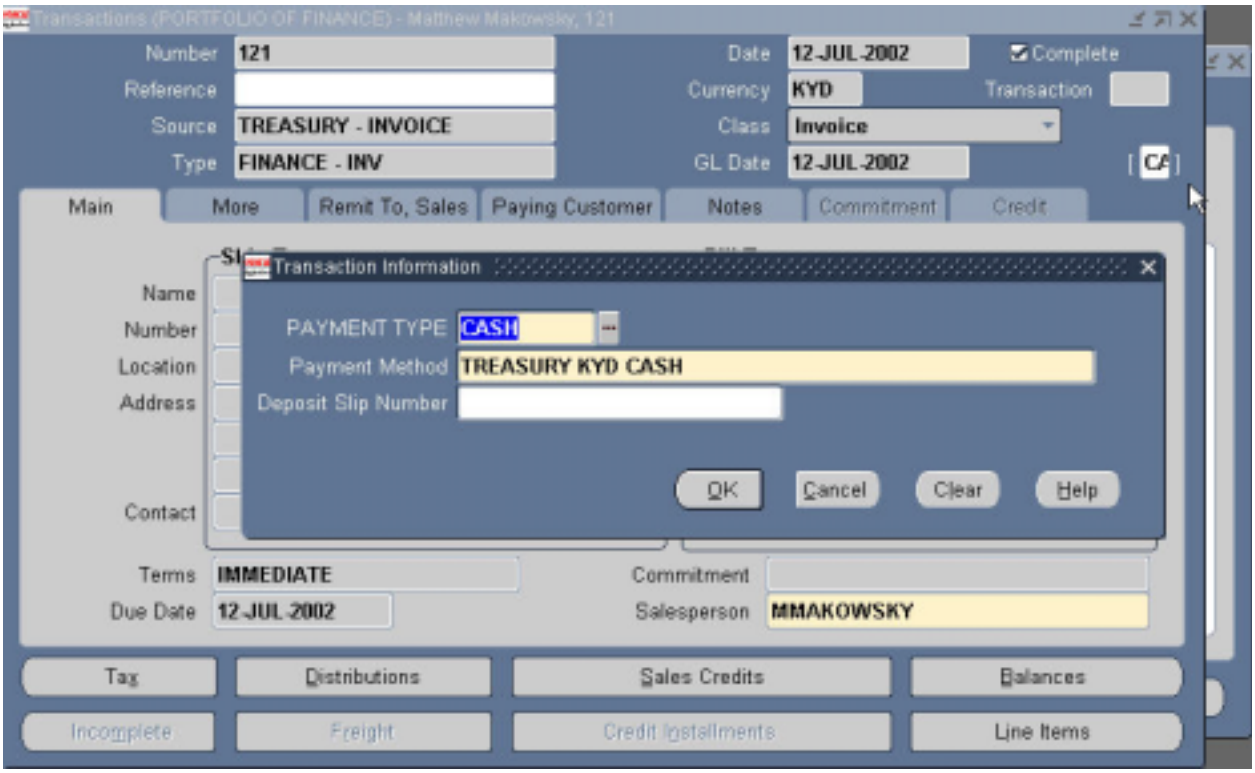

Figure 4. DFF window for entering Payment Type information

2. *Salesperson* field on the Transaction form has been defined as mandatory and is validated against Applications User Id. Normally in the Sales Person setup form there are Source Name and User Name fields. You can relate these two fields with each other. However, this form (*jtfrsdef.fmb*) has a bug (11.5.5 AR Patch set D) and you are not able to choose User Name field from the LOV. I needed to upgrade AR patch level to a later version and apply some other patches in order to fix this simple problem. However, I haven't had time to apply and test these patches. Then, I created a custom mapping table and related Salesperson names with User Names.

Here is the code in the *xxcig_quick_receipts.pll* to accomplish this:

```
…
…
IF (event_name IN ('PRE-INSERT', 'PRE-UPDATE')) AND (v_form_name='ARXTWMAI') THEN

  IF NAME_IN('tgw_header.ras_primary_salesrep_name_mir') = 'No Credit'  -- 11.5.3
     OR
     NAME_IN('tgw_header.ras_primary_salesrep_name_mir') = 'No Sales Credit'  -- 11.5.5
     OR
     NAME_IN('tgw_header.ras_primary_salesrep_name_mir') IS NULL
  THEN

     BEGIN
       SELECT
          xx.salesagentname,
          ra.salesrep_id
       INTO
          param_to_pass1,
          v_salesrep_id
       FROM
          xxcig.xxcig_user_agent xx, -- custom mapping table
          ra_salesreps_all ra,
          fnd_user fu
       WHERE
          fu.user_id = v_user_id
       AND
          xx.salesagentname = ra.name
       AND
          ra.org_id = v_org_id
       AND
          fu.user_name = xx.username;

       COPY(param_to_pass1, 'tgw_header.ras_primary_salesrep_name_mir');
       COPY(v_salesrep_id, 'tgw_header.primary_salesrep_id');

     EXCEPTION

      WHEN NO_DATA_FOUND THEN

        NULL;

     END;

  END IF;
…
…
```

Source 6. Populating Salesperson field based on Apps Userid

3. End users wanted me to populate *Remit To Address* field by the *Source* field on the *arxtwmai.fmb*. There is a one-to-one relationship between these two fields and therefore users didn't want to enter information manually.

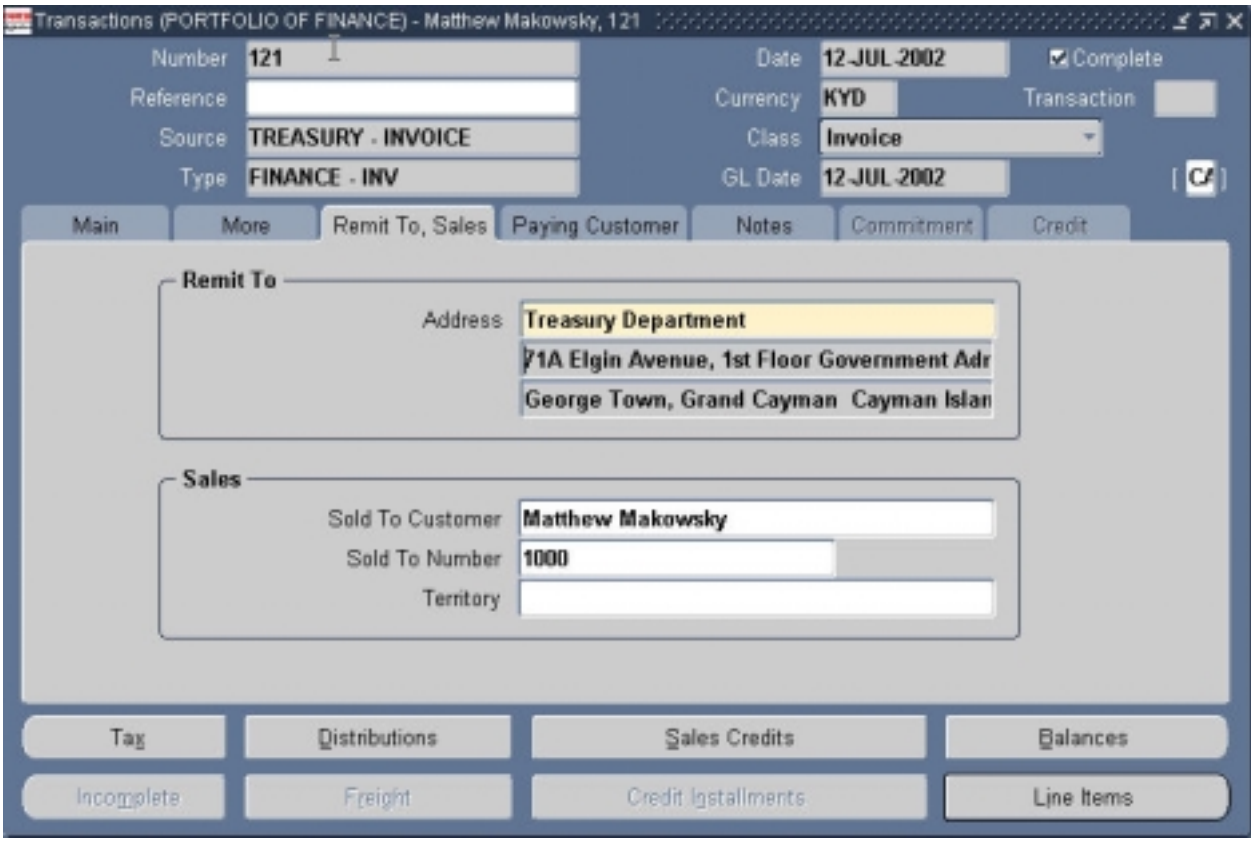

Figure 5. Populating Address field based on Source field

Here is the code in the *xxcig_quick_receipts.pll* to accomplish this:

```
…
IF event_name = 'WHEN-NEW-ITEM-INSTANCE' AND v_form_name='ARXTWMAI'
   AND v_item_name = 'raa_remit_to_address1_mir'
THEN
  BEGIN
     v_source := UPPER(NAME_IN('tgw_header.bs_batch_source_name_mir'));
     SELECT
        r.address1
     INTO
        v_address
     FROM
        ar_remit_to_addresses_v r
      WHERE
        UPPER(r.address1) LIKE RTRIM(LTRIM(SUBSTR(v_source,1, INSTR(v_source,'-')-2)))||'%';

     COPY(v_address, 'tgw_header.raa_remit_to_address1_mir');
     NEXT_ITEM;

  EXCEPTION
  WHEN NO_DATA_FOUND THEN
     NULL;
  WHEN TOO_MANY_ROWS THEN
     NULL;

  END;

END IF;
```

Source 7. Populating Address field based on Source field

4. Users also wanted me to validate the *Description* field on the Transactions Lines form when user leaves the item. By default, even if the information you enter doesn't exist in the LOV defined for the field you can enter it. We needed to change this behavior to prevent us from entering information different from the ones defined in the LOV.

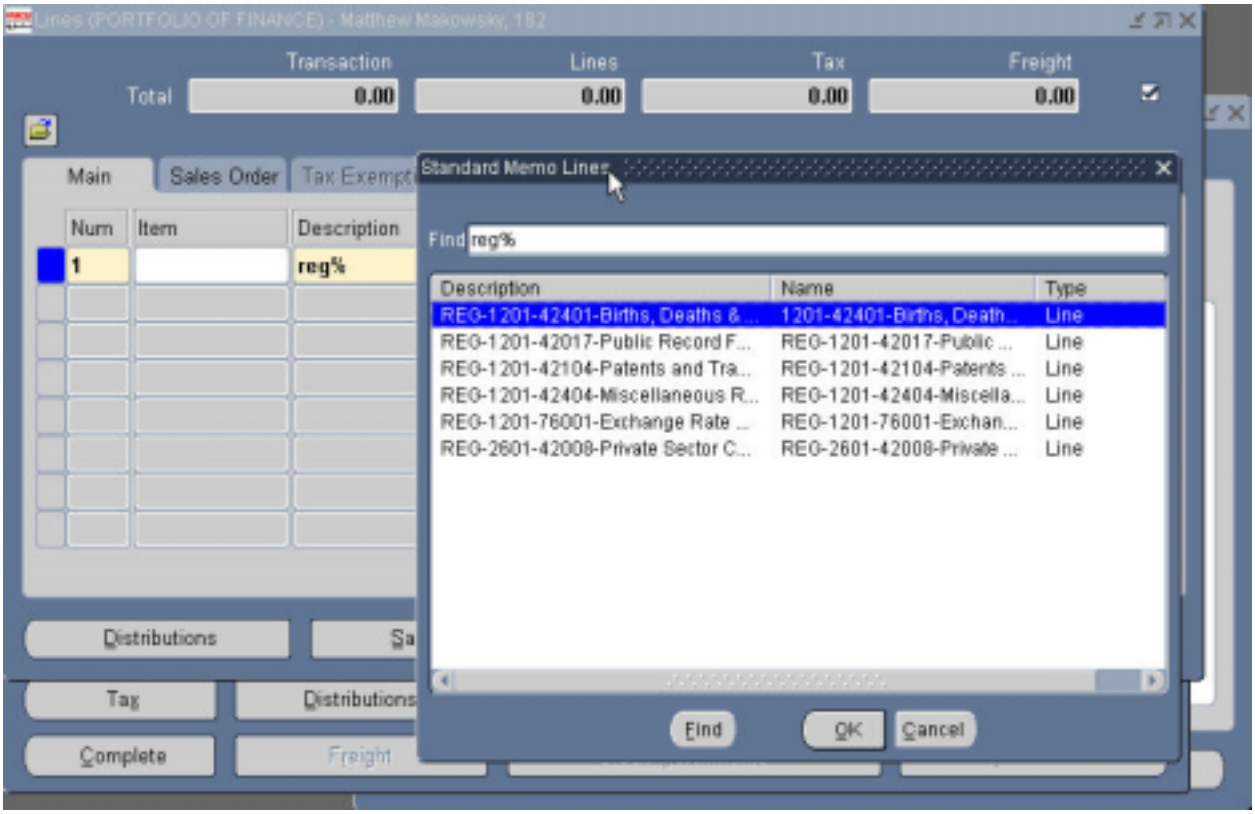

Figure 6. Validating Description field by using an LOV

The reason we haven't used the *Item* field is that this information comes from the *Inventory* items. In order to create an inventory item you need to define a lot of attributes which we don't need along with an item. You have an alternative to that by using the *Description* field as in our example. This field is populated by the Memo Items.

Here is the code in the *xxcig_quick_receipts.pll* to accomplish this:

```
…
…

IF event_name='WHEN-NEW-BLOCK-INSTANCE' AND v_block_name = 'tlin_lines' THEN

  SET_ITEM_PROPERTY('tlin_lines.description', VALIDATE_FROM_LIST, PROPERTY_TRUE);

END IF;
…
…
```

Source 8. Validating Description field by using an LOV

g. What Occurs When We click on Print Receipt Button

Print Receipt button on *xxcig_transaction.fmb* form invokes a window (block) which lets you enter the number of copies you want for a receipt or invoice, and printer information. By default the printer name is the default printer for the user's personal profile if available. You see the window below to enter this information:

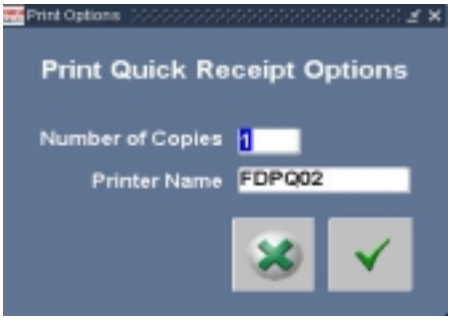

Figure 7. Printing options for bitmap reports

When you click the OK button, it calls a custom API (basically a stored procedure) to pass the required parameters to the custom bitmap listener program (*report_listener.fmb*). This program prints bitmap report requests and sends the output either as a printout or a *.pdf* file to the user. In the following section you will see more about this program.

Then there is a control mechanism by making use of a custom stored function called *f_receipt_exist* which checks whether a receipt has been created for the transaction before or not. If a receipt has been created before it exits the whole process by returning a *false* value. You see that we can reprint a receipt as many times as we want but we create the receipt only once in the system.

Here is the code in the *f_receipt_exist* function to accomplish this:

```
CREATE OR REPLACE FUNCTION f_receipt_exist(v_trx_id IN ra_customer_trx_all.customer_trx_id%TYPE)

RETURN BOOLEAN IS

    dummy NUMBER;

BEGIN

    SELECT COUNT(*) INTO dummy
    FROM xxcig.report_request_queue rrq
    WHERE rrq.parameter1_value = TO_CHAR(v_trx_id) AND
    rrq.report_name  IN  ('CAYQUICK', 'LSCAYQUICK', 'TOCAYQUICK', 'PNCAYQUICK', 'MCCAYQUICK'); -- different quick
receipt reports for different departments

   IF dummy > 1 THEN  -- receipt exists

      RETURN TRUE;

   ELSE -- receipt may exist

      SELECT COUNT(*) INTO dummy
      FROM xxcig.completed_reports cr
      WHERE cr.parameter1_value = TO_CHAR(v_trx_id) AND
      cr.report_name IN ('CAYQUICK', 'LSCAYQUICK', 'TOCAYQUICK', 'PNCAYQUICK', 'MCCAYQUICK');

      IF dummy > 1 THEN  -- receipt exists

        RETURN TRUE;

      ELSE -- receipt still may exist

        SELECT COUNT(*) INTO dummy
        FROM ar_receivable_applications_v
```



```
          WHERE customer_trx_id = v_trx_id;

          IF dummy = 0 THEN  -- receipt does not exist
             RETURN FALSE;
          ELSE
             RETURN TRUE; -- receipt exists
          END IF;

       END IF;

    END IF;

END;
```

Source 9. Checking if a receipt has been created earlier

If a receipt has not been created before, I call a custom stored procedure called *p_quick_receipts* to create the receipt record in the database. I create a text file within this procedure for the corresponding transaction record as below:

```
…
…
  SELECT TO_CHAR(xxcig.seq_filename.NEXTVAL) INTO v_filename FROM DUAL;
  v_filename := v_filename||'.dat'; -- a unique filename

  v_id := UTL_FILE.FOPEN ( v_location, v_filename, v_open_mode);

  UTL_FILE.PUT_LINE ( v_id, v_buffer );
  UTL_FILE.FCLOSE(v_id);
…
…
```

Source 10. Creation a text file for Process Lockbox

After creating the text file we are in the final stage. At this stage we submit the Process Lockbox (*arlplb*) concurrent request within our stored procedure as below:

```
…
…
  req_id_box := FND_REQUEST.SUBMIT_REQUEST (
          'AR',
          'ARLPLB',
           null,
          null,
          FALSE,
          'Y',
          '',
          '',
          'Caybox'||v_filename, -- transmission name
          'Y',
           v_location||'/'||v_filename, -- name and location of the text file we just created above
          'caybox', -- SQL*Loader controlfile name
          '1000', -- transmission format id for the transmission of Cayman Department
          'Y',
          'N',
           v_lockbox_id, -- lockbox id based on receipt method id used
           v_trx_date,
          'A',
          'N',
          'Y',
          '',chr(0),'','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','','',
                      '','','','','','','','','','','','','','','','','','','','','','','','','','','',''); -- for Form 4.5 (plsql 1.x)
…
…
```

Source 11. Submitting Process Lockbox process to create the quick receipts

fnd_request.submit_request kicks off the *arlplb* program as a concurrent program and your main program ends after issuing this command. You don't need to wait for *arlplb*'s finishing to exit your application. In other words for the

next transaction for another customer you don't need to wait the first one to finish. Kicking off the three processes (creating invoice transaction, printing a bitmap receipt, creating a receipt) in parallel from the end user's perspective speeds up the whole business process.

As I have earlier mentioned, if you prefer to create a quick receipt right after you enter a transaction you can use a database trigger. In order to test it I wrote an insert/update trigger on *ra_customer_trx_all* table when the condition UPPER(NEW.complete_flag) = 'Y' is true. The only difference between submitting a concurrent program within a PL/SQL program and a database trigger is that in the database trigger you need to put the following line before submitting the concurrent program as shown below:

```
flag := FND_SUBMIT.SET_MODE(TRUE); -- flag is BOOLEAN

req_id_box := FND_REQUEST.SUBMIT_REQUEST (
                  'AR',
                  'ARLPLB',
                   …
```

Source 12. Using a database trigger to submit a concurrent request

2. How to Print Bitmap Reports without Using Postscript Printers

As I have mentioned earlier, in order to print a bitmap report within Oracle Applications you should have postscript or postscript emulating printers. I am not going to talk about how to setup your printers within Oracle Applications to produce postscript output. In our case I haven't used this method, instead, I developed a simple program using Forms 6i to accomplish bitmap printing.

Originally I developed this solution in the year of 2000 within Oracle Applications 10.7 NCA single-org environment. Then I adapted the whole system including creating quick receipts to 11i multi-org environment. In the beginning I tried using Oracle Report Server on the Application Server node. However, even if it was working for a while, it used to crash a few times during the day. It couldn't be tolerated for sure. 10.7 NCA software uses Developer/2000. The reason for crash might have been of early versions of Oracle Report Server or the web server's itself that is Oracle Web Server 3.0.

Instead of resolving the issue I wrote a program for this purpose using Forms 6i and put it on the Application Server. It also allowed us to develop reports using Reports 6i. I have named this program RLS (report listener software). RLS is basically a listener program that picks up the report requests from a queue (table) and sends the output to the requesting user as a printout or a pdf file by an email. RLS uses a timer and checks the request queue at a specified interval for a specified manager.

When we click the Print Receipt button on our custom form it submits the report request to the queue for the RLS using our custom API (stored procedure) *p_report_queue* as shown below:

```
p_report_queue
  ( v_request_id             IN NUMBER,
    v_report_id              IN NUMBER,
    v_printer_name           IN VARCHAR2, -- name in the apps
    v_copies                 IN VARCHAR2     DEFAULT '1',
    v_orientation            IN  VARCHAR2    DEFAULT 'PORTRAIT',
    v_request_date           IN  DATE,
    v_priority               IN     NUMBER   DEFAULT 50,
    v_status                 IN    VARCHAR2  DEFAULT 'PENDING',
    v_num_of_param           IN    NUMBER    DEFAULT 0,
    v_parameter1_name        IN VARCHAR2     DEFAULT NULL,
    v_parameter1_value       IN VARCHAR2     DEFAULT NULL,
    v_parameter2_name        IN VARCHAR2     DEFAULT NULL,
    v_parameter2_value       IN VARCHAR2     DEFAULT NULL,
…
…
    v_parameter20_name  IN VARCHAR2    DEFAULT NULL,
    v_parameter20_value  IN VARCHAR2    DEFAULT NULL )
```

Source 13. Custom stored procedure to submit bitmap report requests

Then the specified concurrent manager picks up the request from the queue and converts the APPS printer name (a Unix printer queue name) to the associated NT queue name and runs the report using *rwrun60* executable by adding the parameters that have been passed to the RLS through *p_report_queue*.

I would like to summarize the main features of RLS below:

a. Any Oracle bitmap report can be printed using your available Inkjet or Laser printers
b. It doesn't integrate with Oracle Applications only. Its open architecture enables you to run your custom reports for any other application
c. You can define as many as custom concurrent managers in the RLS. You can categorize your reports into their functional purposes and define a concurrent manager to a specific group of reports. Reports run in parallel and the ones in different concurrent managers do not need to wait for other reports' finishing in different concurrent managers.

Here are some screen shots taken from RLS:

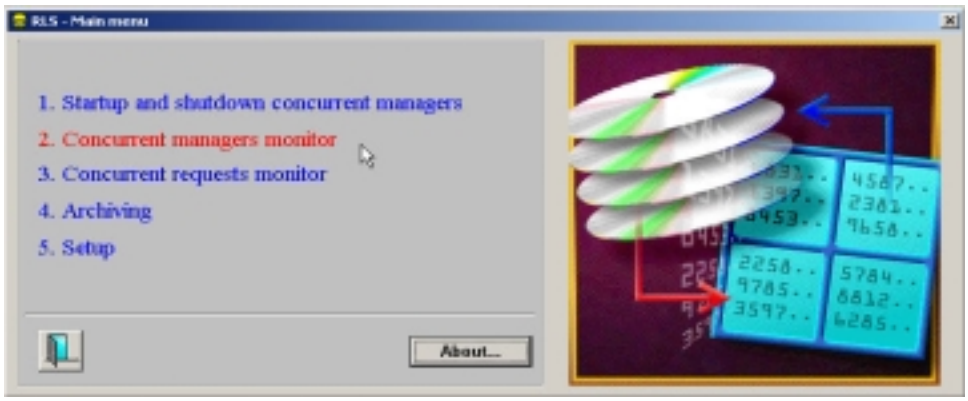

Figure 8. RLS Main Menu

Figure 9. RLS Setup

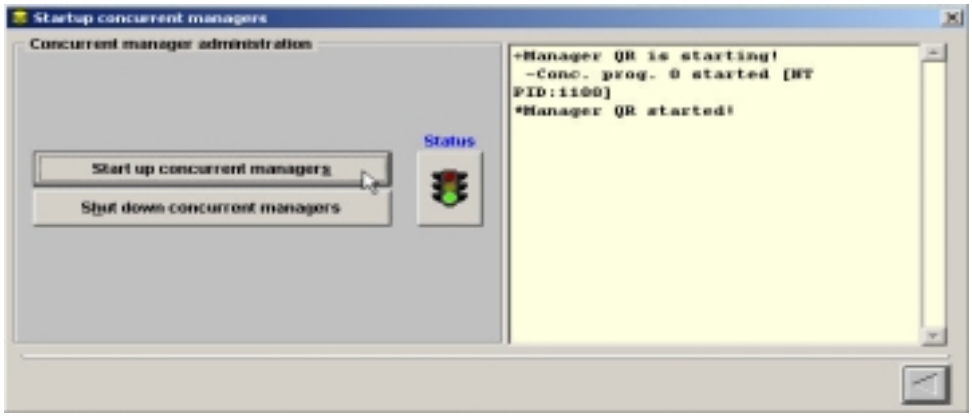

Figure 10. Starting concurrent managers

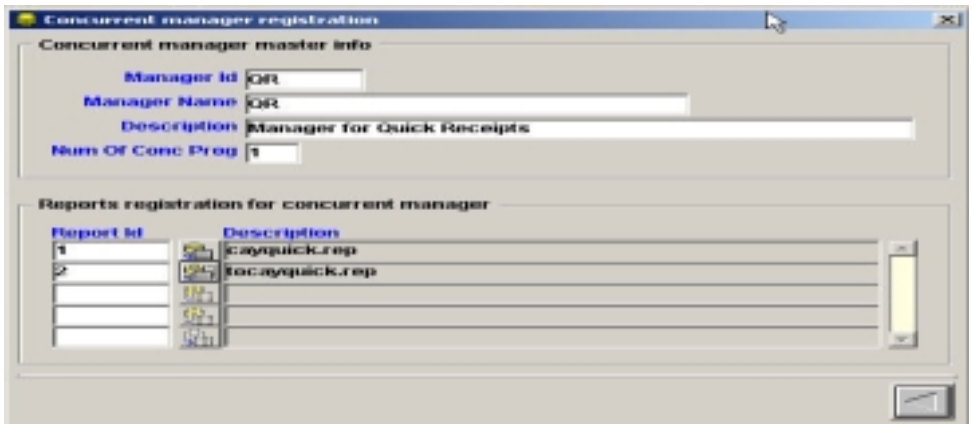

Figure 11. Defining a concurrent manager

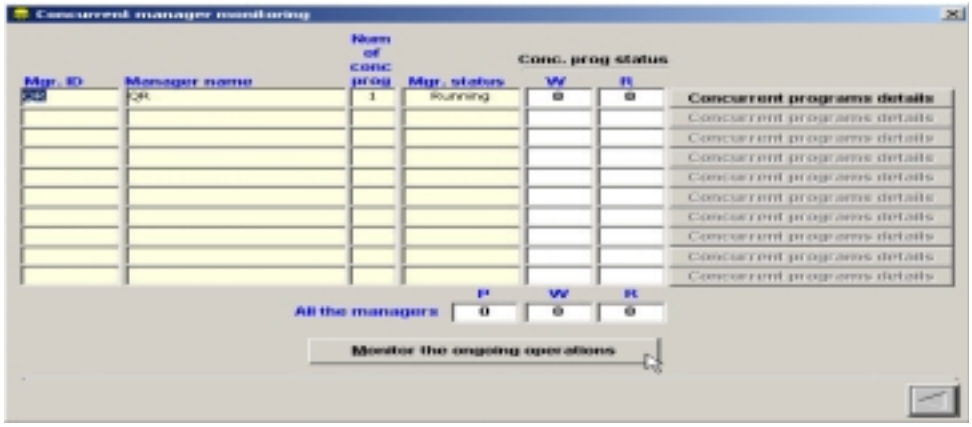

Figure 12. Monitoring concurrent managers

d. The output of the reports can be pointed to either a printer or a file. The file can be forwarded to the requestor or a specific person or group of people.

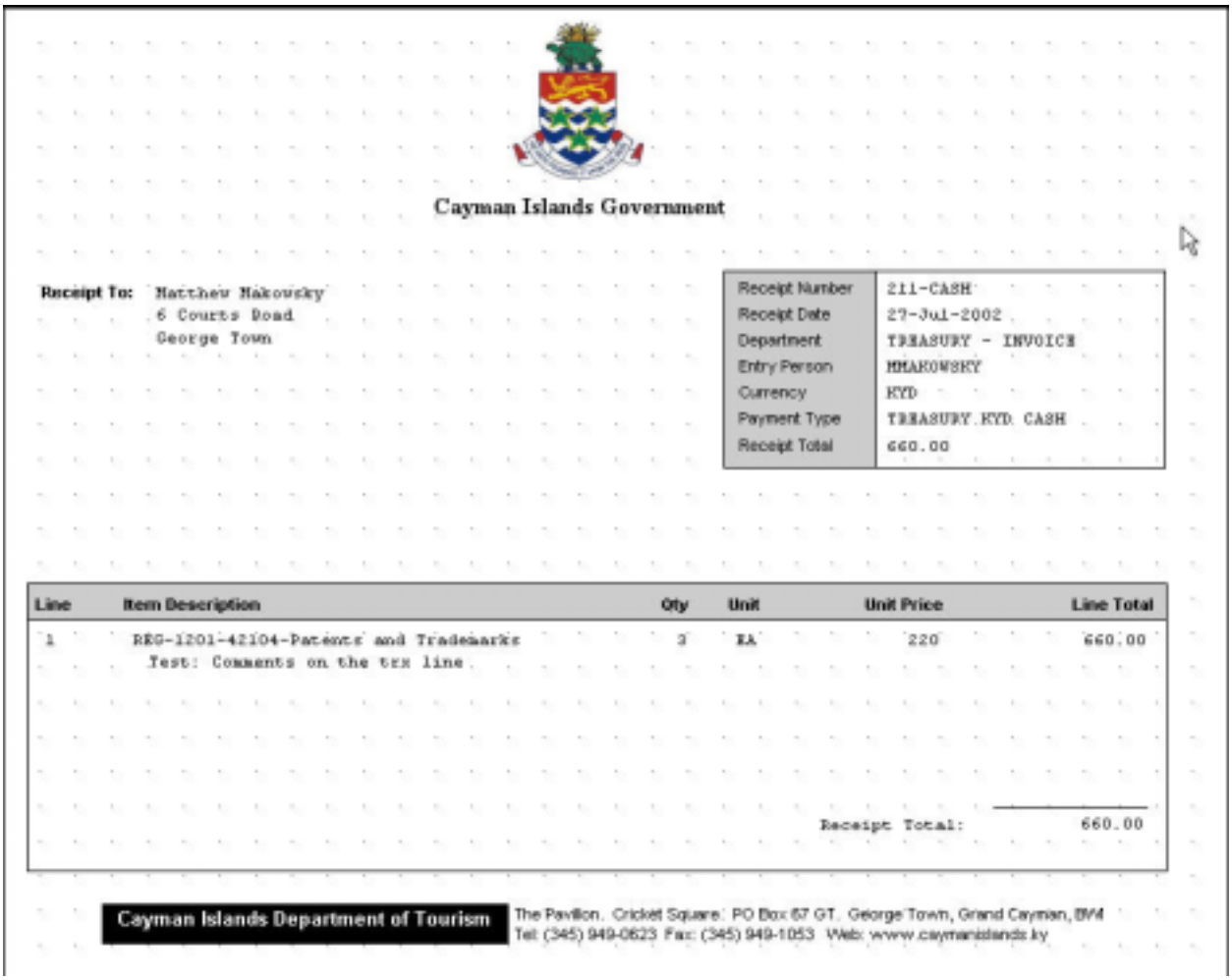

Figure 13. Quick receipt report

e. All the ongoing operations can be monitored in real time, and the processing order of the reports can be changed on the fly. You can assign priorities for each report ranging from 1 (highest) to 99 (lowest) when you register the reports for the RLS (similar to standard Oracle Applications). The reports having the same priority are processed by FIFO basis:

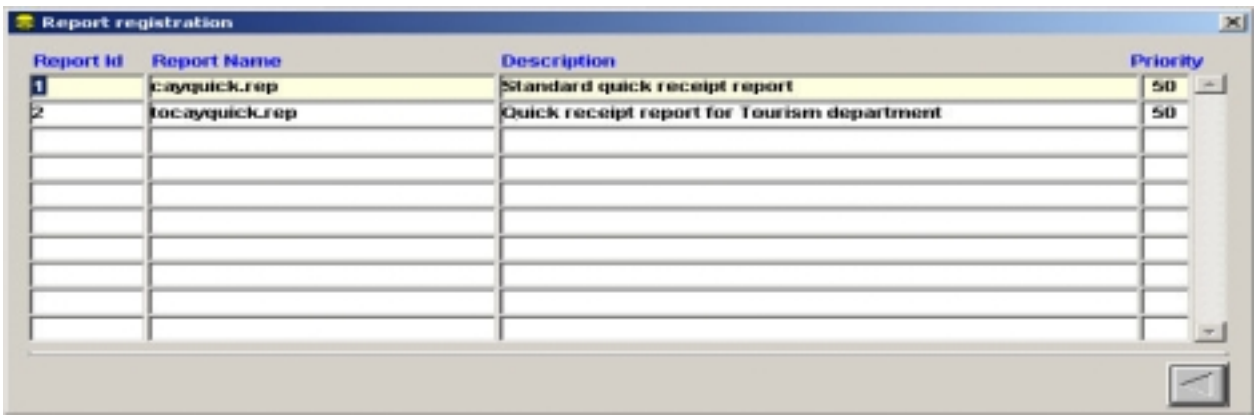

Figure 14. Report registration

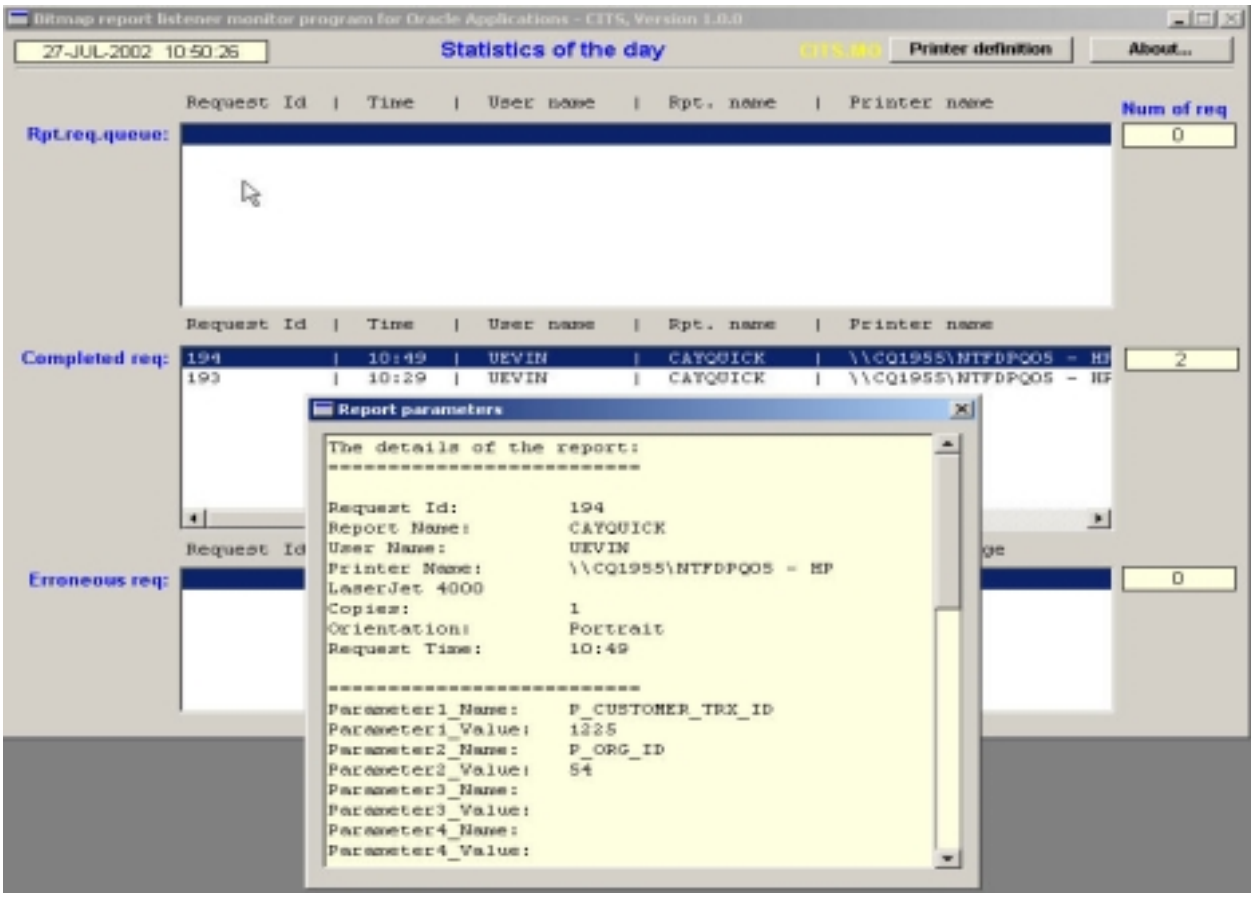

Figure 15. Monitoring report requests

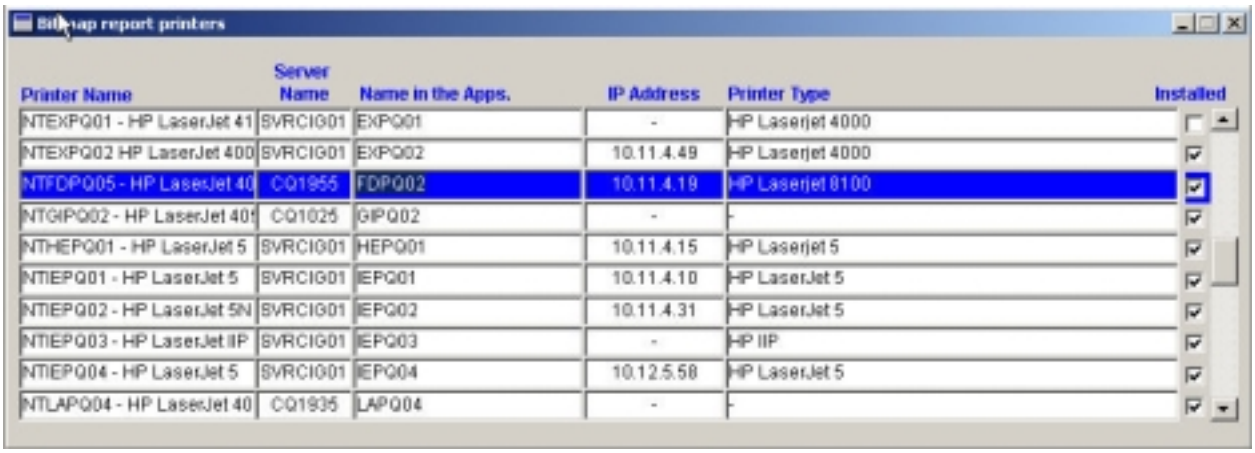

Figure 16. Defining printers in the RLS

f. The printer may not be defined in the RLS but in the Oracle Applications. In that case user gets a warning which lets him change his/her printer name. At the same time RLS sends an email to me or any specified Sysadmin/DBA type of person to define that printer in the RLS by associating it with an existing NT printer queue name.

When I first developed this program it was running on an Oracle 7.3.4 database. That time Oracle database wasn't capable of sending emails as is now. I believe from release 8i there is a database package called *utl_smtp* that allows

you to send emails. In our case, I developed a forms library using *ora_ffi* package to send email messages from a MAPI (Messaging Architecture Programming Interface) compatible client such as Outlook. MAPI is a messaging architecture that enables multiple applications to interact with multiple messaging systems across a variety of hardware platforms. MAPI is made up of a set of common application programming interfaces and a dynamic-link library (DLL) component. Sending emails using MAPI is out of this paper.

Related Customizations

a. You can reprint the same receipts that you have created using Transaction form within standard Receipts form (*arxrwmai.fmb*) as well or create a receipt and directly print it from this screen using the *Zoom* button as shown below:

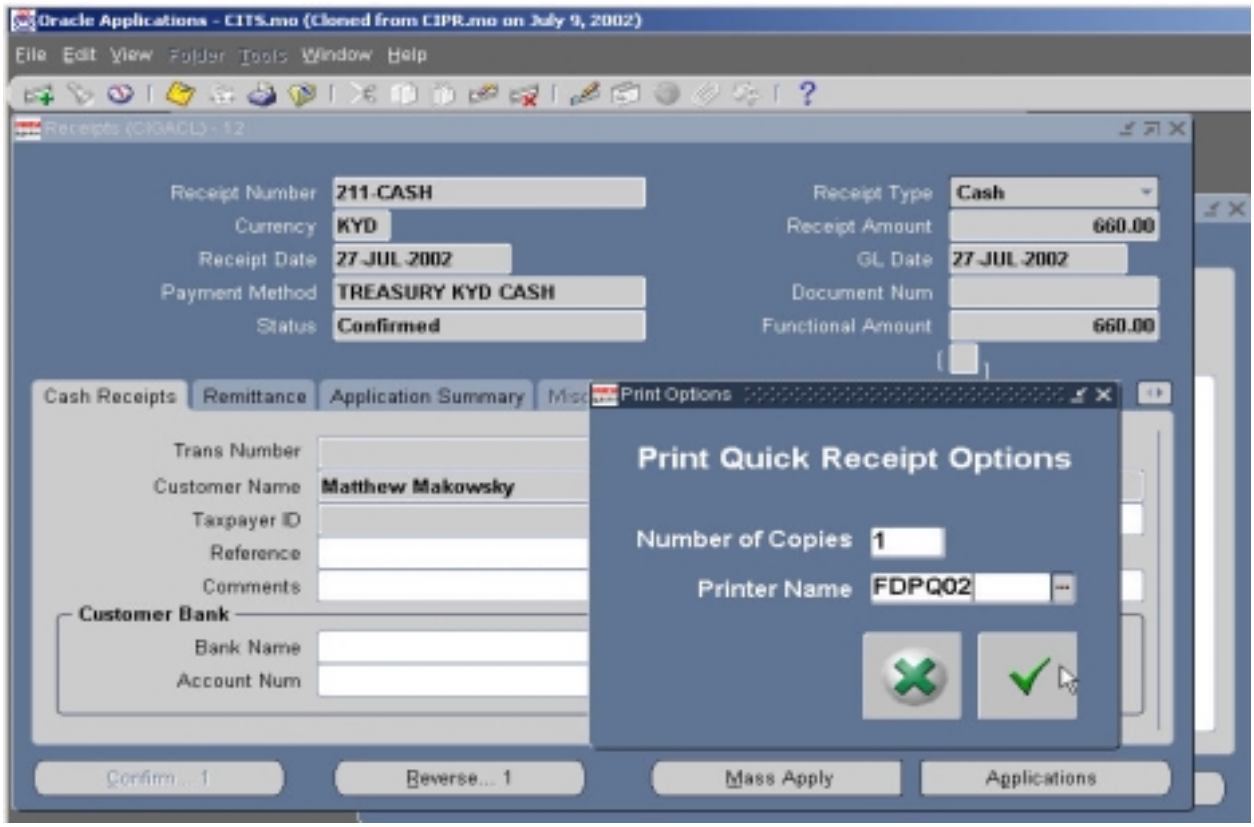

Figure 17. Printing receipts directly from Receipts form

b. We also have used RLS to print bitmap purchase orders from the Purchase Order form.

Conclusion

In this paper you have seen a series of customizations that have been used by the Cayman Islands Government for two years. Making use of these customizations I have solved our two main issues, which are creating quick receipts and printing bitmap reports, and succeeded to develop a basic POS system within Oracle Applications as a final outcome.

About the Author

Ugur Evin is a freelance consultant and currently works for the Cayman Islands Government. He has more than 7 years of post-graduate experience in project leading, DBA and application development on Oracle databases. He is an OCP DBA and OCP Application Developer. He can be reached at ugurevin@yahoo.com for further inquires regarding the paper.

