Introduction

Oracle XML/BI Publisher is a very good reporting tool, which outputs reports into many formats like Word, Excel, PowerPoint, PDF and even to Adobe Flash. You can deliver reports generated from XML/BI Publisher to many destinations like email, fax and printer using delivery manager API'S but it is not directly integrated into the e-business suite's concurrent manager so here in this paper I will discuss about how to deliver an xmlp report to email using PL/SQL.

Using PL/SQL you can write a procedure, which can be submitted as concurrent request to run any XMLP report. Following steps can be used in the procedure to deliver XMLP report using PL/SQL:

1. Submit the XMLP request using FND_REQUEST.SUBMIT_REQUEST
2. Wait for the request to complete using FND_CONCURRENT.WAIT_FOR_REQUEST
3. Email the output file using UTL_SMTP for file less than 32kb. For files greater than 32kb use PKGSENDMAIL given in this paper.

The procedure to submit & email a request is below:

```
CREATE OR REPLACE PROCEDURE spsendemail(errbuf            OUT VARCHAR2,
                                        retcode           OUT VARCHAR2,
                                        papplicationname  IN VARCHAR2,
                                        pprogramshortname IN VARCHAR2,
                                        planguage         IN VARCHAR2,
                                        pterritory        IN VARCHAR2,
                                        poutputformat     IN VARCHAR2,
                                        precipientemail   IN VARCHAR2) IS
  vdirectory    VARCHAR2(20) := 'OUTFILES';
--set this directory to $APPLCSF/out/<SID>_<hostname>
  vfile         BFILE;
  vfilelen      PLS_INTEGER;
  vmaxlinewidth PLS_INTEGER := 54;
  vbuffer       RAW(2100);
  vamt          BINARY_INTEGER := 672;
  vfilepos      PLS_INTEGER := 1; /* pointer for the file */
  vconn         UTL_SMTP.connection;
  vdata         RAW(2100);
  vchunks       PLS_INTEGER;
  vmodulo       PLS_INTEGER;
  vpieces       PLS_INTEGER;
  vmimetypebin  VARCHAR2(30) := 'application/doc';
  vrequestid    NUMBER(20);
  vflag1        BOOLEAN;
  vflag2        BOOLEAN;
  vreqphase     VARCHAR2(50);
  vreqstatus    VARCHAR2(50);
  vreqdevphase  VARCHAR2(50);
  vreqdevstatus VARCHAR2(50);
  vreqmessage   VARCHAR2(50);
BEGIN
```

```
  --Add layout to the XMLP report
  vflag1 := fnd_request.add_layout(papplicationname,
                                   pprogramshortname,
                                   planguage,
                                   pterritory,
                                   poutputformat);
  --Check if the layout has been submitted
  IF (vflag1) THEN
    fnd_file.put_line(fnd_file.LOG, 'The layout has been submitted');
  ELSE
    fnd_file.put_line(fnd_file.LOG, 'The layout has not been submitted');
  END IF;
  --submit the XMLP report request
  vrequestid := fnd_request.submit_request(application => papplicationname,
                                           program     => pprogramshortname,
                                           description => '',
                                           start_time  => '',
                                           sub_request => FALSE);
  COMMIT;
  --Check if the report has been submitted successfully
  IF vrequestid = 0 THEN
    fnd_file.put_line(fnd_file.LOG,
                      'Request Not Submitted due to "' || fnd_message.get ||
'".');
  ELSE
    fnd_file.put_line(fnd_file.LOG,
                      'The Concurrent Request Id is: ' || vrequestid);
  END IF;
  --Wait for the request to complete successfully
  vflag2 := fnd_concurrent.wait_for_request(request_id => vrequestid,
                                            /* interval   => 60,*/
                                            phase      => vreqphase,
                                            status     => vreqstatus,
                                            dev_phase  => vreqdevphase,
                                            dev_status => vreqdevstatus,
                                            MESSAGE    => vreqmessage);
  COMMIT;
  --Check for the report completion
  IF (vreqdevphase = 'COMPLETE' AND vreqdevstatus = 'NORMAL') THEN
    BEGIN
      --Begin to email the report
      BEGIN
        vconn := pkgsendmail.fnbeginmail(sender => '<sender@abc.com>',
                                         --enter email of the sender
                                         recipients => precipientemail,
                                         --enter email of the receiver
                                         subject => '<enter subject here>',
                                         --subject of message
                                         mime_type =>
pkgsendmail.vmultipartmimetype);
      END spbeginmail;

      BEGIN
        pkgsendmail.spattachtext(conn      => vconn,
                                 DATA      => '<h3>Please find the following
Attachment</h3><br><br>',
                                 mime_type => 'text/html');
      END spattachtext;

      BEGIN
        pkgsendmail.spbeginattachment(conn         => vconn,
                                      mime_type    => vmimetypebin,
                                      inline       => TRUE,
```

```
                                    filename     => pprogramshortname || '_'
||
                                                    vrequestid || '_1.' ||
                                                    poutputformat,
                                    transfer_enc => 'base64');

        BEGIN
          vfile    := BFILENAME(vdirectory,
                                pprogramshortname || '_' || vrequestid ||
                                '_1.' || poutputformat);
          vfilelen := DBMS_LOB.getlength(vfile);
          vmodulo  := MOD(vfilelen, vamt);
          vpieces  := TRUNC(vfilelen / vamt);

          IF (vmodulo <> 0) THEN
            vpieces := vpieces + 1;
          END IF;

          DBMS_LOB.fileopen(vfile, DBMS_LOB.file_readonly);
          DBMS_LOB.READ(vfile, vamt, vfilepos, vbuffer);
          vdata := NULL;

          FOR i IN 1 .. vpieces LOOP
            vfilepos := i * vamt + 1;
            vfilelen := vfilelen - vamt;
            vdata    := UTL_RAW.CONCAT(vdata, vbuffer);
            vchunks  := TRUNC(UTL_RAW.LENGTH(vdata) / vmaxlinewidth);

            IF (i <> vpieces) THEN
              vchunks := vchunks - 1;
            END IF;

            pkgsendmail.spwriteraw(conn    => vconn,
                                   MESSAGE => UTL_ENCODE.base64_encode(vdata));
            vdata := NULL;

            IF (vfilelen < vamt AND vfilelen > 0) THEN
              vamt := vfilelen;
            END IF;

            DBMS_LOB.READ(vfile, vamt, vfilepos, vbuffer);
          END LOOP;
        END;

        DBMS_LOB.fileclose(vfile);
        pkgsendmail.spendattachment(conn => vconn);
      END spbeginattachment;

      pkgsendmail.spendmail(conn => vconn);
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
        pkgsendmail.spendattachment(conn => vconn);
        DBMS_LOB.fileclose(vfile);
      WHEN OTHERS THEN
        pkgsendmail.spendattachment(conn => vconn);
        fnd_file.put_line(fnd_file.LOG, SQLCODE || '-' || SQLERRM);
        DBMS_LOB.fileclose(vfile);
    END;
  ELSE
    fnd_file.put_line(fnd_file.LOG, SQLCODE || '-' || SQLERRM);
  END IF;
END spsendemail;
```

PL/SQL Code for PKGSENDMAIL package is below:

```
CREATE OR REPLACE PACKAGE pkgSendMail IS
  vSmtpHost   VARCHAR2(256) := 'localhost.localdomain';
  vSmtpPort   PLS_INTEGER := 25;
  vSmtpDomain VARCHAR2(256) := 'localdomain.com';
  vMailerId           CONSTANT VARCHAR2(256) := 'Mailer by Oracle UTL_SMTP';
  vBoundary           CONSTANT VARCHAR2(256) := '-----
7D81B75CCC90D2974F7A1CBD';
  vFirstBoundary      CONSTANT VARCHAR2(256) := '--' || vBoundary ||
                                                utl_tcp.CRLF;
  vLastBoundary       CONSTANT VARCHAR2(256) := '--' || vBoundary || '--' ||
                                                utl_tcp.CRLF;
  vMultipartMimeType  CONSTANT VARCHAR2(256) := 'multipart/mixed; boundary="'
||
                                                vBoundary || '"';
  vMaxBase64LineWidth CONSTANT PLS_INTEGER := 76 / 4 * 3;

  FUNCTION fnBeginMail(sender     IN VARCHAR2,
                       recipients IN VARCHAR2,
                       subject    IN VARCHAR2,
                       mime_type  IN VARCHAR2 DEFAULT 'text/plain',
                       priority   IN PLS_INTEGER DEFAULT NULL)
    RETURN utl_smtp.connection;

  PROCEDURE spWriteText(conn    IN OUT NOCOPY utl_smtp.connection,
                        message IN VARCHAR2);

  PROCEDURE spWriteRaw(conn    IN OUT NOCOPY utl_smtp.connection,
                       message IN RAW);

  PROCEDURE spAttachText(conn      IN OUT NOCOPY utl_smtp.connection,
                         data      IN VARCHAR2,
                         mime_type IN VARCHAR2 DEFAULT 'text/plain',
                         inline    IN BOOLEAN DEFAULT TRUE,
                         filename  IN VARCHAR2 DEFAULT NULL,
                         last      IN BOOLEAN DEFAULT FALSE);

  PROCEDURE spBeginAttachment(conn         IN OUT NOCOPY utl_smtp.connection,
                              mime_type    IN VARCHAR2 DEFAULT 'text/plain',
                              inline       IN BOOLEAN DEFAULT TRUE,
                              filename     IN VARCHAR2 DEFAULT NULL,
                              transfer_enc IN VARCHAR2 DEFAULT NULL);

  PROCEDURE spEndAttachment(conn IN OUT NOCOPY utl_smtp.connection,
                            last IN BOOLEAN DEFAULT FALSE);

  PROCEDURE spEndMail(conn IN OUT NOCOPY utl_smtp.connection);

  FUNCTION fnBeginSession RETURN utl_smtp.connection;

  PROCEDURE spBeginMailInSession(conn       IN OUT NOCOPY utl_smtp.connection,
                                 sender     IN VARCHAR2,
                                 recipients IN VARCHAR2,
                                 subject    IN VARCHAR2,
                                 mime_type  IN VARCHAR2 DEFAULT 'text/plain',
                                 priority   IN PLS_INTEGER DEFAULT NULL);

  PROCEDURE spEndMailInSession(conn IN OUT NOCOPY utl_smtp.connection);

  PROCEDURE spEndSession(conn IN OUT NOCOPY utl_smtp.connection);
```

```
END pkgSendMail;
/

CREATE OR REPLACE PACKAGE BODY pkgSendMail IS
  FUNCTION get_address(addr_list IN OUT VARCHAR2) RETURN VARCHAR2 IS
    addr VARCHAR2(256);
    i    pls_integer;

    FUNCTION lookup_unquoted_char(str IN VARCHAR2, chrs IN VARCHAR2)
      RETURN pls_integer AS

      c            VARCHAR2(5);
      i            pls_integer;
      len          pls_integer;
      inside_quote BOOLEAN;

    BEGIN
      inside_quote := false;
      i            := 1;
      len          := length(str);
      WHILE (i <= len) LOOP
        c := substr(str, i, 1);
        IF (inside_quote) THEN
          IF (c = '"') THEN
            inside_quote := false;
          ELSIF (c = '\') THEN
            i := i + 1; -- Skip the quote character
          END IF;
        END IF;
        IF (c = '"') THEN
          inside_quote := true;
        END IF;
        IF (instr(chrs, c) >= 1) THEN
          RETURN i;
        END IF;
        i := i + 1;
      END LOOP;
      RETURN 0;
    END;

  BEGIN
    addr_list := ltrim(addr_list);
    i         := lookup_unquoted_char(addr_list, ',;');
    IF (i >= 1) THEN
      addr      := substr(addr_list, 1, i - 1);
      addr_list := substr(addr_list, i + 1);
    ELSE
      addr      := addr_list;
      addr_list := '';
    END IF;
    i := lookup_unquoted_char(addr, '<');
    IF (i >= 1) THEN
      addr := substr(addr, i + 1);
      i    := instr(addr, '>');
      IF (i >= 1) THEN
        addr := substr(addr, 1, i - 1);
      END IF;
    END IF;
    RETURN addr;
  END;

  PROCEDURE spWriteMimeHeader(conn  IN OUT NOCOPY utl_smtp.connection,
                              name  IN VARCHAR2,
```

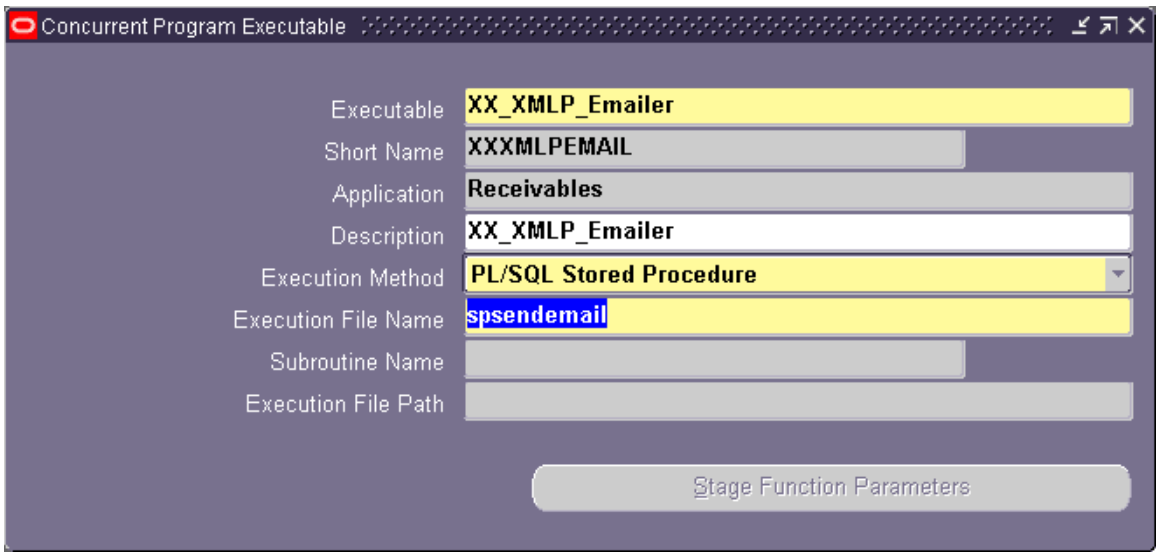
```
                              value IN VARCHAR2) IS
BEGIN
  utl_smtp.write_raw_data(conn,
                          utl_raw.cast_to_raw(name || ': ' || value ||
                                              utl_tcp.CRLF));
END;

PROCEDURE spWriteBoundary(conn IN OUT NOCOPY utl_smtp.connection,
                          last IN BOOLEAN DEFAULT FALSE) AS
BEGIN
  IF (last) THEN
    utl_smtp.write_data(conn, vLastBoundary);
  ELSE
    utl_smtp.write_data(conn, vFirstBoundary);
  END IF;
END;

FUNCTION fnBeginMail(sender     IN VARCHAR2,
                     recipients IN VARCHAR2,
                     subject    IN VARCHAR2,
                     mime_type  IN VARCHAR2 DEFAULT 'text/plain',
                     priority   IN PLS_INTEGER DEFAULT NULL)
  RETURN utl_smtp.connection IS
  conn utl_smtp.connection;
BEGIN
  conn := fnBeginSession;
  spBeginMailInSession(conn,
                       sender,
                       recipients,
                       subject,
                       mime_type,
                       priority);
  RETURN conn;
END;

PROCEDURE spWriteText(conn    IN OUT NOCOPY utl_smtp.connection,
                      message IN VARCHAR2) IS
BEGIN
  utl_smtp.write_data(conn, message);
END;

PROCEDURE spWriteRaw(conn    IN OUT NOCOPY utl_smtp.connection,
                     message IN RAW) IS
BEGIN
  utl_smtp.write_raw_data(conn, message);
END;

PROCEDURE spAttachText(conn      IN OUT NOCOPY utl_smtp.connection,
                       data      IN VARCHAR2,
                       mime_type IN VARCHAR2 DEFAULT 'text/plain',
                       inline    IN BOOLEAN DEFAULT TRUE,
                       filename  IN VARCHAR2 DEFAULT NULL,
                       last      IN BOOLEAN DEFAULT FALSE) IS
BEGIN
  spBeginAttachment(conn, mime_type, inline, filename);
  spWriteText(conn, data);
  spEndAttachment(conn, last);
END;

PROCEDURE spBeginAttachment(conn         IN OUT NOCOPY utl_smtp.connection,
                            mime_type    IN VARCHAR2 DEFAULT 'text/plain',
                            inline       IN BOOLEAN DEFAULT TRUE,
                            filename     IN VARCHAR2 DEFAULT NULL,
```

```
                              transfer_enc IN VARCHAR2 DEFAULT NULL) IS
BEGIN
  spWriteBoundary(conn);
  spWriteMimeHeader(conn, 'Content-Type', mime_type);
  IF (filename IS NOT NULL) THEN
    IF (inline) THEN
      spWriteMimeHeader(conn,
                        'Content-Disposition',
                        'inline; filename="' || filename || '"');
    ELSE
      spWriteMimeHeader(conn,
                        'Content-Disposition',
                        'attachment; filename="' || filename || '"');
    END IF;
  END IF;
  IF (transfer_enc IS NOT NULL) THEN
    spWriteMimeHeader(conn, 'Content-Transfer-Encoding', transfer_enc);
  END IF;
  utl_smtp.write_data(conn, utl_tcp.CRLF);
END;

PROCEDURE spEndAttachment(conn IN OUT NOCOPY utl_smtp.connection,
                          last IN BOOLEAN DEFAULT FALSE) IS
BEGIN
  utl_smtp.write_data(conn, utl_tcp.CRLF);
  IF (last) THEN
    spWriteBoundary(conn, last);
  END IF;
END;

PROCEDURE spEndMail(conn IN OUT NOCOPY utl_smtp.connection) IS
BEGIN
  spEndMailInSession(conn);
  spEndSession(conn);
END;

FUNCTION fnBeginSession RETURN utl_smtp.connection IS
  conn utl_smtp.connection;
BEGIN
  conn := utl_smtp.open_connection(vSmtpHost, vSmtpPort);
  utl_smtp.helo(conn, vSmtpDomain);
  RETURN conn;
END;

PROCEDURE spBeginMailInSession(conn       IN OUT NOCOPY utl_smtp.connection,
                               sender     IN VARCHAR2,
                               recipients IN VARCHAR2,
                               subject    IN VARCHAR2,
                               mime_type  IN VARCHAR2 DEFAULT 'text/plain',
                               priority   IN PLS_INTEGER DEFAULT NULL) IS
  my_recipients VARCHAR2(32767) := recipients;
  my_sender     VARCHAR2(32767) := sender;
BEGIN
  utl_smtp.mail(conn, get_address(my_sender));
  WHILE (my_recipients IS NOT NULL) LOOP
    utl_smtp.rcpt(conn, get_address(my_recipients));
  END LOOP;
  utl_smtp.open_data(conn);
  spWriteMimeHeader(conn, 'From', sender);
  spWriteMimeHeader(conn, 'To', recipients);
  spWriteMimeHeader(conn, 'Subject', subject);
  spWriteMimeHeader(conn, 'Content-Type', mime_type);
  spWriteMimeHeader(conn, 'X-Mailer', vMailerId);
```

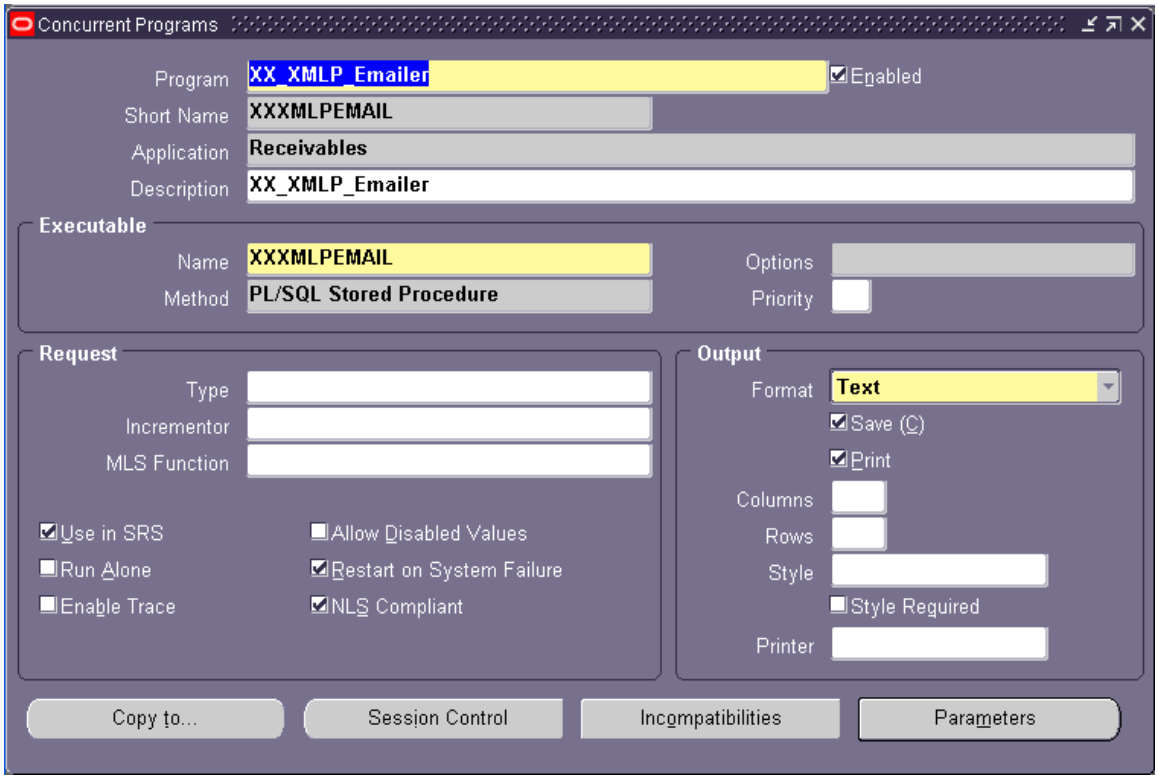
```
    IF (priority IS NOT NULL) THEN
      spWriteMimeHeader(conn, 'X-Priority', priority);
    END IF;
    utl_smtp.write_data(conn, utl_tcp.CRLF);
    IF (mime_type LIKE 'multipart/mixed%') THEN
      spWriteText(conn,
                  'This is a multi-part message in MIME format.' ||
                  utl_tcp.crlf);
    END IF;
  END;

  PROCEDURE spEndMailInSession(conn IN OUT NOCOPY utl_smtp.connection) IS
  BEGIN
    utl_smtp.close_data(conn);
  END;

  PROCEDURE spEndSession(conn IN OUT NOCOPY utl_smtp.connection) IS
  BEGIN
    utl_smtp.quit(conn);
  END;

END;
/
```

Registering the Concurrent Program:

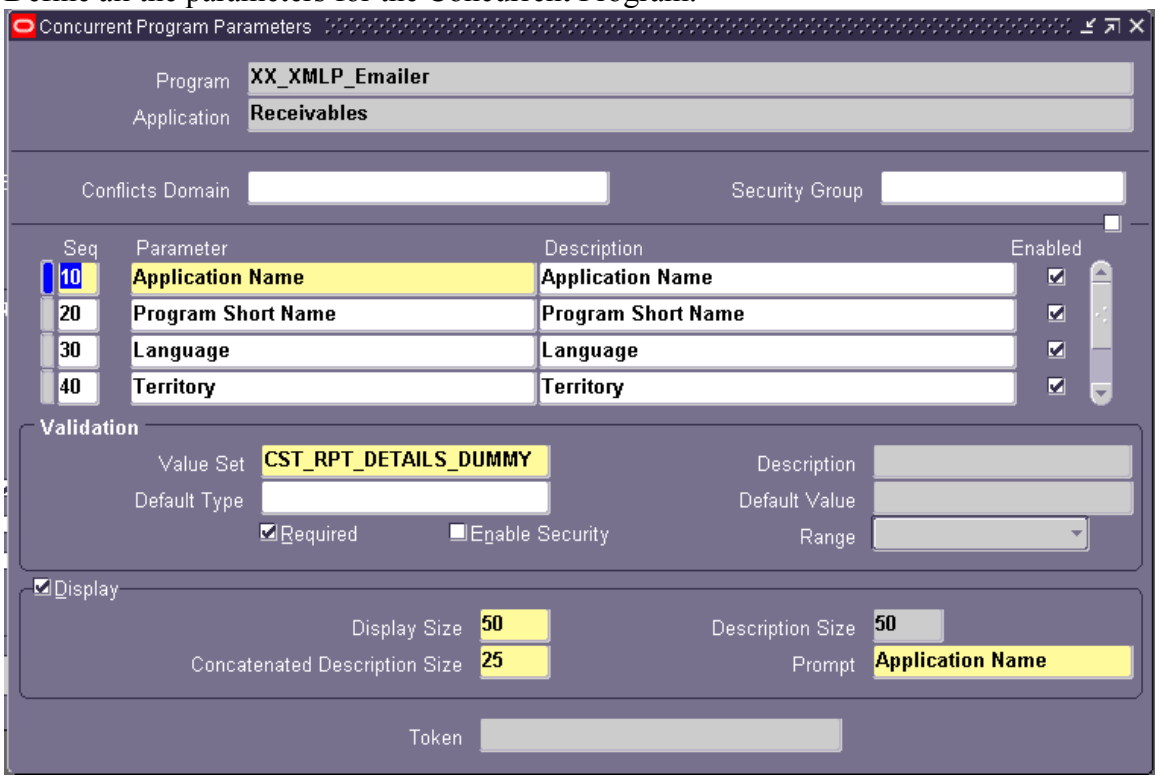
Create the Executable.



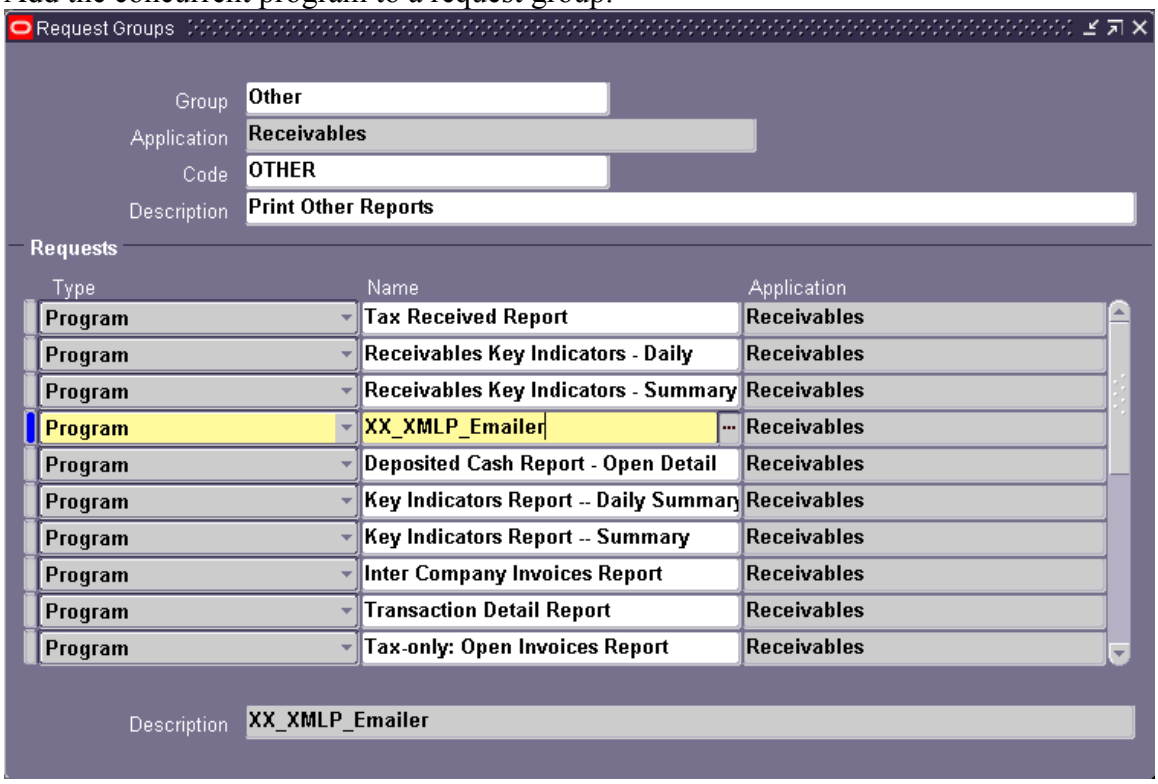
Define the Concurrent Program.



Define all the parameters for the Concurrent Program.

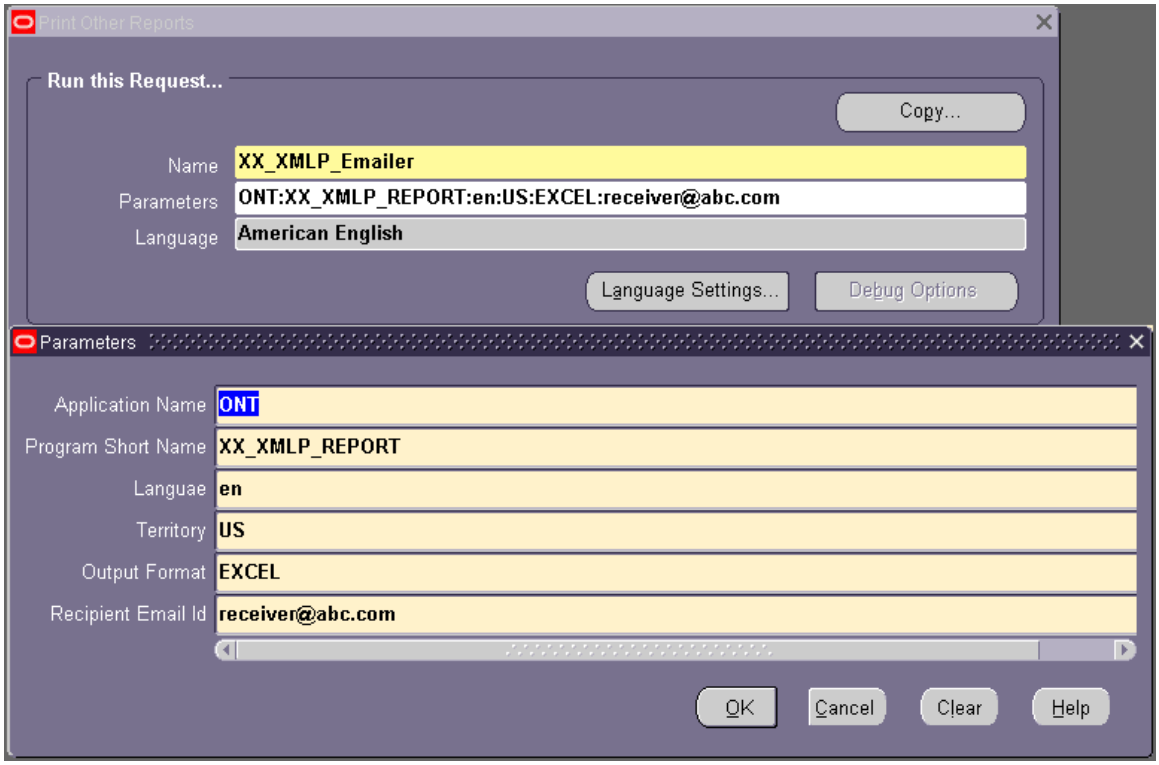


Add the concurrent program to a request group.



Submit the Concurrent request

Submit the Request and supply necessary parameters.



View the status of the request.