EBS Excel Upload Options

John Peters
JRPJR, Inc.
john.peters@jrpjr.com

Introduction
The goal of this presentation was to provide a high level survey of the various methods you can use to upload Excel data to Oracle EBS. It was not meant to drill down into any one method in too much detail, but I hope to have some simple examples for the major upload types in this paper.

Why Excel
I find Excel used in many business processes today for a variety of reasons. Data conversions when migrating a process to Oracle EBS from a legacy system. This allows users to cleanup and edit the data so that it aligns with how the data will be stored in EBS. Interfaces to external system is another common place I find Excel. Sometimes the interface frequency does not justify an automated interface. Or the external system has limited data extract capabilities and once I have a manual step in place using Excel it is often tough to justify full automation.

Oracle EBS data extracts to Excel allow users to manipulate the data in simple tool and then re-upload those results back to Oracle EBS. I often find this when there is a functional gap in EBS or the screens are organized in manner which makes mass updates difficult. A good example of this are EBS OM Price Lists, it is often easier to extract the data out to Excel adjust prices then upload the results back to EBS.

Excel is a flexible tool so it is natural for users to evolve business processes around it.

Drawbacks to Excel
Excel lacks auditability and controls for the integrity of data. Due to this Excel is often the bane of IT and Audit professionals. When dealing with business data the goals should be to try to keep business processes in a system which provides:
Authentication - you know who is accessing the data
Authorization - you know they are allowed to access the data
Auditability - we can record what the user has done
Structure - that ensures that data elements on a record remain linked together and foreign key references are maintained

Excel Will Exist in Business Processes
Enough of my soap box lecture. Excel is not going away any time soon. So we need to find a way to use it where it shines and integrate the data back into EBS.

Ways to Extract to Excel from EBS
So since there are legitimate reasons for Excel, what are some methods to get data into Excel from EBS.

EBS Folder Forms allow users to organize data columns they need in an extract, removing extraneous unrequired data columns. You can then save the contents in the EBS folder form directly to Excel.

BI Publisher now allows you to generate a data extract into a native Excel file format.

Many of my clients use Monarch to parse a structured report output into Excel columns. This is very time consuming to setup and maintain, and is very error prone. So I always try to encourage users to switch to another method for creating an Excel extract.

Many third party reporting tools provide the ability to save the report data to Excel.

Oracle APEX (Applications Express) allows users to save data in a report page to Excel. Like a folder form an APEX interactive report allows users to format columns as required, and put additional filters on the data extract to ensure they have the correct set of the data for the given business purpose.

Manipulate Data in Excel

It is very important to make sure that some simple rules are followed when manipulating data in Excel. These may seem obvious but I have had many cases were business data has been hopeless scrambled in Excel by a user that has sorted the data set incorrectly. So to start with ensure that all data is row structured. When data is sorted ensure you have all columns selected.

When dealing with numeric data with leading zeros, ensure that you format those cells as text so you preserve the leading zero. It so much harder to try to put those leading zeros back after the fact.

Dates in Excel can be formatted many different ways. Users always have their preferred date format and it usually not the standard EBS format of DD-MON-YYYY. Make sure that whatever format is selected you can parse the data value back out in your upload tool and that the format is consistent for all data columns and all rows of data.

Methods to Import Data

I am going to compare and contrast some common methods for importing your Excel data back into Oracle EBS. Some of you might have seen before some might be new. The goal is to expose the reader to a broad set of options with a high level assessment of each. If a method seems interesting the reader can research them in more detail using the internet links I have provided.

DataLoader

DataLoader is a commonly used tool. It has its roots in the character based EBS forms where it was possible to structure a data file in a spreadsheet to use the windows cut and paste feature to stream key strokes into a terminal emulator program.

The Classic version is still free. The Professional version costs $590 per seat.

You can get additional information at: http://www.dataload.com/

DataLoader is essentially formatting a stream of keystrokes that are sent into the EBS window. It's spreadsheet like interface helps to structure the navigation and control keystrokes along with data keystrokes. In addition, discrete keystrokes can have various delays to allow EBS to catch up and be ready for the next keystroke. DataLoader is effectively streaming data into the EBS user interface and we have to ensure that EBS is keeping up or else keystrokes will be lost or applied to the wrong field.

The primary benefit to DataLoader is that the Users can easily identify with how it works. It just like how they enter data manually into the system. In addition, since it is going through the EBS screens it uses the same forms validations as manual entry. This is a huge benefit because often there are separate validation routines in the user screens and APIs.

DataLoader is appropriate for a small number of very repetitive sets of records in a single EBS forms region. Large sets of data can often be used to justify an automated interface. Also multiple regions and pop-ups tend to get very confusing with DataLoader.

DataLoader can load data into EBS Forms and Self-Service (HTML) pages (Professional version only).

The primary drawback to DataLoader is that is not robust. DataLoader is just formatting a stream of keystrokes with fixed delays/pauses. So if network response or system response lags at all keystrokes get out of sequence and data is not entered properly. The DataLoad Professional version has a Load Control feature which is supposed to help solve this but I have not personally worked with this feature so I cannot verify if this solves this issue.

Multi page or region forms are tough to get to work, since this is often parent-child data and the navigation between the pages/pop-ups are often inconsistent.

Since DataLoader is just streaming keystrokes to the EBS screens it ties up a PC during it's running. You can't change window focus and any other PC application that pops up will steal the focus and keystrokes will not make it to the EBS window. This often requires a dedicated PC for DataLoaders to run without interruption.

Lastly DataLoader only runs in Windows OS environments.

Oracle SQL*Loader

Oracle SQL*Loader has been around as long as the database. The cost is free, since it is an included tool. There are many web sites and references that describe how to use SQL*Loader. The most common would be found at Oracle.

SQL*Loader requires an Oracle Home and SQL*NET to communicate back to the database. You must have a database userid and password as well. Since it requires an Oracle Home you run SQL*Loader on either a desktop PC that has the software installed on it or on the database or applications tier server.

The Excel file must be saved to a delimited format and transferred up to the computer that SQL*Loader is running on. These two steps often are a source of error, because users must use the same delimiter the SQL*Loader control file expected. Also file transfer routines often have to handle the classic carriage return and linefeed issues various OS's expect. In addition, because the user might need access to the database or application tier server and audit/control issue often comes up.

SQL*Loader also is file intensive since we have multiple files involved in each data upload:
Data File - the source data we are uploading
Control File - which describes the file structure for parsing
Log File - which records SQL*Loader processing (and is hard to parse/decipher as to success/failure)
Bad File - which holds records with formatting errors or that cause Oracle errors during DB insert
Discards File - which holds records not satisfying a WHEN clause in the Control File
The whole jumble of files can be confusing and overwhelming to a non-IT user.

The primary benefit of SQL*Loader is that it has been around as long as the database. This has created lots of reference information with many sources freely available on the internet,

Once you get SQL*Loader working it is reliable and efficient. There are lots of details to help with debugging when things do fail, but as I mention before it is often not something end users can understand.

Lastly SQL*Loader can handle parent child relationships in data easily inserting data into different tables as required.

On the drawbacks of SQL*Loader, there are many ways to parse a file and the declaration of the parsing in the control file can be quite complex. It is often tough to get SQL*Loader working and you will spend time debugging why it is not working.

SQL*Loader has trouble with data at times. It does not handle line feeds in data fields well. In addition, there are inconsistencies in the way Oracle SQL*Loader and Excel expect quoting logic to be carried out. This often leads to incorrectly parsed data when a single quote or double quote shows up in a column of data.

Since SQL*Loader does not read the native Excel file, just the step of requiring the user to save the Excel file to consistent delimited file format can cause issues.

As was mentioned before often access to servers is required or client install is required. That extra step of transferring a file can lead to errors when it comes to carriage returns and line feed.

The group of files required for SQL*Loader make user debugging and upload verification difficult. And as mentioned before the log file is complicated for the average end user to review to determine success or failure.

In general the many steps to process data make SQL*Loader a non-optimal solution:
- Save Excel file to delimited format
- Transfer file to server with SQL*Loader
- Run SQL*Loader
- Review log file
- Still need to call Oracle API's or load interface tables

Often I will put all of this in a Linux shell script to remove the user from the details and register it as a concurrent program.

DB External Tables

As of Oracle Database 9i External Tables have been supported. This is very similar to the Oracle SQL*Loader processing above. The file is read live as user database sessions request the data, parsed and the data is returned to the user. So this is a different strategy than a staging table loaded using SQL*Loader with incremental data. The data file represents the current state of the table, so if you want to see history it must be included in that data file.

Since this is a standard feature of the database there is no additional cost. Most of the Oracle web sites have additional details on DB External Table functionality.

The delimited or structured file must reside on the DB Tier. SQL*Loader is a client based program and it can exist with any Oracle Home. DB External Tables must reside on the database server. Files must be located in a database server filesystem that has been defined in the DB using the CREATE DIRECTORY statement. You create the DB External Table using the Oracle Create Table statement uses ORGANIZATION EXTERNAL clause to describe parsing and file location. Since the DB External Table is a database object it can be accessed by multiple sessions at the same time.

There are limited benefits to DB External Tables. You can plug a text file into the database and read it live like any other table. In general there are less steps than SQL*Loader. Multiple database sessions can access the DB External Table object at the same time.

There are quite a few drawbacks to DB External Tables. You have to place the data file on the DB tier, with file permissions that allow the DB to read the file. You need DBA access to completely setup the processing. Also it is very tough to debug any parsing issues since you need DBA access to review the logs.

Since it is read and parsed live, the current data in the file is what you see, with SQL*Loader you typically are loading into a staging table iteratively. We have the exact same parsing limitations as SQL*Loader.

PL/SQL

PL/SQL code can utilize the UTL_FILE procedures to read a file on the DB Tier server. Since this is included in standard PL/SQL functionality it is a free feature of the database. There are lots of references on Oracle's sites and out on the internet for PL/SQL.

You would typically do this if you don't like the parsing provided by SQL*Loader or DB External Tables. You will take on that file line parsing functionality in your PL/SQL code.

The DBA must first use CREATE DIRECTORY to identify where on the DB Tier the file will reside. Users must transfer the file up to the DB Tier into this directory.

The data file must be saved as delimited or fixed column text format for parsing. The PL/SQL code can use the DB Utilities in UTL_FILE. Specifically you use the use UTL_FILE.GET_LINE to read a line into a variable. Parsing the line into columns is your responsibility and you then have to build a record to insert into the corresponding database table.

There is one primary benefit to PL/SQL you have complete parsing control over the file lines. This is especially helpful if the data if poorly formatted at the source.

The drawbacks to PL/SQL are similar to what we have seen before. You have to save the Excel file in a delimited format. You have to upload the file to the DB Tier server. Lastly this requires PL/SQL development.

WebADI

Web Application Data Integrators are a component of the Oracle EBS Tech Stack. The definition of the WebADI is done in the EBS screens and the metadata is stored in the EBS database. This feature is included in your EBS licensing so there is no additional cost.

There a lot of references on Oracle sites and on the Internet. OAUG even has a WebADI SIG. I have written several papers on the topic that are posted on my web site: http://jrpjr.com

- Custom SubLedger Accounting JE WebADI I/F
- Custom Web ADI Integrators

WebADI starts with the definition of the integrator in the EBS instance. While this might not require a developer, it is complex and often falls into their realm of responsibility. Oracle has seeded many WebADI integrators and it is often beneficial to review how they were implemented when trying to define your own custom one. Your EBS systems administrator has to grant specific access to users to allow them to use a WebADI integrator.

Once the integrator is defined the end user downloads a template file. This template file can include a data extract. If the template is blank the user then has to cut/paste their Excel data into the template file. This is often a source of error since users sometimes don't include the full data set of either rows or columns.

The WebADI integrator file contains the code and authentication necessary to communicate with the EBS APPS Tier. For this reason the WebADI integrator file is instance specific. You can't take a WebADI file that you downloaded in TEST to verify your upload, and upload that same file to PROD. The instance details are embedded in the integrator. You must copy/past the data from the TEST integrator to the PROD integrator to update to PROD.

In addition, since the integrator spreadsheet includes code, you must set the appropriate macro security settings in Microsoft Excel for Web ADI Integrator to work with Microsoft Excel. Since there is a tight integration with Excel, there are version dependencies on which versions of Excel are supported by WebADI.

The user clicks a button to verify and upload the data. This bi-directional communication is between the user's PC and the EBS APPS Tier server. The user receives immediate feedback as to the success or failure of the WebADI load.

There are no file parsing issues since the integrator has access to the internals of the Excel and it can read the data correctly. You don't have to worry about delimiters, quoting, carriage return linefeeds, or data formats. You don't need to save delimited file and reparse since it reads directly from Excel.

Benefits to WebADI are many. You have many WebADI Integrators seeded by Oracle. You just have to put your data into them and upload. You can now develop custom WebADI Integrators.

You don't need to have APPS or DB Tier access. You don't need a DB user/pwd, authentication and authorization handled within EBS.

When this feature first came out I was wildly enthusiastic about them. After having implemented and support several of them I am not as crazy about them as I use to be. The primary drawback I see is that the authentication and instance connection details are embedded in the Integrator file. I can't take the same file and upload to TEST then turn around and upload to PROD. Also this becomes a two file solution because users often do something to the data in Excel prior to uploading in the integrator. So they have to cut/paste the data between the files.

Custom Web ADI integrators are a challenge to build, documentation and terminology is cryptic. However, you can call a PL/SQL routine to complete the interface of data into Oracle API's or load interface tables.

Third Party Tools

There are many third party tools that provide Excel data file upload capability. This is a testament to the business need to move data between Excel and EBS. If there was not such a huge need there would not be so many tools. I am only going to consider two commercial packages specifically geared towards this functionality:

- More4Apps
- GlobalSoftware

I will also discuss an Excel file upload capability associated with developer tools:

- Toad
- SQL*Developer

Consider More4Apps a WebADI Replacement. You can get additional product details at: https://more4apps.com/

More4Apps has a broad product suite of prebuilt Wizards that are ready for users to run once licensed and installed in your EBS instance. Well over 30 transaction specific wizards like: Suppliers, AP Invoices, Items, BOMs, Routings, etc. These take the Excel file data directly into the EBS API's and into the EBS base object tables. So with one click data goes from desktop directly into the EBS where the users expect it to be. No development required, license, install and deploy.

More4Apps becomes an application module in EBS. Metadata is stored in this custom EBS application and becomes part of the EBS database.

More4Apps also allows their tools to be used for custom Application Interface Wizard development. This allows your development team to create an integrator that might not exist, or needs processing that is slightly different from the seeded Wizards available from More4Apps. This offers improved metadata entry and maintenance over WebADI.

This is the primary offering from More4Apps so you know it is their core competency.

GlobalSoftware also has an Excel upload functionality called Spreadsheet Writeback for Oracle. You can research their product offering at https://www.globalsoftwareinc.com/. Their primary business competency is reporting, and this is an off shoot product for them.

Benefits to these third party tools is that they have supported seeded integration functionality. Development is not required unless you choose to. Their products have to stand up against Oracle's free WebADI functionality so they have established some improved functionality to justify the license costs.

As with any third party product both More4Apps and GlobalSoftware have a limited share of internet resources. You are reliant on those companies for support and assistance. Another drawback is the license expense.

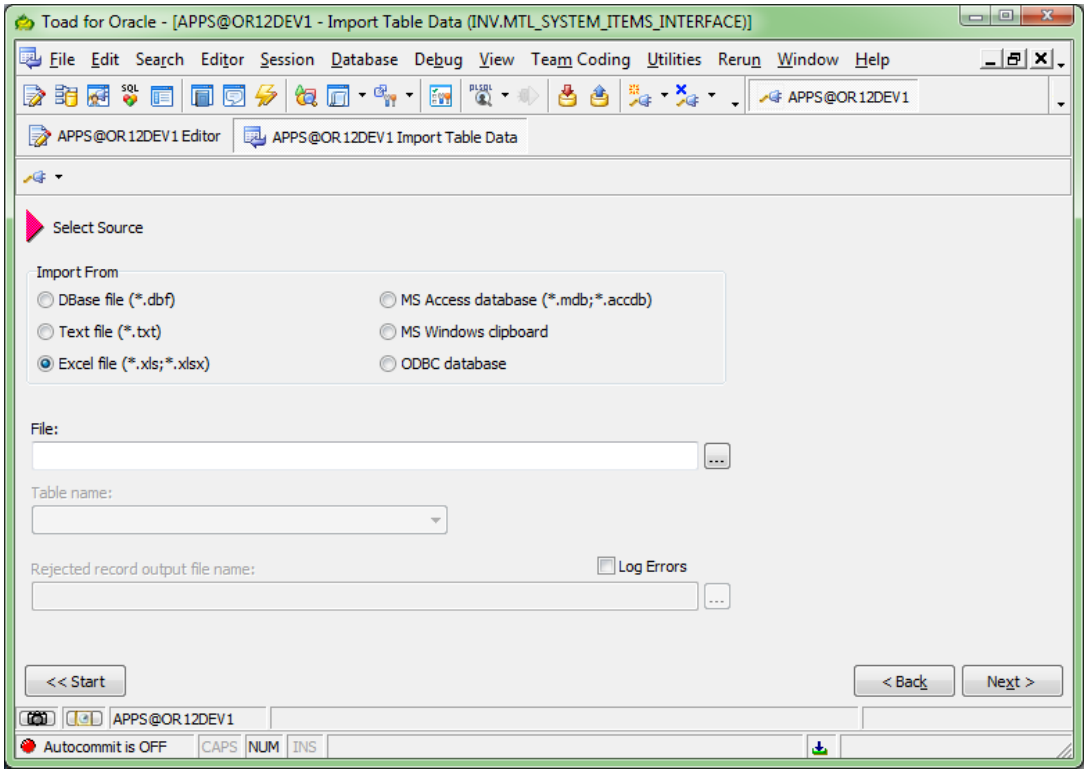
TOAD and SQL*Developer are great development tools. They both have a capability to extract data out of a native Excel file and upload to a database tools. Most people don't realize this functionality exists in these tools. It is a real timesaver for a developer needing to upload some specific data into a table so I wanted to include this in this survey.

Since this is a developer oriented tool set you will need SQL*Net connectivity to the database and a database userid and password.

In TOAD you get to this functionality under the menu path:

Database : Import : Import Table Data

As you navigate through the screens you will see the following screen that allows you specify the source for the data.



The benefits to both TOAD and SQL*Developer loading of Excel file data are:

- Reads native Excel files
- You are already doing development it is a quick and easy way to get data into a database table

The primary drawback is that is a meant for developers not end user.

Oracle APEX

Oracle APEX (Applications Express) is a great rapid application development tool. It is web based with metadata stored in the Oracle EBS instance. It has complete integration with EBS Authentication, Authorization, and Navigation functionality.

Oracle APEX is included in your database license so there is no additional licensing cost.

There are a lot of resources available to learn how to use Oracle APEX. Oracle has a dedicated site just for the product and on that site you can even sign up for a free development account to develop your own applications. Go to https://apex.oracle.com/ for additional details.

This has become my tool of choice and the go to resource I now use for most EBS development. I have several papers on the use of this tool on my web site, http://www.jrpjr.com/.

- Migrate your Discover Reports to Oracle APEX
- Collaborative SIG - APEX Reporting
- Extend EBS Using Applications Express

As I mentioned before Oracle APEX is a rapid development tool. You can stand up some web based functionality quickly, get the users to see it in action, comment and provide feedback and iteratively refine the end applications. This tool can be used for Reporting as well as Data Entry pages. It is a write once and deploy on any platform: Destkop, Tablet, Phone product. Since it runs in the EBS database and is displayed in a user's browser it always requires a live connection to an Oracle DB.

You can also integrate an APEX application page into EBS using EBS form functions. This will allow an APEX page to show up in the an EBS Navigator menu, clicking on the page will launch the APEX page in a web browser.

Now for how to achieve native Excel file parsing functionality. The user client device runs the APEX application in a browser. You develop an APEX page with a file upload button. APEX uses the ORDS (Oracle Rest Data Services) server to communicate with the database. ORDS provides an Excel parsing feature XLS2COLLECTION. This can read a native Excel file with many worksheets and load the data into an Oracle Collection. ORDS XLS2COLLECTION know how to read the binary XLSX file into columns and populates the collection APEX_COLLECTIONS.

Once we have the data in the collection APEX_COLLECTIONS your PL/SQL code can loop through the rows. APEX_COLLECTIONS maps a generic column c001 to hold the worksheet name in the Excel file. 49 additional Excel columns are held in columns c002 to c050. This 49 column limitation is probably the biggest drawback to this solution and is a limitation of the Oracle Collection.

In order to enable this ORDS functionality your DBA must enable the XLS2COLLECTION feature in the ORDS defaults.xml file using the following parameters:

- apex.excel2collection
- apex.excel2collection.name
- apex.excel2collection.onecollection
- apex.excel2collection.useSheetName

The biggest benefit to using APEX is that I typically already have the APEX tool and ORDS component installed in the instance for reporting or data entry functionality extensions. So handling Excel files natively is not a big stretch.

Since the table APEX_COLLECTIONS is already generic in nature, I extend that one step further and have one generic file upload routine. Then I map the c001 to c050 columns in a specific cursor into specific functional column names. This method requires on that cursor that does the select to hold the mapping details. I typically prompt the user to tell me what type of data is being uploaded so I can reference that to know which cursor and downstream processing I need to apply to the data. In addition,

the first row contains the column names for the data, so you can also create an automated or dynamic file mapping routine if you desire.

The drawbacks to APEX based Excel file reading is that you have to have APEX up and running in your environment. The fact that this tool can be used for so many other things minimizes this drawback.

You also have to be familiar with APEX based development. This is a development based solution for reading a native Excel file.

As mentioned before your Excel file is limited by Oracle's collection functionality to 50 columns, with 1 column dedicated to the Excel Worksheet name.

One last drawback is that I currently have not found a way to capture the filename that was uploaded.

Summary and Review

We have reviewed a variety of ways to upload Excel data into the database and ultimately your EBS Application. Each method has some pro's and con's. Some solutions require development. Some solutions require software licenses. The goal of this presentation was to make you aware of these various methods so you can evaluate them for your future use. I hope you found this presentation useful.