Invoice Loader

Purpose

The invoice loader application loads power & gas invoices received in EDI/text file formats (CSV etc.) into a staging database. It allows users to configure how the content in the files will be interpreted as the invoices are mapped to a header/component data structure.

The header/component structure is not intended to be the final invoice representation; rather a staging format where further business logic can be applied to complete processing of the invoices.

Technical Details

The loader application is a SavvyPlus-developed console application written in Python.

It runs on MEL-SVR-AP-001 and is started by a scheduled task. It is designed to run continuously.

The application is located at E:\SavvyApps\InvoiceDataLoader\InvoiceDataLoader.exe and writes log files to E:\ApplicationLogs

The application writes to and reads from the InvoiceLoader database. Configuration table is InvoiceLoaderJobs and file processing records are stored in InvoiceLoaderFiles

The InvoiceLoader database contains a stored procedure RawEDI_to_Staging callable by users to map the loaded content into invoice headers and components. Also a function fileIntegrityChecks which allows users to check for file interpretation problems.

Operation

If the appropriate file format configuration already exists, the process to load an invoice EDI file is:

1. Users places the file in the appropriate queue location for loading
2. Loader application loads file content into the database in a raw format
3. User sets the EDI_format field of the file to match an entry in the Format table
4. User calls RawEDI_to_Staging to create invoice header and component records
5. User calls SQL function fileIntegrityChecks and reviews output to confirm invoices and components have loaded correctly
6. If there are errors in the file's interpretation the file format configuration is modified and steps 4 and 5 are re-run.

File Integrity Checks

e.g. Perform file integrity checks for source_file_id number 35.

```
SELECT * FROM [dbo].[fileIntegrityChecks] (
35)
GO
```

This returns basic information about the file itself (number of rows, invoices and components found and file name).

It also performs various checks at invoice level and component level and issues error/warning messages. Checks include:

Invoice level

- Missing or invalid billing period
- Current charges exc GST + current charges GST = current charges inc GST
- Current charges is not NULL
- Issue date later than billing period end date
- Due date after issue date
- Invoice total = sum of component total
- Invoice must have at least 1 component

Component level

- amount_excl + gst_amount = amount_incl
- component amount not NULL
- invoice charge must be positive, reversal must be negative
- line items must be a CHARGE, ADJUSTMENT, or REVERSAL

Monitoring and Troubleshooting

To trouble-shoot the application consult the log file, or run the application from a dos box to see error messages produced.

Application Configuration

The following parameters can be set in the config file InvoiceDataLoader.cfg located in the deployment folder:

Database Connection

- odbcconnectionstring: ODBC database connection string to connect to database
- retry_time_seconds: time (seconds) to wait between database connection re-tries (e.g. 5)

Refresh Times

- source_locations_refresh_seconds: time interval (seconds) between refreshing the list of source locations to load files from (e.g. 60)
- source_files_refresh_seconds: time interval (seconds) between polling the source folders to look for new files (e.g. 5)
- archive_purge_delay_minutes: time interval (minutes) at which to purge archive folders (e.g. 1)

Configuring Invoice Loading Jobs

Jobs are managed through the InvoiceLoaderJobs table. The fields are as follows:

Field	Purpose
source_folder	The folder to watch for files to process. This should be a local folder on the server running the application as reliable access is assumed.
success_folder	The folder in which to place files after successful processing. This is also the location against which the file purging process will operate.
fail_folder	The folder in which to place files that are not successfully processed.
priority	The job priority. Jobs with lower numbers will be processed before those with higher numbers
active_flag	1 = active, 0 = inactive. Use to temporarily disable jobs without deleting configuration
filename_pattern	dos-style file name matching string e.g. *.zip
handler	text matching an entry in the table of handlers below
handler_params	Text that evaluates to a Python dict containing parameters to pass to the handler. Description/examples below
success_retention_days	Number of days to retain archived files. Files in archive older than this date will be purged. -1 indicates keep files forever
organisation_id	Integer to indicate which organisation the invoices belong. Not used
EDI_format	Text to indicate what file format is expected (e.g. Aurora invoice file)
provider_name	Text indicating the name of the provider of the file e.g. AURORA

Loader - Existing Handlers

Handler	Role
csv_handler	Load invoice data in a CSV-like format into raw EDI tables.
move_only	File is moved from source to archive folder without further processing
unzip	Unzips the file in-place i.e. extracts the contents of the zipped file to the same location.

Configuring CSV file formats

The way CSV content is mapped to invoice information is set in tables Format, FormatColumn and FormatComponent. The format interpreter can cope with invoice-per-row or component-per-row formats.

As part of its processing the RawEDI_To_Staging stored proc creates a temp table that resembles the CSV file plus adds a *row* column. This allows the user to write SQL expressions to process the file contents into the desired structure. The SQL expressions are executed as dynamic SQL - IF CALLED BY AN APPLICATION THIS STORED PROC SHOULD BE RUN FROM AN APPLICATION ACCOUNT WITH MINIMAL DATABASE PERMISSIONS.

Format table

This table contains 1 entry per file format. Fields are:

- EDI_Format - primary key for table. Code to represent the EDI file format
- Description - User description, not used by system
- HeadingRow - integer, row of file that contains column headings
- nFooterRows - integer, number of rows with footer information to ignore
- invoice_unique_expression - SQL expression that identifies unique invoices within the file.
- *_expression - SQL expression that returns the item of information assuming it is run against a table that contains:
    - row - the row number from the file
    - columns named as per the HeadingRow contents

FormatColumn table

This table should contain 1 entry per allowable column in each file format. Fields are:

- EDI_Format - Which EDI_Format does this column belong to
- column_name - text that appears in HeadingRow of CSV file
- is_required - 1 if that field is required to appear in file, 0 if it's an optional field
- header_detail_label - if you want the values in this column captured in InvoiceHeaderDetail then the label to be applied, otherwise NULL
- component_detail_label - if you want the values in this column to be captured in InvoiceComponentDetail then the label to be applied, otherwise NULL

FormatComponent table

This table will be joined with the invoice CSV replica and should contain enough records that the result will be 1 record per invoice component. For example if input file format is 1 invoice component per row, you may only need 1 entry in the FormatComponent table. If the file format has 50 columns, 1 for each different invoice component type, you will need 50 entries in this table to capture that information.

Fields are:

- EDI_Format - Which EDI_Format does this entry apply to?
- row_filter_expression - SQL expression that returns 1 if an invoice component should be returned, 0 if not
- *_expression - SQL expression to capture the relevant value for each field
- only_if_file_has_column - use this FormatComponent entry only if the specified column is found in the file - avoids runtime errors with SQL expressions on unknown fields
- item_type_expression - expression to return CHARGE, ADJUSTMENT or REVERSAL

Compiling

The application is built using Python library PyInstaller. A text file named compile_instructions.txt in the application's source code folder contains instructions to compile the application.

Compilation requires Python and a number of additional libraries to be installed. For details of libraries/versions installed see Python Libraries pag
e.

Notes

When csv_handler is used raw data is loaded into table RawEDI

RawEDI_To_Staging creates entries in:

- InvoiceHeader - basic invoice header info
- InvoiceComponent - basic invoice component info
- InvoiceHeaderDetail - detailed/non-standard invoice header info
- InvoiceComponentDetail - detailed/non-standard invoice component info