



Gracenote On Product Demo Kit

Version: 1.4.0

Published: 07 Apr 2025

DISCLAIMER:

THE PDK IS NOT MEANT TO BE USED AS A PRODUCTION-READY SYSTEM. IT IS MERELY A DEMONSTRATION OF ONE APPROACH (AMONG MANY) TO IMPLEMENT THE INGESTION BACKEND.

CLIENTS WILL NEED TO CREATE THEIR OWN CODE AND SYSTEM FOR LIVE PRODUCTION OR OTHER USES.

Gracenote, Inc.

A Nielsen Company

<http://www.gracenote.com>

[Requirements](#)

[Platform](#)

[Databases](#)

[Applications](#)

[Python Dependencies](#)

[Installation](#)

[Windows](#)

[MacOS/Ubuntu/Linux](#)

[API Settings](#)

[Dependencies](#)

[Upgrades](#)

[Installation directory](#)

[DB schema changes](#)

[Usage](#)

[Using Seed Files](#)

[SFTP Hosts](#)

[Using Stats](#)

[Consuming Data](#)

[TablePlus](#)

[Sample Queries](#)

[SQLite CLI Client](#)

[Basics](#)

[Export](#)

[Import](#)

[Batch Mode](#)

[Structure](#)

[Configuration](#)

[API Credentials](#)

[FTP Credentials \(seed files\)](#)

[SQLite](#)

[Postgres](#)

[MySQL](#)

[Misc](#)

[Postgres-specific Config](#)

[Utilities](#)

[Limitations](#)

[Dependencies](#)

[Mac](#)

[Ubuntu](#)

[Windows/MSYS2](#)

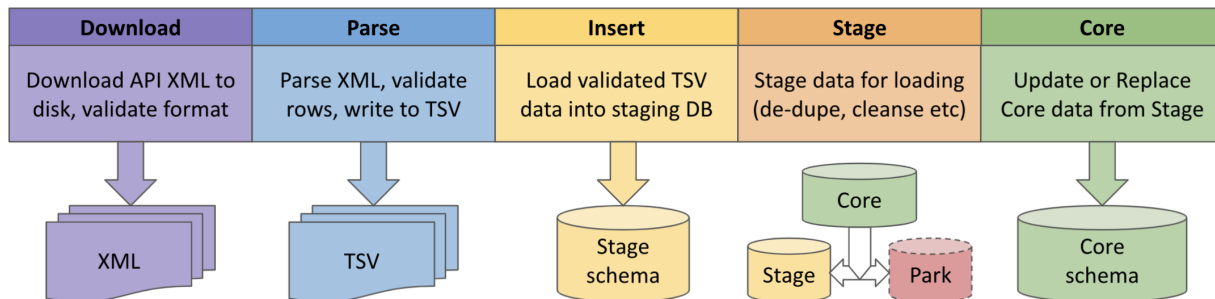
[History](#)

Overview

Gracernote On Product Demonstration Kit is a sample implementation of the ingestion backend for Gracernote On data delivery API. The purpose of the kit is to speed up the evaluation of Gracernote data delivered via the API and the implementation of customer ingestion systems. The kit currently demonstrates how to work with the API and provides sample parsers and database schema.

DISCLAIMER: THE PDK IS NOT MEANT TO BE USED AS A PRODUCTION-READY SYSTEM. IT IS MERELY A DEMONSTRATION OF ONE APPROACH (AMONG MANY) TO IMPLEMENT THE INGESTION BACKEND. CLIENTS WILL NEED TO CREATE THEIR OWN CODE AND SYSTEM FOR LIVE PRODUCTION OR OTHER USES.

When started, the kit downloads the API data until no more updates are available and then processes the downloaded batch through the ETL steps:



The implementation is built using Python, shell scripts, widely available tools such as curl, xmllint, sqlite; it runs natively on Linux and Mac, it also runs on Windows using MSYS2 environment (based on Cygwin).

Requirements

Platform

- Ubuntu/Linux
- MacOS
- Windows/MSYS2 or /WSL

Databases

- SQLite (recommended)
- Postgres
- MySQL

Applications

- curl, xmllint, sqlite3
- Python 3.x (latest recommended)

Python Dependencies

- Python 3.x
 - SQLite: sqlite3 module - library version 3.14+
 - Postgres: psycopg2 module (also needs Postgres client libraries)
 - MySQL: mysql-connector-python module (also needs MySQL client libraries)

Installation

Windows

- Download and install MSYS2 environment: <https://www.msys2.org/>
- Update MSYS2 environment: <https://www.msys2.org/docs/updating/>
- Install Python into MSYS2 environment `$ pacman -S python`
- Install SQLite and libxml2 into MSYS2 environment
 - `$ pacman -S sqlite3`
 - `$ pacman -S libxml2`
- Download the PDK archive into your home directory within MSYS2 environment (e.g. C:\msys64\home\username) and unzip it into a folder (e.g. PDK-v1.0)

MacOS/Ubuntu/Linux

- Download the PDK archive into your home directory and unzip it into a folder (e.g. PDK-v1.0)

API Settings

Your Gracenote customer care representative will have provided you with the demonstration API key to be used with the PDK. Create an empty configuration file (e.g. config/my.conf) and set the APIKEY parameter to the value of the key. See [Configuration](#) section for details.

Dependencies

When run, the script will check whether required dependencies (system and Python modules) are satisfied and will provide feedback if they are not. The environment recommended in the requirements section (latest Python 3.x) should satisfy all Python dependencies for the default configuration of the PDK.

[Dependencies](#) section contains approaches for resolving dependencies, however note that each system is different and it's not possible to provide a universal recipe to resolve an arbitrary dependency. If you're running into issues, please consider using the recommended environment.

Upgrades

This section covers a scenario when you already have an existing PDK and are upgrading to a newer version.

Installation directory

It is recommended to install different releases of the PDK in separate directories

DB schema changes

If the new release calls out DB schema change, then the new release cannot be used with the DB schemas from the previous release. An exception to this is when you're using the default DB engine - SQLite (which uses file-based databases) - AND when you're installing different PDK releases in different directories. In all other cases, either drop the previous release schemas (accumulated data will be lost) or change the schema prefix in the PDK configuration file of the new release to make it distinct from the previous release (accumulated data will be preserved).

Usage

Start the run script in the terminal with your configuration file as a parameter.

- `./run.sh -c my.conf`

The run script will download the data from the API using stored `update_ids` (or zero if it's a first start) until there are no more updates; will parse the downloaded data into a tabular format (TSVs), insert the data into the staging schema and from there will update the data in the core schema (see diagram in the Overview section).

The script can be run as often as needed to get the latest data, load it in staging and update the core database. The initial update ("cold start") with the demonstration API key should take less than an hour with a bulk of time (~75%) consumed by the download phase.

The script expects to be started from its directory so if you run it on cron, use something like `cd /home/user/pdk_v0 && ./run.sh -c my.conf`

In the event of an error, the script has limited ability to store its state and then, on subsequent run, proceed to the point of interruption and continue without impacting data integrity. This is

meant for user interruption (Ctrl-C) as well as simple errors such as running out of disk space or not being able to connect to the database. In case of exceptions unhandled by the code, or multiple consecutive interruptions by errors in different places it is recommended to clear the state and reload the data from update_id zero (after addressing underlying issues). Refer to the [Utilities](#) section below for helper scripts to clear out the PDK state and data.

Using Seed Files

To use the seed files, set seed file FTP credentials and path in your configuration file. Do not use trailing slash in FTP_PATH:

- FTP_HOST=ftp://on.tmstv.com
- FTP_USER=my_user
- FTP_PASS='my_pass'
- FTP_PATH=path/to/seed/files

Also, set the YYYYMMDD matching regex for the seed manifest file name. For example, regex for cxlb_file_manifest_20200415.xml manifest file name would look like so:

- SEED_MANIFEST_REGEX='cxlb_file_manifest_\([0-9]{8}\)\.xml'

Here, `\([0-9]{8}\)` is a pattern to match 8 sequential digits (YYYYMMDD) located between `cxlb_file_manifest_` and `.xml` (dot is escaped with backslash). The regex must be enclosed in single quotes.

Start the run script overriding the default delivery mode (API) with -d option SEED:

- `./run.sh -c my.conf -d SEED`

SFTP Hosts

For SFTP hosts, specify protocol accordingly: `sftp://on.tmstv.com`. Below are the troubleshooting steps for common errors:

- *curl: (1) protocol "sftp" not supported or disabled in libcurl*
 - Ensure you have the latest version of curl installed
- *curl: (60) SSL peer certificate or SSH remote key was not OK*
 - Verify that you're able to connect to the SFTP host using sftp utility and your SFTP username: `sftp my_user@on.tmstv.com`

Using Stats

Start the stats script in the terminal with your configuration file as a parameter.

- `./utils/run_stats.sh -c my.conf`

The script will execute the `db/.../stats.sql` file against your configured database. On the first run, this will create `_stats` table to collect the metrics, and will then run a couple of metrics on key entities - specifically count of ingested objects per ingest job and count of updated objects per time interval (15min). The SQL script is meant to be run periodically (e.g. on cron); default stats queries are written to aggregate unprocessed data (via checking against maximum `_stats` timestamp for a given metric), so that the repeat runs of the script on the same dataset would not insert new stats. The data in `_stats` can then be viewed as timeseries using visualization tools like Grafana.

The `stats.sql` script can be enhanced with additional metrics for `_stats` table; the run stats script also supports optional command line options that enable customization of the stat scripts being run: `-s alt_sql_suffix`, `-o schema_override`, `-p parameters`

- Run a different stats SQL file using `-s` option: `./utils/run_stats.sh -s daily` will run `stats_daily.sql`
- Run stats against a different (vs configured) schema using `-o` option:
`./utils/run_stats.sh -o on_customer_99999_full_schedule_core`
- Pass pipe-separated ordered parameters to the SQL file using `-p` option. Parameters are referenced as `{1},{2},...` from SQL: `./utils/run_stats.sh -p "60m|12h" - {1}` will evaluate to 60m, `{2}` to 12h

Consuming Data

Once the script ran successfully, it appears to have fetched some data and loaded it up somewhere. That's great but how can we work with this data?

The default configuration uses file-based SQLite database engine and loads the data into `.db` files in the root directory of the PDK. Two files are created - core and stage databases (names below are examples, your schema names may differ). The data is accumulated in the core database, while the stage database serves as a temporary place to store and stage the incoming data before loading it into the core database.

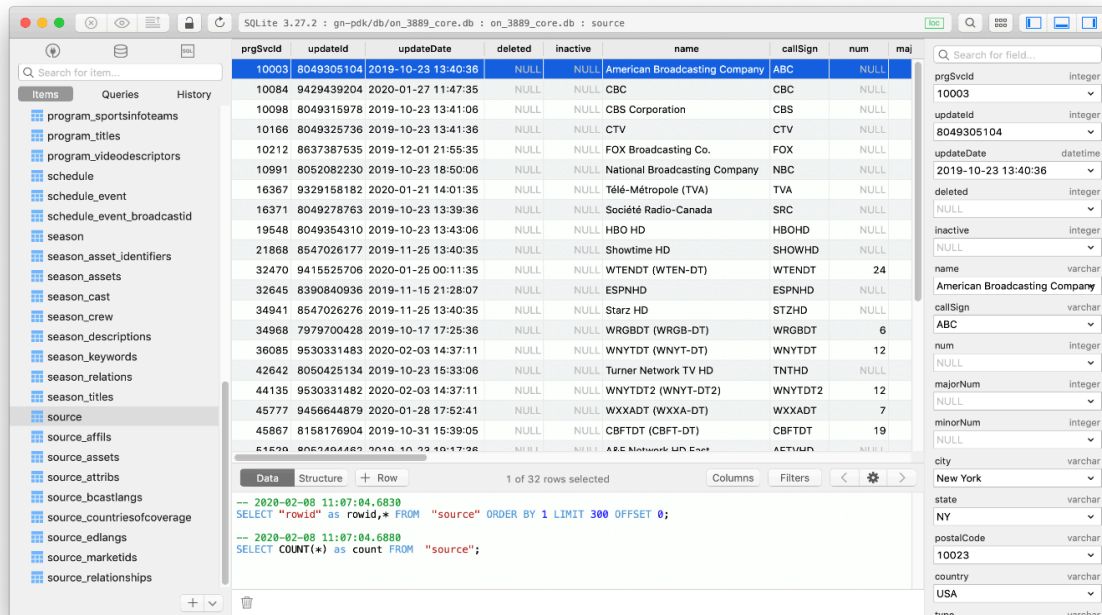
Unset

```
$ ls -l *.db
on_sample_core.db
on_sample_stage.db
```

The following tools can be used for viewing and interacting with the database:

TablePlus

If you're on Mac or Windows, you can use [TablePlus](#), which is a free and easy-to-use database client, allowing you to view database structure and data, run queries and export query results. TablePlus supports multiple database engines and will provide the same familiar interface if the PDK is configured to use another database engine. TablePlus is a great way to work with data in interactive mode.



Sample Queries

The following sample SQL queries are provided in the docs/sample_queries/ directory:

- source_info.sql - extract basic TV channel information with some metrics
- channel_schedule.sql - extract the schedule of a channel along with basic program information in a table with each airing being one row
- program_details.sql - extract one or more program records with more detailed information, without images and video descriptors
- program_videodescriptors.sql - VideoDescriptor retrieval for a TMSId
- videodescriptor_taxonomy.sql - VideoDescriptor full taxonomy in matrix/pivot representation

SQLite CLI Client

If you don't mind the command line and/or need to work with data in automated fashion, SQLite command line client is a versatile tool that can run SQL queries, import tabular data into tables and export it from either tables or query results, in interactive or batch mode.

Basics

Unset

```
$ sqlite3 on_sample_core.db
SQLite version 3.24.0 2018-06-04 14:10:15
Enter ".help" for usage hints.
sqlite>
sqlite> -- view db schema
sqlite> .schema
CREATE TABLE IF NOT EXISTS "source" (...
...
sqlite>
sqlite> -- open another db; will close the current db; empty db will be
created if file doesn't exist
sqlite> .open some_other_database.db
sqlite>
sqlite> -- open multiple dbs
sqlite> ATTACH 'path1/database_1.db' as db1      -- to query, use prefix:
db1.table_name
sqlite> ATTACH 'path2/database_2.db' as db2      -- to query, use prefix:
db2.table_name
sqlite>
sqlite> -- show table header in queries
sqlite> .headers on
sqlite>
sqlite> -- run SQL query
sqlite> SELECT * FROM source LIMIT 10;
prgSvcId|updateId|updateDate|deleted|inactive|name|callSign|...
10003|8049305104|2019-10-23 13:40:36|||American Broadcasting
Company|ABC|...
...
sqlite>
sqlite> -- exit client
sqlite> .exit
```

\$

Export

Unset

```
sqlite> -- specify output format (tab=TSV, csv=CSV)
sqlite> .mode tab
sqlite>
sqlite> -- specify output file
sqlite> .output my_export.tsv
sqlite>
sqlite> -- run query; results will be written into the file
sqlite> SELECT * FROM source;
sqlite>
sqlite> -- redirect the output back to console
sqlite> .output stdout
```

Import

Unset

```
sqlite> -- create a table to import the file to (assume 2 columns)
sqlite> CREATE TABLE import_table(column1 text, column2 int);
sqlite>
sqlite> -- specify file format (tab=TSV, csv=CSV)
sqlite> .mode tab
sqlite>
sqlite> -- import the file
sqlite> .import import_file.tsv import_table
```

Batch Mode

Script to output table row count

```
Unset
-- output_count.sql script --

.print SQLite: running table counts...
.mode tab
.output table_counts.tsv

SELECT 'source', COUNT(*) FROM source;
SELECT 'schedule', COUNT(*) FROM schedule;
SELECT 'celebrity', COUNT(*) FROM celebrity;
SELECT 'program', COUNT(*) FROM program;
SELECT 'season', COUNT(*) FROM season;
```

Running the script

```
Unset
$ sqlite3 on_sample_core.db < output_count.sql
SQLite: running table counts...
$
$ cat table_counts.tsv
source 32
schedule 480
celebrity 1113904
program 35482
season 6139
$
```

Structure

- bin - shell and Python scripts including parsers
- config - configuration files for PDK and databases
- db - SQL scripts
- files - temporary XML and TSV files
- logs - run logs
- utils - [utility scripts](#)
- vars - variables containing current PDK state

Configuration

Default configuration parameters are contained in the **config/default.conf** file. Rather than editing this file directly, it is recommended to create a custom configuration file (e.g. **config/my.conf**); any parameters there will override default parameters when **run.sh** script is given the custom config file with **-c** option. This makes it easier to transfer your configuration to another version of PDK in the future. The custom config files must be located in the **config** directory, and given to **run.sh** script without path (e.g. **./run.sh -c my.conf**).

Configuration files can be edited by nano editor available on most Linux/Mac environments

- `$ nano config/my.conf`
- Use Ctrl-O to save the edited file and Ctrl-X to exit the editor

The following provides a brief description of important configuration settings

- DATABASE - database engine used by PDK - SQLITE or PGSQL or MYSQL or MONGO. Default: SQLITE
- SCHEMA_PREFIX - partial schema name to which *_stage* and *_core* suffixes are appended to to form full schema names (or .db file names in case of SQLite). Default: on_sample
- DB_DELETES - whether to delete objects flagged for deletion by the API from the database. Default: TRUE

API Credentials

- BASE_URL
- APIKEY
- STREAM_ID - optional, use only if streamId was provided

FTP Credentials (seed files)

- FTP_HOST
- FTP_USER
- FTP_PASS - supply password in single quotes
- FTP_PATH - no trailing slash
- SEED_MANIFEST_REGEX - YYYYMMDD matching regex for the seed manifest file name

SQLite

- SQLITE_DIR - location of SQLite .db files. Default: . (PDK root dir)

Postgres

- PGSQL_HOST
- PGSQL_PORT
- PGSQL_USER - the user must have schema creation privileges
- PGSQL_PASS - supply password in single quotes. If using default user (postgres), be sure a password exists for that user. Please refer to Postgres documentation for further info.
- PGSQL_DBNAME - db_name must be existing database - the script is not able to create one

MySQL

- MYSQL_HOST
- MYSQL_PORT
- MYSQL_USER - the user must have schema creation privileges
- MYSQL_PASS - supply password in single quotes

Misc

- WITH_XML_AFTER_PARSE - what to do with XML file upon successful parse - DELETE or LOG. Default: DELETE
- WITH_TSV_AFTER_IMPORT - what to do with TSV file upon successful import into staging schema - DELETE or LOG. Default: DELETE
- NULL_VALUE - value designating NULL in TSV files. Default: __null__
- APPEND_TSV - setting this to TRUE can work in conjunction with DATABASE=NONE to accumulate all updates in TSV files without loading them into a database. Default: FALSE
- LOG_INVALID_SOURCE - keep source XML for rows that fail validation. Default: FALSE
- RDB_DATA_LOAD - only load subset of data (FULL|SHELL|*arbitrary_label*) into Relational DB, to conserve space/speed up DB ops. FULL option loads everything. SHELL option will load only the following for each entity (parent table): primary key, updateId/Date, deleted/inactive flags, MD5 and PDK ingestId fields. Specifying arbitrary label will result in loading of SHELL option data plus any fields specifying said label in config/fields.tsv, column #10. Multiple comma-separated labels can be specified per field (no spaces, case-sensitive, do not use label SHELL or FULL). Default: FULL

Postgres-specific Config

- PG_CORE_UPDATE_THREADS - use multiple threads to run core updates on the database. Default: disabled (single thread).

- `PG_KEEP_IDX_ON_UPDATE` - keep non-key indexes and constraints during core update. When `FALSE`, non-key indexes and constraints in the core schema will be dropped and recreated after the update completes. Will have no effect on user-created indexes in core schema. Can be useful for running large updates, e.g. after ingestion pause. Default: `TRUE`

Utilities

- `$./utils/show_vars.sh` to show variables such as latest `update_ids`, last run status etc
- `$./utils/rm_all.sh -c my.conf` to clear out all variables, any xml/tsv files and specified schemas from the database in the given configuration file
- `$./utils/rm_schema.pgsql.sh -c my.conf` to drop specified schemas from PostgreSQL database in the given configuration file
- `$./utils/rm_schema.mysql.sh -c my.conf` to drop specified schemas from MySQL database in the given configuration file
- `$./utils/rm_schema.sqlite.sh -c my.conf` to drop specified SQLite databases in the given configuration file
- `$./utils/rm_vars.sh` to clear out all variables and reset `update_ids` to zero. This does not delete the data in the database.
- `$./utils/rm_xml.sh` to clear out xml files. Normally there shouldn't be any unless the job was interrupted, in which case you may want to keep them.
- `$./utils/rm_tsv.sh` to clear out tsv files. Normally there shouldn't be any unless the job was interrupted, in which case you may want to keep them.

Limitations

- PDK path must not contain whitespaces
- While user interrupts (Ctrl-C) are supported, terminating the PDK process via other means could leave it in an inconsistent state. To recover, it is recommended to clear the state and reload the data from `update_id` zero
- Deleted shell objects (i.e. those consisting solely of object id, update id/date, deleted flag) arriving from certain API endpoints (e.g. On API Celebrities) are ingested into Core as is (in `DB_DELETES=FALSE` mode).

Dependencies

This section contains some approaches for resolving dependencies, however note that each system is different and it's not possible to provide a universal recipe to resolve an arbitrary dependency. If you're running into issues, please consider using the recommended environment.

Mac

- Python 3.x
 - Postgres
 - `pip3 install psycopg2`
 - MySQL
 - `pip3 install mysql-connector-python`

Ubuntu

- xmllint
 - `sudo apt-get install libxml2-utils`
- curl
 - `sudo apt-get install curl`
- SQLite libs
 - `sudo apt-get install sqlite3 libsqlite3-dev`
- Postgres libs
 - `sudo apt-get install postgresql-client libpq-dev`
- MySQL libs
 - `sudo apt-get install python3-mysql.connector`
- Python 3.x
 - Postgres
 - `pip3 install psycopg2`
 - MySQL
 - `pip3 install mysql-connector-python`

Windows/MSYS2

- SQLite
 - `pacman -S sqlite3`
- Python 3.x
 - `pacman -S python`

History

- v1.4.0
 - OnSports support with SportsEvents / Sports endpoints -> DB schema change
- v1.3.20
 - Update schema for OnSports compatibility -> DB schema change
 - Update schema to capture product changes (schedule eventId, source_altname) -> DB schema change
 - Modified/improved support for Main-Dependent entity decomposition/reconstruction
- v1.3.19
 - Add sample queries
 - Stats - streamdetails with Pg/My/Lt output, release streamdetails gen to utils
 - Add run_stats reentry check
 - HASH_ORIG_DATA option (default=TRUE) to include full data (sans updateId/Date) into MD5 calculation, regardless of RDB_DATA_LOAD setting
- v1.3.18.1
 - Update schema to capture product changes (program_epinfo:subType, celebrity_participanttype:isPrimary) -> DB schema change
 - Include program_title:subType (type=full) as program:title_subType -> DB schema change
- v1.3.18
 - Update schema to capture product changes (program rel label, source_transportid, sport eventId) -> DB schema change
 - Update SourcePrograms schema -> DB schema change
 - Add ProgramAnnotations endpoint -> DB schema change
 - Workaround for missing VideoDescriptorTaxonomyItem identifier (use themeld for descriptortaxonomy)
- v1.3.17
 - Update schema to capture product changes (SourcePrograms, restricted asset flag) -> DB schema change
 - Stats: add parameter support / more flexibility to the run script + add dimension field to stats table
 - Multithreaded PostgreSQL core update edge case handling - check pid count before waiting for pids
- v1.3.16
 - Update schema to capture product changes (origAudioLangs, sourceGroups, exactStart/EndDateTime) -> DB schema change
 - Experimental support for multithreaded PostgreSQL core update
 - Interrupted core update sequence now resumable from a stored batch number (vs from start)

- Support for UPDATE_FLAGS feature (workaround for deleted/inactive "shell" API updates)
 - Fix for "composite" timestamp in seed file download
 - Include dl_perf.sh and status.sh utils in product PDK releases
 - Docs: deprecate SchemaSpy due to lack of support for current GraphViz version
- v1.3.15
 - Support for Postgres Staging and Core Update by entity / parallelized
 - Add (MySQL,SQLite) or refactor (PgSQL) support for implicit inactives
 - Reorg/rename Postgres DB replace core files; reorg/rename SQLite/MySQL DB files to match Postgres convention
 - Fix missing delete flag in shell/skeleton ingest mode
 - Fix extraction of deleted/inactive flags in Source parser
 - Fix to properly create and drop stage indexes in Pg
- v1.3.14.2
 - Documentation update
- v1.3.14.1
 - Temp fix to prevent deletion of inactive dependent entities
 - Fix issue with missing updates due to discrepancy in max batch ord and row count
- v1.3.14
 - Added missing fields (celebrity:nationality,ethnicity,baseline asset rank; season_asset:seasonId; source:reach) -> DB schema change
 - Modified VideoDescriptors schema/parser to match delivery packaging (in taxonomyItems) -> DB schema change
 - Added RDB_DATA_LOAD=ENTITY_PKIDX shell ingest mode: loads entity PKs, update info, indexed fields which includes all FKs
 - Shell ingests (RDB_DATA_LOAD=ENTITY_PK_ONLY) now load "inactive" field
 - Added API_CALLS_MAX option to limit # of calls in endpoint download (smaller updates for MySQL)
- v1.3.13
 - Added schedule_event:exactSource -> DB schema change
- v1.3.12
 - Bug fixes, documentation updates
- v1.3.11
 - Schema modifications to reflect product updates (transport ids, ratings) -> DB schema change
 - Fixed issue with indexes not being dropped in Postgres core replace
 - Separated insert flow from staging flow; updated flow diagram
 - Added DB schema version check
- v1.3.10
 - Schema modifications to reflect product changes -> DB schema change
 - Various parser fixes
- v1.3.5

- Schema modifications to reflect product changes -> DB schema change
 - Support On API CV endpoint
 - Support database deletes
 - MongoDB support (beta)
 - Various parser/logic fixes
 - v1.2.13
 - Support all On API endpoints incl. Enhanced Celebrity -> DB schema change
 - Expand array of abnormal conditions after which PDK can be restarted and still preserve data integrity (now includes errors/user interrupts/process terminations)
 - MySQL support
 - Various parser/logic fixes
 - v1.1
 - Custom config files to override defaults -> config file change
 - OVD, VOD, VideoDescriptor endpoints -> DB schema change
 - Support for seed files provided via FTP
 - v1.0
 - Initial GA release
-