Index Exchange Import Suite

Spring 2023

–

Robert Bannocks
Prog Sales AB

Document version

version	date	comments
0.1	Initial release	Alpha
0.2	Second release	Beta, Added list of files at the end of the document an move to shelve database.
0.3	7 July 2023	Final
0.4	(in) the future	Take account of move of fileid and reportID to the MySQL database.

Software releases

Name	version	date	comments
Trial version	0.1	14 April 2023	First to take data from Index Exchange and push to MySQL
First in the wild	0.2	21 April 2023	First release used in anger
Second edition	0.3	02 May 2023	Includes code to check if reports already exist on Index Exchange and maintain a database of pairs of publisherID, report name and reportID
Third edition	0.4	04 May 2023	Move database from JSON to Shelve to facilitate multiple instance of main program running at the same time
Production 1	0.5	12 May 2023	
Production 2	1	2 June 2023	Full CAL API functionality
Production 3	2	(in) The future	Move the fileID and reportID databases from shelve to the MySQL DB

Overview of the suite

This documentation details a suite of programs written for Prog Sales AB whose purpose is to fetch via REST API data from Index Exchange and push this to a MySQL database. Currently two of the APIs are used the reporting API and the CAL API. This document details the reporting API code. Another document details the CAL code which is an extension of this code and uses some of the files and functions written for the reporting API. All the code is written in Python.

System components

The system consists of a number of components. Firstly, there are the programs and files in this suite being described here. Some of these files contain essential data for the suite to operate (report_to_db_table_mapping.py, and the authentication information in the .env-secrets file). Secondly there is the Linux machine on which these scripts are run. Thirdly, there is the MySQL database instance used to store downloaded data. Fourthly, there are the local stores of data which are separate from the suite (i.e. not in the GitHub repository but are held on local file stores on the server outside the Python code for the suite. Lastly, there is the instance of the Grafana tool.

Currently the scripts, i.e. programs that form this suite, the MySQL database instance, the local stores, and the Grafana instance are all hosted on the same machine.

System operation

Overview of the use of the reporting API

The code for the reporting API will fetch from local storage each named report given on the command line, upload that report, run it, download the resultant data, and upload that data to the MySQL database. It will do this for each `publisherid` listed in the prog_ix_publishers table in the MySQL database. Hence the number of reports upload is the cross product of the number of entries in the prog_ix_publishers table in the database and the number of reports listed on the command line.

All tables referred to in this document are in the prog_stats database within MySQL.

Once the data is in MySQL it can be visualised in the Prog Sales instance of Grafana.

Reports available

Various reports have been defined for Prog Sales use and they are maintained on the Prog Sales Server in a store called the templates store details of which are given below. Reports are Index Exchange reports coded in JSON. They are uploaded to the Index Exchange servers and run there. Reports, as shown below were defined on the Index Exchange web site and then downloaded to the templates store using the `get-report-template.py` utility program.

Tracking and disposal of reports on the Index Exchange infrastructure

Internally the code maintains a database of tuples of (publisheID, report name, reportID). This is used to reuse existent reports on the Index Exchange infrastructure so that the system does not become overwhelmed with new automatically generated reports. In the event that there is not an entry for a combination of publisherID and Report name a new report will be created on the Index Exchange server and recorded in the local database of reportIDs. Once all reports are updated or created, then they are run and the results, which consist of a list of fieIDs, are stored. Then for each fileID the corresponding file is downloaded from Index Exchange and then the resultant data is inserted into the MySQL database.

If for some reason the local database is lost or corrupted the program tidy-reports.py can be run to remove reports from the Index Exchange server which are not in the local database of reports (this only removes reports with a report title matching the auto generated reports title form(ulation)).

Controlling insertion of data into the MySQL database

A database in the form of a file, the report_to_db_table_mapping.py, file needs to be maintained. This tells the suite which MySQL database table to insert the data downloaded from Index Exchange for a given report. All tables are assumed to exist within the prog_stats database and that the MySQL user used to run the insert has to have permissions to read and write to the table within the database.

MySQL database tables

These are listed in the section below. All tables exist within the database prog_stats database.

Manual alignment of report fields and database table definition

The code can only assume the field types returned from Index Exchange are the same as those in the relevant MySQL database table definition (as determined by the report_to_db_table_mapping.py file). There is no way to coordinate the 2 and hence the task of coordinating the report definitions and the MySQL database table definitions needs to be performed manually.

The code takes care of converting all the relevant data types across, this is performed automatically by the mysql_connector Python module. This cannot cope with data which is of the wrong type for the database column and cannot be cast. I.e. converting alphabetic characters to integers. Hence the need for the external coordination of the report output and the database table definitions.

The code modifies the downloaded report definitions by modifying the Report Title to a standard from (this allows the tidy-report.py program to recognise automatically generated reports not in the reportID database), make the download format CSV.TXT, and changes the account_id in the report for each upload/update and run as the report needs to be uploaded/updated and run once of each publisher_id.

Operation of the suite

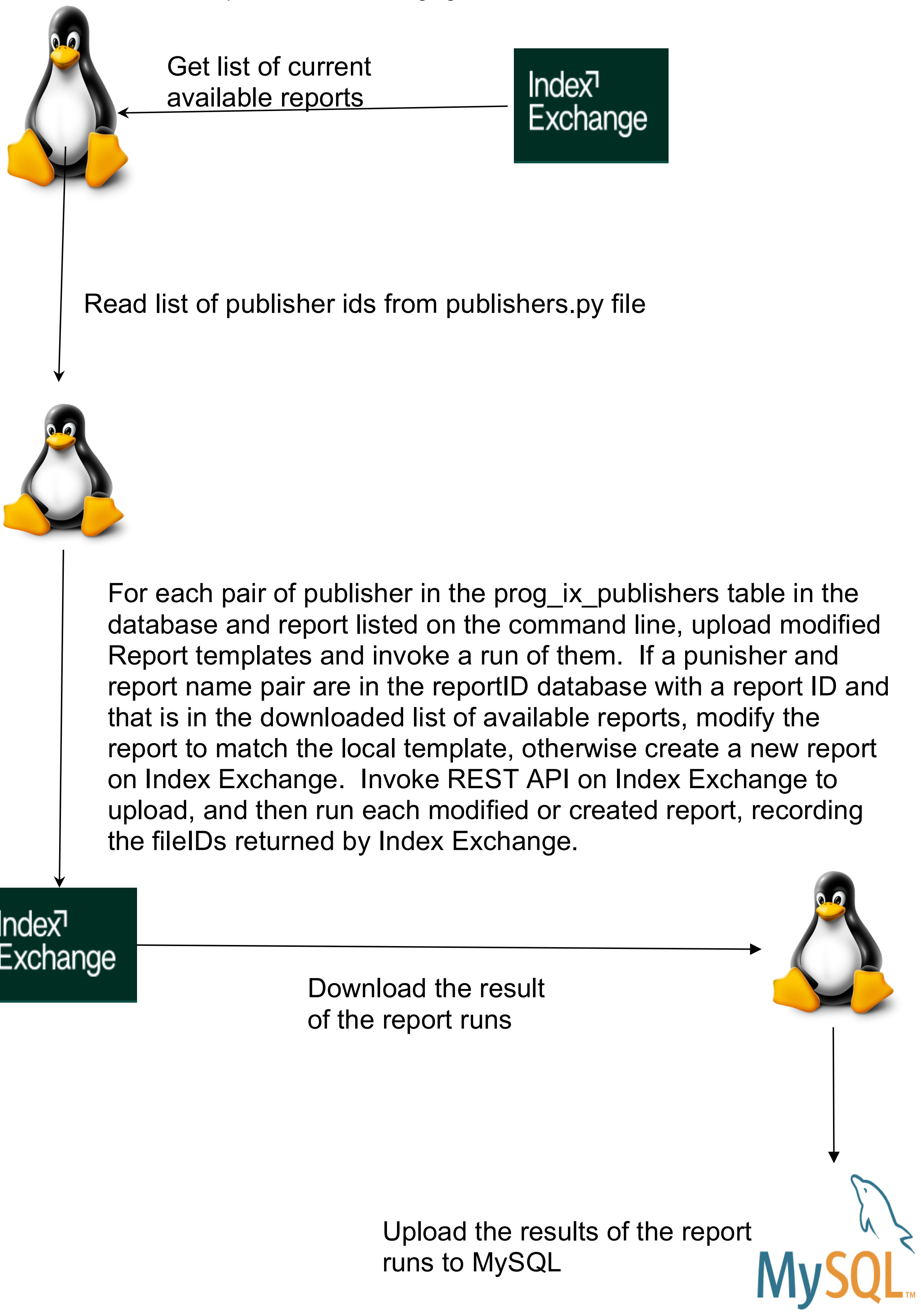
The main program

The suite is operated by running the program index-exchange-to-db.py file. This is also referred to as the *main program*. It's the main program which deals with the data fetched from the reporting API. This was historically the first part of the suite to be implemented. It has subsequently been augmented with other code to use other APIs such as the CAL API.

The main program should be run out of Cron on a regular basis. It is run with the names of the reports that the user desires to run as arguments. In common with Linux syntax thse report names should be quoted if they contain spaces and are listed after any options to the main program.

The publishers from the publishers file that are operated on are by default all of those listed prog_ix_publishers table in the database, but this may be limited with the -l command line option.

The overall data flow is depicted in the following figures.



Get list of current available reports

Read list of publisher ids from publishers.py file

For each pair of publisher in the prog_ix_publishers table in the database and report listed on the command line, upload modified Report templates and invoke a run of them. If a punisher and report name pair are in the reportID database with a report ID and that is in the downloaded list of available reports, modify the report to match the local template, otherwise create a new report on Index Exchange. Invoke REST API on Index Exchange to upload, and then run each modified or created report, recording the fileIDs returned by Index Exchange.

Download the result of the report runs

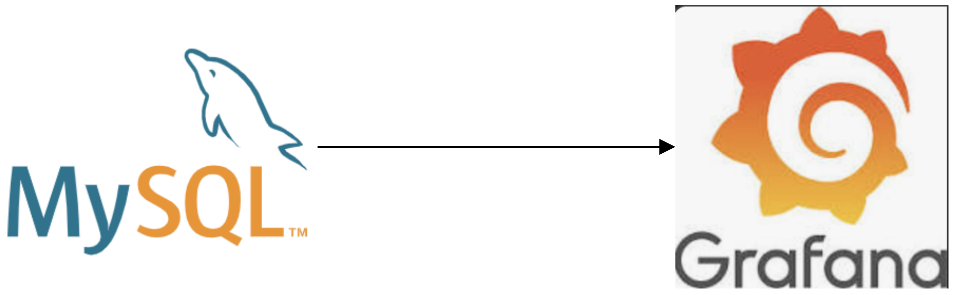
Upload the results of the report runs to MySQL

Utility programs

Detailed below are various utility programs that make operation of the suite easier or provide essential additional functionality. The most significant utility is the get-report-template.py utility which downloads a report to the local template store for use with the suite.

Grafana set up

The Grafana set up is simpler and it simply gets data from the MySQL database as shown in the figure below



Setting up the system

This section explains the various elements needed to set up the system and how to do so from fresh.

Installing the suite from fresh - getting the suite

These instructions assume that the following are already available: the Linux server that the suite will be running on, the MySQL database instance, The account information to access the Index Exchange reports API, and a MySQL user, with appropriate permissions to read, insert, and modify data into the tables.

iximport user on the Linux server

A Linux server and user is needed to run the scripts on the server that runs the suite. Setting up the server is beyond the scope of this document and assumed complete. We start with the creation of the user account used to run the suite.

For this purpose the iximport user should be created. This user can be generated from the root account on the server with the following command

```
useradd -c "Index Exchange Database Import user" -m iximport
```

By default, as set up, this user has a locked password and should be accessed via sudo and su.

The GitHub repository

All code is stored in the GitHub repo at:
`http:///github.com/rob-bannocks/IndexExchange/index-import.git`.
This includes current versions of the the report_to_db_tbale_mapping.py file. The only information not kept here is the authentication information which is kept in the `.env-secrets` file in the user that runs the suite's home directory.

Cloning the repro

Perform the following in the home directory of the iximport user.

```
git clone https://github.com/rob-bannocks/IndexExchange/index-
import.git
```

Prerequisites for the suite

A number of prerequisites are needed to run the suite. Firstly python 3.9 or latter is needed and the following Python modules are needed:

Python module	Needed for

Requests	Make TCP/IP connections for REST calls
Python-dotenv	Process command line arguments for the main and utility programs
Mysql-connector (this itself needs the MySQL client Linux software)	Connect to the MySQL database, received, and send data to it.
mysql-connector-python	Interface MySQL connector with Python
gzip	Used to unzip gzip-ed files sent through the CAL API.
urllib3	Used to manipulate URLs

These can be installed by running the import-setup as a shell script file either in a Python virtual environment or as root on the Linux server which will run the suite. They have been installed as root on the server at Prog Sales. Hence any user can access them.

```
sh import-setup
```

In addition, a number of datastores need to be set up. The two most important have already been mentioned: prog_ix_publishers table in the database and the report_to_db_table_mapping.py file. These contain information that the suite cannot operate without and are maintained manually.

The file report_to_db_table_mapping.py file is part of the GitHub repository. The prog_ix_publishers table is a part of the prog_sales database. The code also automatically maintains the reportID database. This is created automatically if it does not exist.

MySQL Server

A MySQL server is needed to permanently store the data downloaded from Index Exchange. This is currently co-located on the Prog Sales server. Setting up a MySQL server is beyond the scope of this document, however, all the tables should reside in the prog_stats database instance on the server. The tables prog_ix_publishers and the tables mentioned in the report_to_db_table_mapping.py file will also need to be created manually.

Authentication and accounts

Accounts

A user account on the Index Exchange infrastructure is needed to interact with the REST APIs. This has to be obtained from Index Exchange. The storage of this information is detailed in the authentications section.

The MySQL user account

A MySQL user account on the server hosting the MySQL database instance is needed. Because of imitations in the Python mysql-connector framework this user's password cannot be the current default standard_sha2_password type. The password type should be of type mysql_native_password MySQL.

This user needs to have select, insert, modify, and update permissions for each table listed in the report_to_db_tabale_mapping.py file databases associated with the CAL API code and to read and select from the prog_ix_publishers table.

Authentication storage

In order for the main program to operate by default the .env-secrets file must be present in the home directory of the user running the suite. This is kept out of the GitHub Repro.

This file has a simple syntax of

```
PARAMETERNAME=”value”
```

The values that the program reads from this file are given in the table below. For the main program to operate an Index Exchange username and password, and a MySQL username and password are needed. The MySQL host will default to localhost if not specified.

`INDEXEXCHANGEUSERNAME`	Email address of the Index Exchange user whose identity is used to authenticate to Index Exchange for the reports API via the REST API
`INDEXEXCHANGEPASSWORD`	Password for the Index Exchange user whose identity is used to authenticate to Index Exchange for the reports API via the REST API
`PROGSALESMYSQLUSERNAME`	MySQL username to use connecting to the local MySQL instance.
`PROGSALESMYSQLPASSWORD`	Password for the MySQL user to use connecting to the local MySQL instance. Note because of limitations in the mysql-connector this password needs to be an MySQL native password and not the contemporary default of new MySQL users of a sha2 authentication password.
`PROGSALESMYSQLHOSTNAME`	Hostname on which the MySQL database sits. This defaults to localhost if not supplied or present.

This file can be maintained manually or created and maintained by using the create-env-secrets-store.py utility.

Note that for testing purposes these settings can be overridden on a variable by variable basis by setting the shell environment variable of the same name. Hence to set the MySQL username without changing the .env-secrets file you would perform the following shell commands:

```
export PROGSALESMYSQLUSER=freddy
```

And then run the main or utility program of your choice in the same shell. Other shells without this will continue to use information in the .env-settings file.

The prog_ix_publishers table

This is a database table containing an entry for each publisherID that will have the suite download data for. It is a simple data structure containing an integer and a text description of the publisher.

The integer is the Index Exchange defined accountID. The text description is only used in debugging.

The report_to_db_table_mapping.py file

This is a many to one mapping of report names (as found in the templates store - i.e. the filename the report is found in) to database table. All tables are assumed to be in the same database which is defined in the mysql_db.py file in the function get_mysql_db_name to be "prog_stats".

Setting up the authentication
The best way to set up the authentication is to use the create-env-secrest-store.py utility. See further down for a description of this, however, an example of use would be:

```
Python3 create-env-secrest-store.py username@PROGSALES.com INDEX-EXCHANGE-
PASSWORD my_sql-user MYSQL-PASSWORD
```

Downloading reports to the template store

The suite relies on copies of the reports to be run being in the templates store. The name of the report given on the command line is the name of the file containing the report in the templates store.

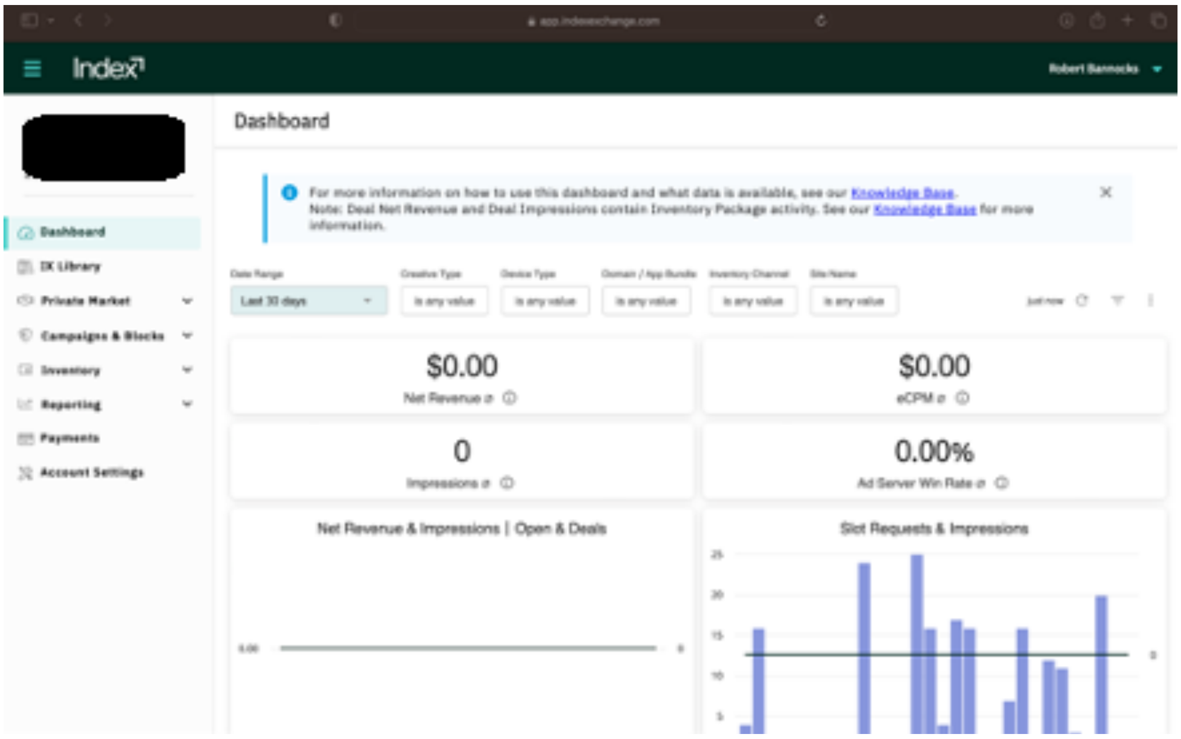
Generating a report

To set up the system from base you should login to the Index Exchange Web GUI. Then select a (any) publisher as shown in the following screenshots:

Login and select publisher

To define a report use any publisher.

After selecting the publisher



[redacted]

Dashboard

For more information on how to use this dashboard and what data is available, see our Knowledge Base.
Note: Deal Net Revenue and Deal Impressions contain Inventory Package activity. See our Knowledge Base for more information.

Date Range: Last 30 days | Creative Type: is any value | Device Type: is any value | Domain / App Bundle: is any value | Inventory Channel: is any value | Site Name: is any value

$0.00 Net Revenue

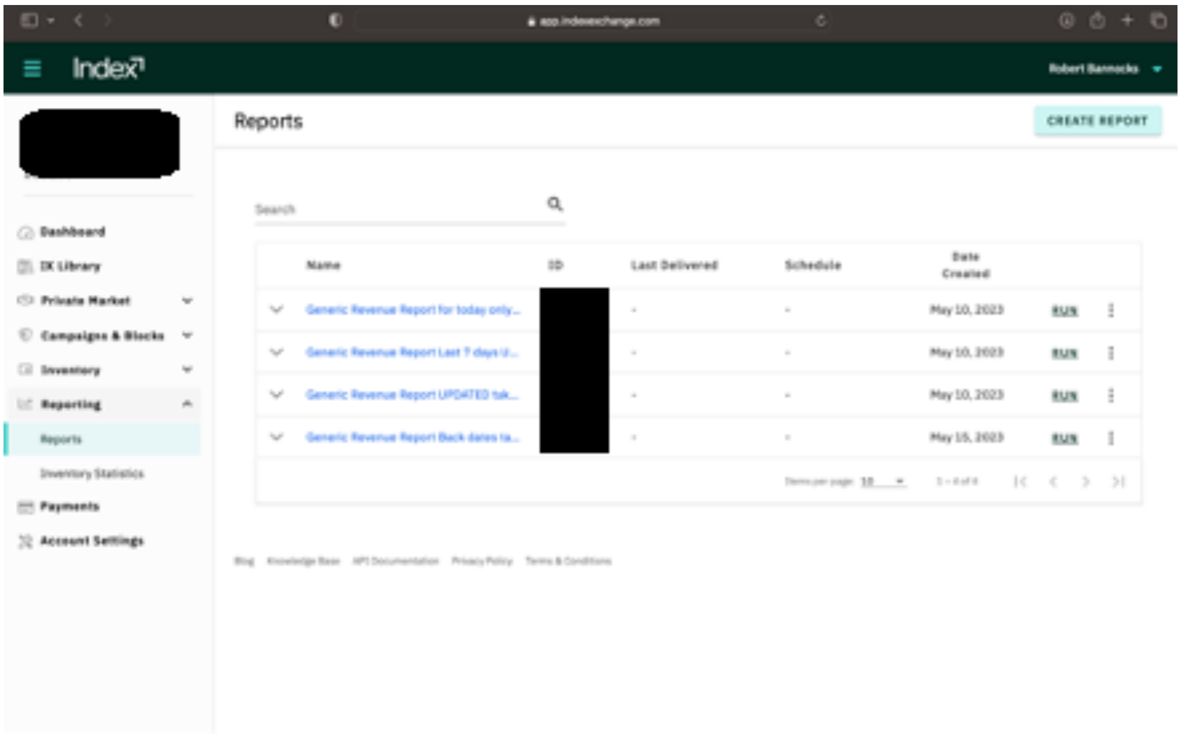
$0.00 eCPM

0 Impressions

0.00% Ad Server Win Rate

The from the left-hand menu select the reporting and within that select reports
As shown in the following screenshots

Selecting reports

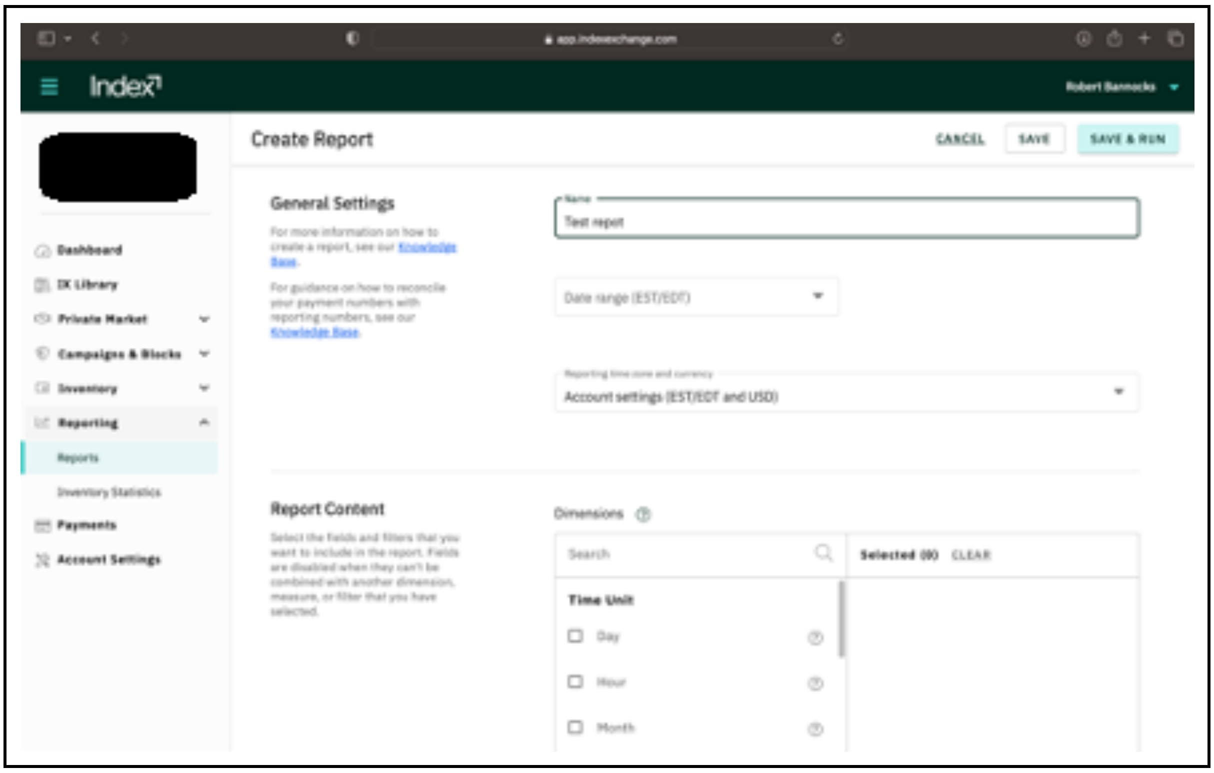


[redacted]

Reports

Name	ID	Last Delivered	Schedule	Date Created	
Generic Revenue Report for today only...	[redacted]	-	-	May 10, 2023	RUN
Generic Revenue Report Last 7 days U...	[redacted]	-	-	May 10, 2023	RUN
Generic Revenue Report UPDATED tak...	[redacted]	-	-	May 10, 2023	RUN
Generic Revenue Report Back dates ta...	[redacted]	-	-	May 15, 2023	RUN

Now select define a new report by selecting “create report”


app.indexexchange.com

Index

Robert Bannocks

[redacted]

Dashboard
IX Library
Private Market
Campaigns & Blocks
Inventory
Reporting
Reports
Inventory Statistics
Payments
Account Settings

Create Report

CANCEL SAVE SAVE & RUN

General Settings

For more information on how to create a report, see our Knowledge Base.

For guidance on how to reconcile your payment numbers with reporting numbers, see our Knowledge Base.

Name: Test report

Date range (EST/EDT)

Reporting time zone and currency: Account settings (EST/EDT and USD)

Report Content

Select the fields and filters that you want to include in the report. Fields are disabled when they can't be combined with another dimension, measure, or filter that you have selected.

Dimensions

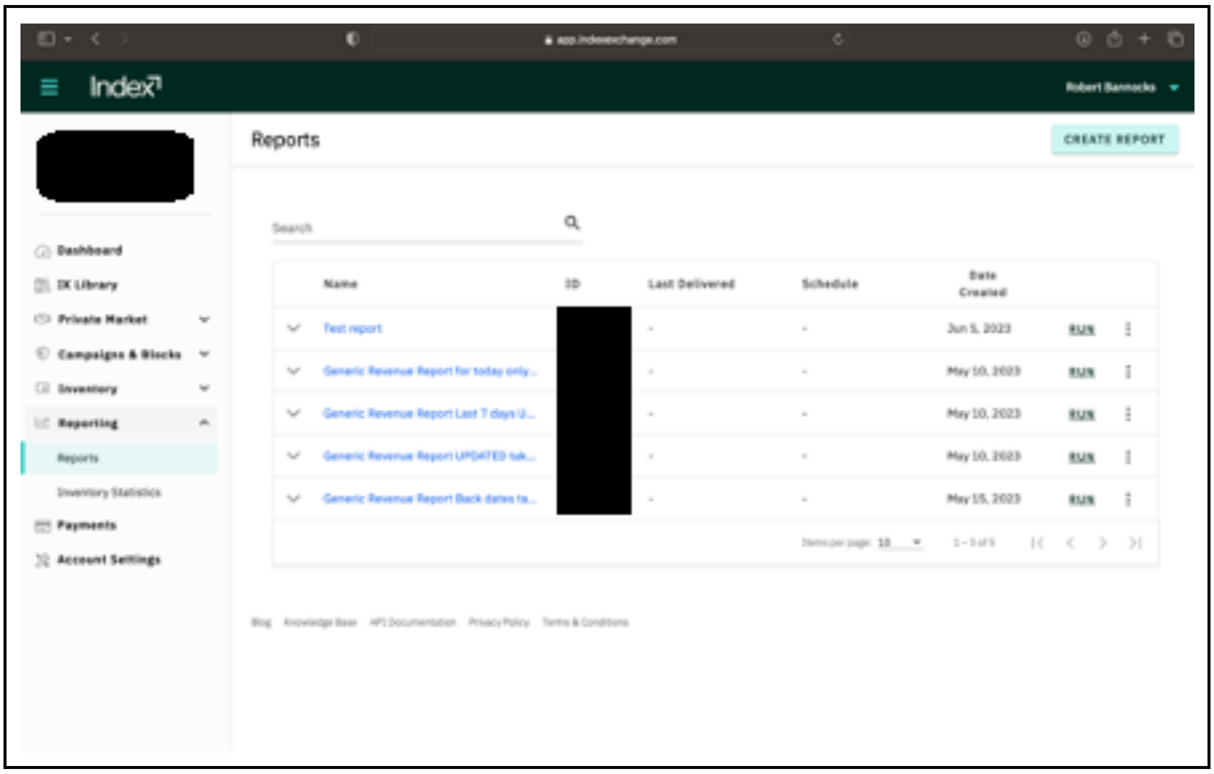
Search

Selected (0) CLEAR

Time Unit
- [ ] Day
- [ ] Hour
- [ ] Month

Give the report a descriptive name as this will be the name of the file that the report is saved in in the templates store and used on the command line to effect its running and resultant data collection. Also select the metrics you want to download in the report. For each metric there must be a column in the database table that this report will generate data for. You will also need to set the date range the report will run for.

Save the report by clicking the save box and make a note of the report ID number shown in the list of reports (under the ID heading):



Reports

Name	ID	Last Delivered	Schedule	Date Created		
Test report	[redacted]	-	-	Jun 5, 2023	RUN	⋮
Generic Revenue Report for today only...	[redacted]	-	-	May 10, 2023	RUN	⋮
Generic Revenue Report Last 7 days U...	[redacted]	-	-	May 10, 2023	RUN	⋮
Generic Revenue Report UPDATED tak...	[redacted]	-	-	May 10, 2023	RUN	⋮
Generic Revenue Report Back dates ta...	[redacted]	-	-	May 15, 2023	RUN	⋮

Items per page: 10 1 – 5 of 5

Now use the get-a-report-template.py file to download the report to the local store.

```
python3 get-a-template-report.py 1965885
```

Where you replace `1965885` with the actual report number noted from the ID column in the previous screen.

Repeat this for each distinct report you want to get the suite to run and download the data from.

Next you need to add new reports to the report_to_db_table_mapping.py file and define the database table in the MySQL database that the data from the/these reports will be placed in.

The report_to_db_table_mapping.py file

This is a python file which contains a dictionary which in a many to one map maps the data resulting from a given report name (i.e. the filename of the report stored in the templates store) to the database table that the resultant information is stored in.
The dictionary consists of a key of the report name and a value of a dictionary which specifies the database table and a reportID. The following is and example element of this dictionary:

```
'Report Title': { db_table: 'database table name', ' reportID': 1234}
```

As described elsewhere the report title is the filename of the report template in the templates store.

Cron Jobs

Now the reports are downloaded to the templates store and the system set up you need to schedule runs of the suite from Cron. This is simply done by using Cron to select how often to run the reports and invoking Python with the name of the main program (index-exchange-to-db.py and the reports you wish the main program to process. Note that because Python lacks a local database that can handle simultaneous access from multiple processes only one instance of the main program should be scheduled to run at a time.

The current list of Cron reports is shown below.

Cron Jobs

The `iximport` user currently has the following Cron jobs to run the suite for various reports:

```
0 8,11,20 * * 2,4,5 /usr/bin/python3 index-import/index-exchange-to-db.py "Generic Revenue
Report" "Generic Revenue Report for today only"
# on Wednesdays send activity report for daily download but not evening retry
0 8 * * 3 /usr/bin/python3 index-import/index-exchange-to-db.py -R "Generic Revenue Report"
"Generic Revenue Report for today only"
0 11,20 * * 3 /usr/bin/python3 index-import/index-exchange-to-db.py "Generic Revenue
Report" "Generic Revenue Report for today only"
# on monday catch up from the weekend
0 8,11,20 * * 1 /usr/bin/python3 index-import/index-exchange-to-db.py "Generic Revenue
Report Last 7 days" "Generic Revenue Report for today only"
# daily incremental
0 6-7,9-10,12-16 * * 1,2,3,4,5 /usr/bin/python3 index-import/index-exchange-to-db.py
"Generic Revenue Report for today only"
# Download the log and mapping info.
# impressions and logs
0 * * * * /usr/bin/python3 index-import/cal-downloads.py -M -l
# mapping data
0 6 * * * /usr/bin/python3 index-import/cal-downloads.py
```

A Grafana instance

Although external to the suite the data in the database is visualised with Grafana. This also is currently on port 3000 of Prog Sales server. Setting up Grafana is beyond the scope of this document.

Terminology

The main program

The main program, the index-exchange-to-db.py program, is referred to as the main program and is discussed below.

Data stores

A number of file locations are defined as data stores within the suite. These are defined and explained in the table below. They are referred to as stores of one kind or another. They are realised as files within a directory. The name of the files and location in the file system of the directories is defined in the settings.py file. In addition, as already mentioned there is one important file maintained as part of the Github repo the report_to_db_table_mapping.py file, detailed above.

Stores

There are a number of stores of information called Stores. These are all kept in files wich are located within subdirectories of the user running the main program's home directory. Their location is defined in the settings.py file.

The data stores are created automatically by the functions that access them (which are used by the main program and utility programs). Note however the main program cannot run until the report templates listed on the command line have been downloaded from Index Exchange by the get-report-template.py utility (to the template store and the report_to_db_table_mapping.py file created).

store	purpose	Default location. Which is the subdirectory, listed below, of the home directory of the user running the main program	Function in settings which determines this
Data download store	Hold csv files resulting from run of reports. Only used if -n option given on command line.	INDEX-DATA-DOWNLOAD	
Templates store	Used to hold the templates of report updated or uploaded to Index Exchange	INDEX-TEMPLATES	
Settings store	Used to store the reportID database	INDEX-SETTINGS	
Report runs download queue	Not yet implemented		Meant for reports which were unable to be submitted and/or run (because Index Exchange refused them) and hence run at a latter occasion
Reports to be run download queue	Not yet implemented		Meant for a store of report IDs, publisher IDs and report names for reports which could not be downloaded on the

			last run of the main program (e.g. due to Index Exchange reporting them as not available)

Database tables

The suite relies on the prog_ix_publisher database table in addition to the tables listed in the report_to_db_table_mapping.py file. This can be created in the MySQL client as follows.

```
CREATE TABLE `prog_ix_publishers` (
  `publisher_id` int NOT NULL,
  `publisher` varchar(64) DEFAULT NULL,
  `country` varchar(2) DEFAULT '',
  PRIMARY KEY (`publisher_id`)
```

As discussed elsewhere this needs to be manually populated.

Details of current reports used

The following reports as of the writing are defined and downloaded to the local template store.

Report name	Purpose	comments
The Generic Revenue Report	Get revenue for the previous day. Writes to prog_ix_stats database.	Because of timezone issues must be run after 9am UTC
The Generic Revenue Report Last 7 days	Revenue report for the last 7 days Writes to prog_ix_stats database.	Run on Mondays to catch up for the weekend. The same comment about 9am UTC above applies.
The Generic Revenue Report for today only	See Index Exchange documentation. May contain incomplete data. Writes to prog_ix_stats database. Run each hour. This overwrites the previous run within a calendar (or UTC) day.	Run hourly, subsequent runs overwrite data previously downloaded
Generic Revenue Report Back dates	Used to bootstrap the database with data back to the 1 January of the year.	

	Only used to bootstrap the database with data.	

General notes about reports uploaded or updated

Note that the main program modifies these properties of the report downloaded on each update:

Report Title

The name to be of a standard form based on the local name of the report as picked from the local templated director

```
<report Name> taken from Prog API Tempoary AutoUploaded report by
<authusername> <time>
```

Whew the angle brackets denote text that depends on the variable in the angle brackets. Updated reports have “taken from” preceded by “UPDATED”

Report format

All report have the format similarly forced to “csv.txt” format. I.e. not compressed.

Timezone

The regionSetting is forcibly set to “standard” to make all time reporting in UTC.

AccountID

The account id that report is run for. (The main program cycles through the reports given on the command line submitting one report and run for each combination of report on the command line and publisher_id in the rog_ix_publishers table (or list given with -l if specified)).

Getting a Summary of the main programs actviviy

If the main program is invoked with the -R flag a short report of activity is produced at the end of its run.

Utility programs

These utility programs are provided to interact with the system and Index Exchange.

The get-report-template.py utility

This program takes one or more reportIDs and downloads the report specification as a JSON file which it places in a file name of the respective reports title in the templates store. This allows reports to be specified on the Index Exchange server and downloaded for use on the suite. See above for information on how the reports are modified when uploaded or updated and how to specify where the output of the report is placed in the MySQL data base.

The create-env-secrets-file.py utility

This program creates a .env-secrets file in the Linux user's home directory of the user running the utility. The values of the variables in this file are detailed above. The program takes up to 5 arguments.

```
create-env-secrets-file.py Index-Echange-username index-eschange-
password mysql-username mysql-password mysql-hostname
```

All arguments are optional. If none are provided the program prints the current content of the .env-secrets file.

The get-a-report-run.py utility

This utility will download the results of a report run from Index Exchange. These will be written to a file with the reportID name in the download store (this can be changed with the -d option). As the results of the report runs do not contain data as to the publisher that the data refers to the publisherid can be specified with the -p option. If not supplied the publisherid will default to 0. Only fileIDs that are available from Index Exchange will be downloaded. FileIDs that are not available will be ignored.

The ix-reportid-db-to-json utility

This utility will convert the binary shelve reportID database to JSON format, printing it to Standard out. The ReportID database is held in shelve format which is a binary format and this utility is provided to allow human inspection of the contents.

The json-to-reportid-db utility

This utility program likewise will take as the first command line argument a JSON file and write it to the reportID database in shelve formal

Utility program and options

The below table lists the utility programs and their command line options.

Note all commands support the -h, or -help command line options which prints a short help text and the -D option which turns on verbose debugging.

Program name	purpose	Command line parameters	comments
get-a report-run-py	Download results of report runs from Index Exchange. These are deposited in the data download stores	[-p publisher] Specify the publisher to which the report related [-j] [-s status] Select which status of report to download. Omitting leads to all. Status can take values “new” and “downloaded” only [- d directory] Directory to which to download reports, overrides the default of the data download store fileID [fileID] The fileIDs from index exchange to download	Index Exchange to do nott retain the publisher to which a report run related in the returned CSV file. Hence the need for the -p option
get-report-template.py	Will download an existing template report from Index exchange. This means you can define the report in the Index Exchange GUI and then download it for	Followed by a space separated list of reportIDs to download	Reports are downloaded to the template store and stored in JSON in a file name of the report title. Change the name used by using the Linux command mv command in

	use with this suite of programs. See above for usage details		template store’s directory.
ix-reportid-db-to-json utility	This utility will convert the binary reportID database to JSON format. The reportID database is held in shelve format which is a binary format and this utility is provided to allow human inspection of the contents.		
json-to-reportid-db utility	This will take a JSON definition of the reportID database and write it as a shelve database in the settings store.		

The index-exchange-to-db.py program, the main program

This is the central program in the suite which manages the upload or update of template reports to the Index Exchange servers, invoking a run of those reports, collecting the resulting data from Index Exchange and re-formatting the data and inserting it into the local MySQL database used by Grafana. This is also known as the main program. It is highly structured and uses functions from most all of the other Python files in the suite with the exception of the utility programs.

Options for the main program

Option	Meaning and effect
-R	Produce a short report on activity at the end of the program's run.
-n	write downloaded data to the data download store as well as updating the MySQL database. Data is save in filenames that are the fileID of the downloaded data.
-I ID [,ID]	Option is followers by a comma separated list of publisherIDs. Limit the publisherIDs that the program uploads/updates the reports for to the list, in comma separated form, following the -I option.
-d	Directory in which files created with the -n option are written to. -n uses the data download store by default.
-D	Produce voluminous debugging output.
-r <n>	After the reports are uploaded or updated and run, the files that contain the data resulting from these reports are downloaded. There can be a delay in the availability of such information and the program retries to download the report results from Index Exchange again by sleeping for 10 seconds and then retrying those reports which have not yet been downloaded. The program does this by default for 15 times, this number can be changed by following the -r option with the number of retries desired.

Catalogue of files in the code

See the CAL API documentation for a list of files used by the CAL API code.

File	purpose
`authenticationAPI.py`	Direct implementation of REST APIs for the report Authentication
`authenticationSupport.py`	Support functions that build on and make easier to use authentication API functions
`create-env-secrest-store.py`	Utility to set up report, MySQL, and CAL API authentication secrets.
`deactivate-reportid.py`	Script to deactivate (read delete) reports on the Index Exchange infrastructure
`get-a-report-run.py`	Utility to fetch the result of a report run. NB as mentioned elsewhere the results of report runs do not store the publisher id so use the -p option to set the publisherID in the output file.
`get-list-avaliable-files.py`	Utility to get a list of fileIDs which can be downloaded.
`get-report-template.py`	Utility to download a reportID. Takes a reportID taken from the web GUI and (by default) stores the report in the templates store.
`import-setup`	Shell script to install needed Python modules.
`index-exchange-to-db.py`	Main program which uploads reports, runs them, fetches the results and stores the resultant data in the MySQL database. Uses almost all other files here except the utility programs and the CAL API files. AKA the main program.
`indexExchangeToDBSupport.py`	Contains the support functions and main code used in the index-to-exchange-to-db.py file
`ix-reportid-db-to-json.py`	Utility program to dump the reportID database to a JSON file
`json-to-ix-reportid-db.py`	Utility takes a JSON file and imports it into the reportID database..
`mysql_db.py`	Functions to interact with the MySQL database.
`readwrite.py`	Functions to read and write files from and to local file store.
`report_to_db_table_mapping.py`	A Python file with a Python dictionary which tells

	the index-exchange-to-db.py file which MySQL database table to put the result of each named report run.
`reportdownloadAPI.py`	Direct API functions to use the REST APIs for the download reports API section of the reports API.
`reportdownloadSupport.py`	Functions which build on the raw API functions in the reportdownloadAPI.py functions.
`reportdownloadWrapper.py`	Functions which build further on the functions in the reportdownloadSupport.py functions.
`reportmanagementAPI.py`	Direct API functions to use the REST APIs for the report management API section of the reports API.
`reportmanagementSupport.py`	Functions which build on the raw API functions in the reportmanagementAPI.py functions.
`settings.py`	Python file with functions which provide the settings and default settings used by other functions, e.g. the file location of the various stores and other configuration files.
`tidy-report.py`	Report to remove all autogenerated reports that are not in the local reportID database.
`urlencoding.py`	Support functions to transform data into a form unable over HTTP.
`utility.py`	Utility functions to support all other files.

See the CAL documentation for details of files used by the CAL download code.