Qualcomm Technologies, Inc.

DIRBS Core Release 8.0.0

Release Notes

70-GD079-15 Rev. 8.0.0

April 24, 2018

Revision history

Revision	Date	Description
8.0.0	April 2018	Initial release

Contents

Tables

1 DIRBS Core Release 8.0.0

1.1 In this release

Error! Reference source not found. lists additional assets that form the DIRBS Core Release 8.0.0.

Table 1-1 DIRBS Core Release 8.0.0 assets

Asset	Function
dirbs-8.0.0-py3-none-any.whl	File containing packaged form of the DIRBS Core code
docker/	Directory containing support files required to build and run sample DIRBS Docker containers
etc/	Directory containing files that should be copied to /opt/dirbs/etc (see Chapter 2), including: ▪ CSV validation schema (in etc/schema/) ▪ Sample crontabs for scheduling imports and other jobs (in etc/crontabs) ▪ Makefiles that can be used to monitor directories for new operator data, GSMA TAC DB data, registration list data, or golden list data ▪ Sample configuration file (config.yml, see Chapter 3) that can be modified to suit the DIRBS deployment
opensource_requirements.txt	File listing open source software dependencies to install
test_requirements.txt	File listing open source software dependencies to install the automated suite
tests/	Directory containing Python test scripts and test data files required to run the automated regression test suite

1.2 Changes since 7.0.0 release

1.2.1 Breaking changes

Database

We now require the postgresql-hll PostgreSQL extension to be installed on the PostgreSQL server used with DIRBS Core. Without this extension installed in the database, `dirbs-db upgrade` will fail to run.

Before database installation/migration can proceed, the following must be run as DB superuser:

```
CREATE SCHEMA hll;
GRANT USAGE ON SCHEMA hll TO dirbs_core_base;
CREATE EXTENSION hll SCHEMA hll;
```

PostgreSQL 10 is required at minimum; otherwise, DIRBS Core will not run.

Due to restructuring the database schema, many tables have changed names in this release:

- **seen_imeis (now named network_imeis)**: This is no longer stored on a per-operator basis and is a country-wide listing of all IMEIs ever observed on any network. Column names are unchanged besides removal of operator_id and first_seen_import_id.
- **seen_triplets (now named monthly_network_triplets_per_mno):** Import ID bitmasks have been removed since these increase data size dramatically for daily imports and the use case for them was unclear. Otherwise, this table should function as seen_triplets in 7.0.0.

Reporting

- The definition of a gross-add IMEI has changed. Previously, an IMEI first observed on an MNO network during a month was reported as a gross-add. Now, it is only reported as a gross-add if it was the first month that the IMEI was seen on *any* network
- The blacklist violations report was removed since it was incorrect in the case of unpairing. A new spec for counting/listing blacklist violations has been generated but not implemented.

Additional changes

- **Registration list importer:** New columns are now required in registration list imports (make, model, and status). A new pre-validation schema has been provided. These can be left empty, but columns must be present.
- **Stolen list importer:** Status is a new column that is now required in stolen list imports. This can be left empty, but the column must be present.

1.2.2 New features

Operator data importer

During import, daily stats for the MNO corresponding to data in the import are calculated:

- Distinct IMEI count
- Distinct IMSI count
- Distinct MSISDN count
- Distinct Triplet (IMEI, IMSI, MSISDN) count
- Distinct IMEI-IMSI count
- Distinct IMEI-MSISDN count
- Distinct IMSI-MSISDN count

These counts are not stored as simple integers, but instead as HyperLogLog (HLL) sketches, e.g., distinct counts over any time period can be generated in real-time, but with some error.

HLL is a fixed-size, set-like structure used for distinct value counting with tunable precision.

For example, in 1280 bytes, HLL can estimate the count of tens of billions of distinct values with relative error given by the expression $\pm 1.04/\sqrt{(2_{\log 2m})}$.

All

Each IMEI is allocated to a "virtual" IMEI shard, running from 0-99. We then allocate these virtual shards to physical partitions. The default number of physical shards is 4, meaning that each physical shard contains roughly 25 virtual shards. This can be changed using the new `dirbs-db repartition` command.

These tables are now sharded/partitioned based on the 13th and 14th digits of their IMEI:

- classification_state
- historic_pairing_list
- historic_stolen_list
- historic_registration_list
- blacklist
- exceptions_lists
- notifications_list
- network_imeis
- monthly_network_triplets_per_mno
- monthly_network_triplets_country

After data has been sharded by IMEI, we can perform many operations on a per-shard basis since we know that there are no interactions between shards. This parallelization was done for the following:

- All importers (except for GSMA and golden list since they are not sharded)
- Calculation matching IMEIs during classification and storage of those IMEIs in the classification_state table
- List generation calculation of triplets to notify

To take advantage of this, we have bumped the default value of `max-db-connections` from 4 to 8. There are no longer any problems raising this to 16 or 32.

At some point, the bottleneck might become the I/O performance of the hardware, especially during operations that write large amounts of data (like importing) and on rotational disks (non-SSD).

Additional new features

- **Reporting:** The standard report (`dirbs-report standard`) was re-worked to take advantage of the HLL feature when calculating distinct identifier counts over the month or for daily counts. These can now be calculated in milliseconds versus hours.
- **Classification:** DIRBS Core now provides a native solution for the grandfathering/amnesty of IMEIs. A regulator can configure an amnesty evaluation period for:
  - Seeding the list of IMEIs to grant amnesty to
  - An amnesty period
  - Classification conditions eligible for amnesty

  IMEIs will not be blacklisted until the amnesty period is over.

- **API documentation:** Swagger/open API documentation is now provided for DIRBS Core and available at `/apidocs/v1/` on the API blade image.
- **Classification/API:** Classification and APIs now support the concept of "provisional compliance" for the stolen list and registration list.
  - Allows the DVS to return a "negative" result for IMEIs that have been reported lost, but we do not yet want to blacklist. Likewise, a device that is registered pending approval can potentially also received a different status.
  - Feature relies on extra status data provided in the status column of registration and stolen list imports.
- **Operator importer:** Since classification operates at a country-level, we now store a copy of network data pre-aggregated to the country level.
  - Requires approximately twice the amount of disk storage for the DB, but the payoff is that we no longer need to group/aggregate to find distinct triplets in a given month.
  - Also speeds up country-level reporting since we already have pre-aggregated data containing unique triplets at a country level.
- **Classification/list generation:** If `dirbs-classify` and `dirbs-listgen` are run in verbose, the exact window used for classification or list generation will be printed in the console output.

1.2.3 Enhancements

- **PostgreSQL config:** Updated recommendations provided for PostgreSQL configuration settings.
- **API:** Database schema checks are now only made when the Core API is first started up to improve maximum loaded performance by ~10%.
  - Once a successful request has been served, this check is skipped. This is intended to be migrated to the new Health Check API proposed as part of the V2 API specification.

1.2.4 Bug fixes

- **Reporting:** Fixed out of memory issue encountered in standard report when running a report on partition with approximately one billion distinct IMEI/IMSI/MSISDN triplets.
- **Classification:** Fixed bug where there `dirbs-classify` took a long time counting distinct IMEIs in seen_imeis if the safety check is enabled.
- **GSMA TAC DB importer:** Removed artificial limits to data input in GSMA TAC DB. This was preventing a recent GSMA TAC DB from being imported as the bands column was more than 4096 characters long.
  - Now, all database columns are TEXT typed (except for TAC, which is still enforced to be less than 8 characters) and the pre-validator schema has been updated to also reflect this.
- **All importers:** Fixed bug where the configured historic thresholds contained in the config .yml were silently ignored and defaults were used instead.
- **Reporting:** Fixed crash in standard report generation when there were no classification conditions configured

1.3 Known issues

- Some settings in the suppled .yml config have changed in this release. Ensure that you have updated your config based on the provided sample config in the 7.0.0 release.
- Some recommendations for PostgreSQL config have changed in this release:
  - Check memory recommendations contained in the supplied sample PostgreSQL config. To avoid encountering OOM errors in PostgreSQL due to memory overcommit in Linux, we recommend modifying kernel parameters to avoid this.

    For more information, see: https://www.postgresql.org/docs/10/static/kernel-resources.html#LINUX-MEMORY-OVERCOMMIT
- Given the large database schema changes in this release, `dirbs-db upgrade` can take a very long time.

  On a sample five-month data set containing 370 million entries in the `seen_triplets` table, the upgrade takes about 12 hours on our test hardware. If more than five months is stored or if each month contains more data, we expect this time to increase in a linear fashion.

  It is essential that the upgrade is atomic to avoid partially upgraded databases that cannot be fixed. Therefore, the entire upgrade is performed in a single transaction, e.g., some exclusive locks are held for a long time. This upgrade should only be done on an offline copy of the database.
- There are no hard and fast rules for choosing the number of physical shards to use in a deployment.

  Factors affecting this choice:
  - Data size
  - Number of connections/CPUs available on the database server
  - Disk type on the database server

  In general, the aim was to exploit as much parallelism as possible on the database server. The default setting of 4 should work reasonably well for most deployments. This can be increased to a higher value for higher performance – especially if the deployment has large data, SSD disks, and is running a higher value than 8 for `max_db_connections`.

  Ideally, this value should be picked immediately after installation of the schema, as it is quick to change for an empty database, but can also be changed at a date using `dirbs-db repartition`.
- In this release, we store two copies of network triplets data (country and per-MNO level).
  - Previous releases: We only stored data at a per-MNO level. This offers performance improvements for components like classifications that only care about data at a country level (country-level reports also benefit from this).
  - The downside is that we are storing two copies of the same data, so disk usage requirements increased. In both cases, we stored data in tables with a fill factor of 45 so that HOT updates could be made in the most efficient way and indices were not fragmented.
    - All tables now take up twice as much space and twice as much I/O to scan. This was true in all DIRBS Core releases from 4.0.0 onwards.

- We generally do not need this extra space for any data except for the current month, so `dirbs-db repartition` will "compact" data that is not the latest year and month encountered so that it is tightly packed (fill factor of 100). This is twice the savings in disk storage for all but the current month. Therefore, we do not expect overall disk usage requirements to increase much as long as an occasional `dirbs-db repartition` is performed to re-write tables (this is good for performance as well). This can be done with the same number of physical shards to just do the re-pack.

- Resolved
  - In previous versions of the release notes, we recommended taking care when setting `max_db_connections` during `dirbs-report` or `dirbs-classify`. Due to improvements made in this release, namely PostgreSQL config updates as well as code changes, we now recommend setting this to a high value (even the default value has changed from 4 to 8). You will likely find that that a value of 16 or 32 will yield higher performance, especially if the number of partitions is greater than 4.

2 Installation

This installation guide builds a new Docker image by downloading a base Ubuntu 16.04 image and downloading and installing required third party software modules into the image as a convenience to the end user.

NOTE: The reader acknowledges and agrees that it is entirely and solely responsible for the selection and use of all third-party software modules downloaded and installed by this installation method, including securing all appropriate and proper rights of use to any of such third-party software modules and to comply fully with any terms of use that may apply to or accompany any such third-party software modules.

Qualcomm Technologies, Inc. does not undertake any obligations, duties, or other responsibilities in connection with the selection or use by the reader of any of such third-party software modules.

2.1 Docker installation

Docker installation can be run on any platform Docker supports.

For instructions on installing Docker on a particular host platform, see https://docs.docker.com/engine/installation

2.1.1 Building Docker images

Assuming that you are in the root folder for the distributables, run the following command to build all sample Docker images:

```
make -f docker/prd/Makefile
```

2.1.2 Running Docker images

This section shows basic commands used to run each type of container, assuming the config file contained all settings at the time the config file was built.

2.1.2.1 Data Processing blade image

After Docker images have been built, start the Data Processing blade image by running

```
docker run --rm --tmpfs /tmp -e DIRBS_OPERATORS=<operator_list> \
   -p 2222:22 dirbs-processing
```

This `docker run` command uses the following standard Docker options:

- `--rm`: Instructs Docker to remove the container from the system once it has been stopped.
- `-e`: Set an environment variable inside the container.
- `-p 2222:22`: Maps the SSH port (22) inside the container to port 2222 on the host system.
- `--tmpfs /tmp`: Instructs docker to use an in-memory filesystem for the `/tmp` directory for performance reasons.

2.1.2.2 Operator Upload blade image

After Docker images have been built, start the Operator Upload blade image by running
`docker run --rm --tmpfs /tmp -e DIRBS_OPERATORS=<operator_list> -p 2222:22 dirbs-upload`

This `docker run` command uses the following standard Docker options:

- `--rm`: Instructs Docker to remove the container from the system once it has been stopped.
- `-e`: Set an environment variable inside the container.
- `-p 2222:22`: Maps SSH port (22) inside the container to port 2222 on the host system.
- `--tmpfs /tmp`: Instructs docker to use an in-memory filesystem for the `/tmp` directory for performance reasons.

`<operator_list>` should be a comma-separated list of operator IDs that match the list of operators configured in the config file.

2.1.2.3 API blade image

After Docker images have been built, start the API blade image by running
`docker run --rm --tmpfs /tmp -p 5000:5000 dirbs-api`

- `--rm`: Instructs Docker to remove the container from the system once it has been stopped.
- `-p 5000:5000`: Maps webserver port (5000) inside the container to port 5000 on the host system.
- `--tmpfs /tmp`: Instructs docker to use an in-memory filesystem for the `/tmp` directory for performance reasons.

2.2 Advanced topics

2.2.1 API and Data Processing blades

2.2.1.1 Mounting a config file inside a container

To mount a config file from the host into the Data Processing and API blade containers so that DIRBS configuration can be modified after build time, use the following option to the `docker run` command for those images:

`-v <abs_path_to_config_on_host.yml>:/home/dirbs/.dirbs.yml:ro`

This option mounts the file located at `abs_path_to_config_on_host.yml` to `/home/dirbs/.dirbs.yml` inside the container. This will be read-only inside the container. Changes made on the host will be reflected inside the container.

2.2.1.2 Mounting a `.pgpass` file inside a container

While database credentials can be edited directly inside the `/home/dirbs/.dirbs.yml` file, a `.pgpass` file (standard PostgreSQL connection string file) can be mounted inside the container as well.

Advantages are:

- The person responsible for editing the config YML does not automatically get access to DB credentials.
- A `.pgpass` file can configure a password for multiple users, not just one.

Use the following option to the `docker run` command for the Data Processing and API blade containers:

```
-v <abs_pgpass_file_on_host>:/home/dirbs/.pgpass:ro
```

This option mounts the file located at `abs_pgpass_file_on_host.yml` to `/home/dirbs/.pgpass` inside the container. This will be read-only inside the container.

NOTE: The `.pgpass` file will be ignored if UNIX file permissions are not stricter than 0700.

2.2.1.3 Passing in environment variables for StatsD and PostgreSQL connection details

If set, the following environment variables override configuration file settings. If there is a command-line argument specified, this continues to override environment variables:

- `DIRBS_DB_HOST`: Host that PostgreSQL database runs on (default: localhost)
- `DIRBS_DB_PORT`: Port that PostgreSQL database runs on (default: 5432)
- `DIRBS_DB_DATABASE`: PostgreSQL database name to connect to
- `DIRBS_DB_USER`: PostgreSQL user to connect as
- `DIRBS_DB_PASSWORD`: PostgreSQL password for `DIRBS_DB_USER`
- `DIRBS_STATSD_HOST`: Host that StatsD run on (default: localhost)
- `DIRBS_STATSD_PORT`: Port that StatsD listens on (default: 8125)
- `DIRBS_ENV`: Unique environment string used by StatsD to distinguish between different hosts or environments when sending metrics

The values of these environment variables can be passed through to Docker containers in two ways:

- Populate a file with a list of the above `KEY=VALUE` lines
  - This file can then be passed to the `docker run` command via the `--env-file` command-line option
- Specify each key-value pair on the `docker run` command-line using the `-e` command-line option

2.2.1.4 Mounting a persistent log directory

To ensure that logs are persisted between container restarts, the following CLI option should be provided to `docker run`:

```
-v <abs_log_dir_on_host>:/var/log/dirbs
```

This option mounts the host directory located at `abs_log_dir_on_host` to `/var/log/dirbs` in the container.

2.2.2 Operator Upload and Data Processing blades

2.2.2.1 Mounting a persistent data directory

For Operator Upload and Data Processing blades, the data directory should be persistent between container restarts.

To ensure this is the case, a host directory can be mounted into the container using the following CLI option to `docker run`:

```
-v <abs_data_dir_on_host>:/data
```

This option mounts the host directory located at `abs_data_dir_on_host` to `/data` in the container.

2.3 Database installation

2.3.1 Installing a new database

NOTE: Creating a new database from scratch assumes that you are already running a PostgreSQL instance.

1. Install the PostgreSQL HLL extension

DIRBS Core requires the installation of the PostgreSQL HLL extension to function (https://github.com/citusdata/postgresql-hll).

If using our provided Docker image for PostgreSQL, it is already installed.

If on RDS in AWS, this extension should also be optionally available.

Otherwise, you will likely need to build and install the extension. Consult README in the linked GitHub repo for instructions on how to do this.

2. Creating a PostgreSQL role with `superuser` privileges

If you are using the provided Docker image for PostgreSQL, the environment variables `DB_ROOT_USER` and `DB_ROOT_PASSWORD` can be used the first time the container is started to automatically create a superuser account.

If you would rather not supply these environment variables, are not using our Docker image, or have an existing PostgreSQL database server, the following SQL command can be run as a superuser to create an appropriate role:

CREATE ROLE <username> WITH SUPERUSER LOGIN ENCRYPTED PASSWORD '<password>';

This account should only be used to create databases and new roles. It is necessary to be a superuser account as the HLL extension cannot be created in a new database by a non-superuser. It should **not** be used as a general account given the privileges granted to it.

3. Installing base roles

After creating a superuser, create the base roles that DIRBS requires. These are all marked `NOLOGIN`, meaning it is not possible to login as these roles – they are just abstract roles that can be `GRANT`'ed to real users with `LOGIN` privilege.

These roles must exist before the database can be created or installed.

a. Run the following command on the DIRBS processing blade:

```
dirbs-db --db-user <username> --db-password-prompt install_roles
```

where `<username>` is the name of the superuser created in Step 2.

You will be prompted for your password.

Since this is a high privilege database account and this is a one-off command, it is more secure to enter the password manually via the prompt than to store it in a config file.

4. Creating an empty database

Ownership of the database is important as we must ensure that **every** power user has rights to do everything, rather than just the user creating the DB.

To create an empty database owned by `dirbs_core_power_user` role:

a. Use the `psql` command to login to the `postgres` database on the PostgreSQL server using the superuser role created in Step 2.

b. Run the following SQL command to create the empty database:

```
CREATE DATABASE <database_name> OWNER dirbs_core_power_user;
```

c. Connect to the new database using the following command in `psql`:

```
\c <database_name>
```

d. Run the following SQL commands to install the HLL extension in the new database:

```
CREATE SCHEMA hll;
GRANT USAGE ON SCHEMA hll TO dirbs_core_base;
CREATE EXTENSION hll SCHEMA hll;
```

5. Create user accounts

With an empty database, we can create user accounts while still logged into the PostgreSQL database as the superuser. This can also be done later.

a. Create an initial power user (can do everything):

```
CREATE USER <username> WITH LOGIN ENCRYPTED PASSWORD '<password>' IN
ROLE dirbs_core_power_user;
```

Optionally, extra users with lower privileges can be created to perform specific tasks with minimum required privileges:

```
CREATE USER <username> WITH LOGIN ENCRYPTED PASSWORD '<password>' IN
ROLE <role_name>;
```

Roles can be GRANT'ed or REVOKE'd after user creation:

```
GRANT <role_name> TO <username>;
REVOKE <role_name> FROM <username>;
```

Table 2-1 provides a complete list of roles and their functions.

Table 2-1 Roles and functions

Role	Function
dirbs_core_api	Runs REST-ful APIs
dirbs_core_catalog	Runs dirbs-catalog
dirbs_core_classify	Runs dirbs-classify
dirbs_core_import_golden_list	Runs dirbs-import golden_list
dirbs_core_import_gsma	Runs dirbs-import gsma_tac
dirbs_core_import_operator	Runs dirbs-import operator
dirbs_core_import_pairing_list	Runs dirbs-import pairing_list
dirbs_core_import_registration_list	Runs dirbs-import registration_list
dirbs_core_import_stolen_list	Runs dirbs-import stolen_list
dirbs_core_listgen	Runs dirbs-listgen
dirbs_core_poweruser	Can do everything (only user that can do dirbs-db and dirbs-prune)
dirbs_core_report	Runs dirbs-report

A user can be in multiple roles. To see a list of which users are on the system and which roles they have, we recommend using the psql command \du to view the list of database users.

6. Install a schema

Once user accounts have been created and the connection details configured for DIRBS, install the DB schema:

```
dirbs-db --db-user <username_of_power_user> --db-password-prompt install
```

NOTE: This DB user should be the power user created in Step 5a and not the superuser created in Step 2 for security reasons. We do not recommend storing passwords for power users in config files or environment variables for additional security.

7. Choose a number of physical shards

Many DIRBS Core tables are split into multiple physical partitions based on their IMEI. The default number of partitions is 4. For larger deployments, it is recommended to increase the number of partitions to achieve higher potential parallelism and performance.

This is done via the command:

```
dirbs-db --db-user <username_of_power_user> --db-password-prompt
repartition --num-physical-shards <num_shards>
```

This can be done at any time, but we recommend initially avoiding a lengthy re-write of all existing tables later.

2.3.2 Upgrading a database schema when a new release is received

To upgrade a schema from a previous release, run:

```
dirbs-db --db-user <super_user> --db-password-prompt install_roles
dirbs-db --db-user <power_user> --db-password-prompt upgrade
```

where

- `<super_user>` is the superuser (see Step 2 of Section 2.3.1)
- `<power_user>` is the power user (see Step 5a of Section 2.3.1)

3 Configuration

An annotated config file `etc/config.yml` was distributed with this release. For information on the available settings and what they mean, refer to the comments in the annotated sample config file.

This annotated config file is automatically copied into built Docker images and is in both files:

- `/opt/dirbs/etc/config.yml`
- `/home/dirbs/.dirbs.yml`

When DIRBS Core software runs, it looks for a config file in `~/.dirbs.yml`. If that file is not found, it looks in the system location `/opt/dirbs/etc/config.yml`.

Once the Docker image has been built, the configuration can be modified by mounting a config file as a volume to `~/.dirbs.yml` to override the config file that the Docker image was originally built with (see Section 2.2.1.1).