# HCL MT CH-MSP Product Documentation

## Importing Databases - Post Import Steps - Populating OM

## Importing the databases

This page gives an overview of the two-phase database migration process. For background information, see The need for database import staging.

**Important: Please make sure that users do NOT login and access Connections before their data are imported!**

**1. Prepare staging databases**
- A staging server with DB2 is needed for data massaging to fix up external users and their references:
  - All Connections databases should have been created
  - There should be no existing data on all tables
  - There should be no constraints
- See the "Preparing the DB2 staging environment" section in Importing databases to staging for the specific steps to prepare the staging server.

**2. Fixup data on staging server**
See the 'Importing the Exports' section in Importing databases to staging for the specific scripts to run for the following steps:

**a) Import data provided by IBM to staging database server**
- Run import script to import data from IBM for the migrating customer to the staging server
- **Note:** All ifx files will be imported to the staging server, regardless of whether the data need to be massaged or not

**b) Fix up external users on staging server**
Scripts are provided to massage the data as follows:
- Generate new subscriber IDs, and replace the old subscriber IDs with the new ones in all references in all databases
  - **Note:** New subscriber IDs are generated by appending the migrating orgId to the old subscriberId
- Generate new email addresses, and replace the old email addresses with the new ones in all references in all databases
  - **Note:** New email addresses are constructed by inserting the migrating orgId to the old email addresses, e.g., olduser_orgId@old_company.com
- Generate new internal IDs, and update all references to the old internal IDs in all databases with the new internal IDs

**Note:** There is no update to the Profiles databases, because there are no external users exported for Profiles from the IBM Cloud.

**c) Export massaged data from staging server**
- Scripts are provided to export all data on the staging database server
  - All data belong to the same migrating organization because it starts out with empty databases
- **Note:** The final package of the exported ifx files are one-to-one matched with the package delivered from IBM

**d) Generate ldif files and CSV file**
- In the provided package, shell script **create_ldif.sh** is provided to generate an ldif file for all INTERNAL users for the migrating organization
  - The ldif file is ready to be used to add the internal users to LDAP
- Scripts are provided to generate a CSV file for all external users
  - CSV file contains the following information:

- The original subscriber ID
- Original email addresses
- Whether they are active or inactive
- Original internal ID
- New internal ID

**About create_ldif.sh:** This script allows you to generate an ldif file from the peopledb (employee table). Because we don't know the exact specifics of your LDAP configuration, the code in the script provided is standard and you can edit it to generate the appropriate fields for your LDAP. The script optionally queries the employee table to extract the user's specifics. To see the usage, invoke the script with the -? option. There are three parameters that are required, -o orgId, -n orgName and -d for the base dn. A -q parameter can also be provided and it will execute the DB2 query and create the CSV file that is used to construct the ldif. If you choose to use your own CSV file, it must be named input_file.csv. You have to redirect the standard output to create the ldif file.

**Example: sh create_ldif.sh -q -o 1000530836 -n HCLTestOrg1 -d "ou=collab,dc=hcl,dc=com" > ScriptTestOrg1.ldif**

**3. Import data to production server**
- Run script to import all massaged ifx files exported from staging server (see Importing databases to production for specific steps)
- **Note**: The number of ifx files are exactly the same as they are in the original export from IBM.

**4. Add internal users to LDAP**
- Use the ldif file generated from the staging server, add the internal users to the LDAP
- **Note:** There is no need to provision internal users to Connections because they are brought into Connections databases via import.

**5. Register external users**

See Provisioning external users for managing external users.

## Importing databases to staging

This page provides instructions for the preliminary import of the databases. For an overview of all data migration tasks, see Importing the databases.

### Preparing the DB2 staging environment

Prior to converting the data, each database needs to be prepared to manipulate the data by adding in a schema, two tables, and some stored procedures that will support the mapping of identities and update the content fields, as well as the dropping of all constraints from Connections tables and cleaning any data out of them that might have been added during the database creation by the Connections database creation scripts.

1. Run create_db.sh -d (provided with Connections) to create the DBs. The -d option will drop the tables and create new ones. For the most part the database tables should be empty, though In some cases the service creation script will insert default "operational" data into the some of the tables that needs to be removed.
2. Run drop-db-constraints.sh to remove the constraints that will block data manipulation, i.e. foreign keys. This script will also delete any row data that had been inserted by the service specific scripts.
3. Run db2 -td@ -vf create-xcc-db.sql to create (recreate) the XCC database and tables. While the create_db.sh creates the XCC database, population doesn't happen
   until runtime, so for the full data import to happen properly the XCC database needs to be in place
4. Run add-extusers.sh to add the EXTUSERS schema, tables and stored procedures in the databases that be used for the data massaging
5. At this point it would be a good idea to create a full backup of the databases. By doing so, you can simply restore the fully prepared set of databases rather than perform the above steps for the next org that you are migrating.

### Importing the "exports"

1. Run insert.sh <org-id> from the importUtils.zip script package as has been described in the import documentation. This script will perform the preliminary import of the customer data into the staging databases.
2. Run populate-extusers-tables.sh <org-id>. This script will extract the external users from the Connections databases and populate the tables in the EXTUSERS schema to prepare for the data "massaging"
3. Run id-update-cols.sh This script will update all of the columns across all of the DBs with the "massaged" identifiers of the external users.
4. Run extid-update-cols.sh This script will update the member tables with the new external (subscriber id) and the id that is internal to each service. The new external id will be the external user's subscriber id concatenated with the new org id. The new internal id will be a unique string value.
5. Run create-extuser-libraries.sh <org-id> to create a files library for the formerly external users, so that when they are provisioned they will automatically have a library.
6. Run set-extuser-to-inactive.sh <org_id> to set the state of all external users to inactive.
7. Run dump-all-external-users.sh To get a dump of all external users with their old and new id values, you can use this list to aid in the creation of LDAP entries for the "new" org users.
8. Run export-all-tables.sh to generate the new set of exports to be imported to the target DB, using the import scripts in the importUtills.zip package.

## Importing the databases to production

This page provides instructions for the final import of the databases. For an overview of all data migration tasks, see Importing the databases.

We recommend that you do the imports during an outage window. There are a small number of instances for which table constraints needed to be dropped to successfully import the data.

1. Run a backup of ALL databases prior to starting an import of the data for a new organization.
2. Do not log in as any user for an org until all content for that organization has been imported. This means do not import the profiles database and test out those users prior to all other content being imported. If you do, records can be created for those users that will conflict with the import.
3. Either before or after you import EMPINST (profiles), you need to ensure that you have LDAP records for the users before they can log in.
4. Copy the package from the staging server to the production database server, after the data have been massaged for external users as described in the previous step.
5. To run the main import script, insert.sh, place it in the directory immediately above the database export directories, e.g., <your directory>/db-exports. As the Db2 administrator, e.g., db2admin, you can invoke insert.sh for each database individually or all databases at once.
   - For one database, the invocation is bash ./insert.sh <org_id> EMPINST, and then you would do the rest.
   - For all databases, the invocation is bash ./insert.sh <org_id>. **It is recommended that you import all databases at once.**
6. Update.sh performs update_insert operations instead of insert operations. You would use this if for some reason you need to do a re-import a given database.

## Database fixup scripts

There are two scripts that have to be run to perform fix ups after the database import is complete:

**clearEncryption.sh:** This script sets the encrypted flag for any files that had been encrypted at upload time. In the MT environment per-file encryption is not supported, so all files that had been encrypted were decrypted at export time. It sets the encrypt flag in the files.media table to 0 and clears the encryption key field. If this script is not run, files that had been encrypted, now decrypted, will not be accessible and will cause errors when a user tries to open them. To invoke this script, run

**sh clearEncryption.sh -org-id=<org_id>**

**Fixup_rc.sh:** This script fixes up a situation with recommended communities, where the migrated URL retains the Cloud value, and needs to be updated to use the orgs vanity URL. To invoke this script, run

**sh fixup_rc.sh -org-id=<org_id> -vanity-host=myorg.domain.com**

### Rejections During Import

As you import customer data into your multitenant environment you will note that some data may have been rejected. There are a number of reasons why data is rejected and in most cases it is due to data that might be common to all orgs being noted as duplicates. You will notice this, in most cases, member profile or member principal tables for Activities and Forums for example, Tags related data for Files and Wikis as well as some Homepage data. Please see this document for the an explanation for known rejections.

There have been some cases during the import, where an unusually high number of rejections have occurred. We've noted, that in those cases, an org user (customer) had logged into Connections prior to the final import of data having been done. As we had noted in the documentation this can cause issues during the import of database data. If an organization user has logged in, the rejections will likely have a noticeable impact to the end user.

Several of the Connections applications maintain, "member tables" that effectively map the users as they are known by Profiles (and in the cloud, the BSS) to a user identity used by the application. The user identifiers in the member tables are added during the migration with values that had been established in the cloud. If an org user, that had an identity in the cloud, logs in, a new identifier, different from what is to be imported, can be generated, and added into one or more of the member tables, depending on what the user does. Because these identifiers for each user need to be unique, any data associated with that org user will be rejected on import for violating database constraints.

To remedy this, we've created some scripts that will

- Assist in determining if there are any prematurely logged in users in your production system.
- Quickly identify rejections in the import logs, so you can judge whether you need to worry (refer to the list of known rejections)
- Fix those member tables that might have records associated with the user(s) that might have logged in prematurely

The scripts are located in the ***importUtils.zip*** package in the "fixup" directory, they are.

***parse-export-log.sh <log-file>. (e.g. sh parse-export-log.sh db-import.log)*** is a grep-based script that will summarize import rejections that you can use to quickly determine what rejections that might have occurred during an import into production.

***get-all-org-members.sh <org-id>*** is a simple script that queries the member tables and outputs each member's member id, subscriber id and email address in csv format. Note that "subscriber id" and "member id" are general terms, the column names in the member tables vary.

1. This script should be run against the staging database once all of the data is massaged, to capture the list of members that are expected to be imported into production.
2. Prior to importing data into production, this should be run against production first to determine if any org users have logged in prematurely. If any users appear in the member tables do not proceed with the import until you have fixed up the affected member tables. (see below)
3. If in the case where you had already imported the tables and noted a broad set of rejections and you suspect that org users had logged in, you can run this against production to extract the list of members that had actually been successfully imported into the member tables, where you can compare against the list of members that were meant to be imported from the staging environment. (You may have to reimport the org into staging)

***fix-premature-login.sh <org-id> <subscriber-id> (e.g. sh fix-premature-login.sh 1000530836 1000531578).*** This script, given the subscriber id of an org user that might have logged in prematurely, will go through all of the member tables and the member "login" tables and effectively mark the email address and the subscriber id of that user as invalid. By changing those values, a proper import will be allowed to proceed. The email address will be preceded by 'xxxx_' and the subscriber id will be prepended with 'xxxx' as well as appended with 'xxxx'. So for example, if amy_hcltestorg1@isc4sb.com, having a subscriber id of 1000531578, in the database tables will be *xxxx_amy_hcltestorg1@isc4sb.com* and *xxxx1000531578xxxx* respectively.

Prematurely logged in users may have limited capability as to what they can do. Simply navigating to the communities tab, will create a records in the communities member tables, this may also be the case with activities. If they create anything like an activity or a community records will also be created in the homepage member related tables as well.

Any data in those tables, once the script is run will become inaccessible unless a new user is provisioned using the neutralized subscriber id and email address. Once the migrated "amy_hcltestorg1" is provisioned only the migrated content will be visible (we have noted if a forum post prematurely is created it may show up in the activity stream and will be inaccessible to the migrated amy_hcltestorg1.)

You can run this script either before the import or after the import.

1. If as a first step, you check to see if a user has indeed logged in prior to doing the import, using using the ***get-all-org-members.sh*** script, you can run the script to "neutralize" the member table data. You can then run the import using ***insert.sh*** as you normally would.

2. If you had already run the import using ***insert.sh*** and discovered a broad range of rejections, once you have found the subscriber id for the user that had prematurely logged, run the script to update the member table data. You can then execute the import using ***update.sh*** to complete importing the data.

## Post-import steps

### 1. URL Fixup using ICXT

Because the target environment address is not known, URLs that point to Connections Cloud resources are migrated pointing their original location in SmartCloud. You can install and use ICXT to locate those URLs and change them so that they point to the appropriate deployment.

<TBD>

### 2. Search Re-Indexing

Search indexes are not migrated, so the content needs to be re-indexed.. Please follow the standard
HCL Connections Search documentation

### 3. Recreating the Communities Catalog Index

Upon migration you may experience that the Communities Catalog is not visible. To display the catalog, we are providing a script to effectively allow for the catalog index to be created in the background without having to explicitly re-create the index and restart the servers. in the importUtils.zip package you will find a new fixup script,***update-community-catalog.sh*** that is to be run after you have completed importing all of the data for each organization. The script updates a timestamp for each community that will force the background indexer to run and index that organization's communities.

To run the script simply invoke it: ***sh update-community-catalog.sh <org id of the organization being migrated>***

### 4. (Optional) Database cleanup script

A "cleanup" script has been provided to assist in backing out changes to the DB that occur as a result of an import. This script should be run only in the event of recognized anomalies immediately after the import is done. There are limitations, in that some deletions cannot be performed because of DB constraints. Delete query can only delete data that is associated by the org_id of the org being imported. This will be particularly notable for the many of many member tables, where there will be external users with org_id's that differ from the imported org. Because this process is imperfect, having a DB backup prior to the import is the best practice.

If you run this script for a given application, you will have to restore all of the on-disk files as well, because database triggers will force records to be added to tables that are used to indicate which files are to be cleaned up from disk.

This script is based on a generic "export" script, but modified to do "clean up" rather than export. For a specified database, it will read a "rule" file to get the list of tables in the database and perform a "clean up" on each table. At the moment, the export rule (SQL that selects certain rows for each table) is ignored, and a generic brute-force "delete all records for the specified org" is performed for each table. It is intended that the "update" script (for importing org data using insert_update) can be executed after running this script (without rejecting any rows).

- Syntax: bash ./clean.sh orgid dbname :
  Specify the org id and database name to be cleaned. Example: bash ../clean.sh 1000530836 SNCOMM

# Populating user info for Orient Me

After data are imported for an organization, newly migrated user data need to be populated to Mongo DB for Orient Me to start capturing user activities.

For those organizations that have been migrated, but their user data have not been populated to Mongo DB, the following steps are also applicable.

The general migration documentation about data migration for Orient Me is here:

https://help.hcltechsw.com/connections/v65/admin/install/cp_config_om_prepare_migrate_profiles.html

**1. Prepare the admin account for migration**

Make sure that 'connectionsAdmin' user, e.g., wasadmin, has an entry in MEMBERPROFILE table in Communities DB. If not, you can run the direct SQL to insert it:

i). First, find the UUID for the connectionsAdmin user:

/opt/IBM/WebSphere/AppServer/profiles/AppSrv01/config/cells/bvt1Node01Cell/fileRegistry.xml:

```
    <wim:identifier externalId="966fbd35-9a6a-491e-a4ca-64e3733d0c86" externalName="uid=wasadmin,o=defaultWIMFileBasedRealm"
        uniqueId="966fbd35-9a6a-491e-a4ca-64e3733d0c86" uniqueName="uid=wasadmin,o=defaultWIMFileBasedRealm"/>
    <wim:parent>
```

ii). Use the UUID found above, and use it in the following SQL:

```
INSERT INTO SNCOMM.MEMBERPROFILE
(MEMBER_UUID,DIRECTORY_UUID,DISPLAY,EMAIL,"LOCALE",LASTLOGIN,STATE,ORG_ID,GROUP_LAST_SYNCH,LOWER_DISPLAY,ISEXTERNAL)
VALUES
('wasadmin','<UUID_FOUND_ABOVE','Communities Administrator',NULL,'en_US',CURRENT_TIMESTAMP,0,'',CURRENT_TIMESTAMP,'communities
administrator',0)
;
```

**2. Clean any possible data for the migrating organization**

i). Login to the server where Connections Component Pack is installed, get into the mongo-0 pod
ii). Check to see whether there are any data already for the specific organization:

```
rs0:PRIMARY> db.idmapping.find({ "orgId" : "<orgid you are migrating>" })
rs0:PRIMARY> db.profiledb.find({ "orgId" : "<orgid you are migrating>" })
```

If there are existing data, you can delete them like the following:

```
db.idmapping.deleteMany( { "orgId" : "<orgid you are migrating>" } )
db.profiledb.deleteMany( { "orgId" : "<orgid you are migrating>" } )
```

**3. Migrating one organization at a time:**

a). Login to the server where Connections Component Pack is installed, and get to the 'people-migrate' pod:

```
kubectl exec -n connections -it $(kubectl get pods -n connections | grep people-migrate | awk '{print $1}') sh
```

You should find 'migrationConfig' file at the folder, which is the home folder: /usr/src/app

b). Modify **migrationConfig**

i). For SERVER_HOST, use the 'generic' hostname, i.e., do not use the org-specific URL.
ii). For the URLs with PROFILES_xxxx properties, add the orgId parameter to the path.
iii). For PROFILES_USER_NAME and COMMUNITIES_USER_NAME, use the 'connectionsAdmin', e.g., wasadmin
iv). Sample migrationConfig file:

```
{
 "API_FORMAT": "atom",
 "SERVER_HOST": "https://lcauto4.cnx.cwp.pnp-hcl.com",
 "MONGODB_HOST": "mongo:27017",
 "MONGODB_PORT": "",
 "MONGODB_RS_NAME": "rs0",
 "PROFILES_USER_NAME": "wasadmin",
 "PROFILES_USER_PASSWORD": "pqgtdlwk",
 "PROFILES_ADMIN_API_PATH": "/profiles/admin/atom/profiles.do?orgId=4000000002",
 "PROFILES_REPORT_CHAIN_API_PATH": "/profiles/atom/reportingChain.do?orgId=4000000002",
 "PROFILES_NETWORK_API_PATH": "/profiles/atom/connections.do?orgId=4000000002",
 "PROFILES_NETWORK_PAGESIZE": 100,
 "PROFILES_FOLLOW_API_PATH": "/profiles/follow/atom/resources?orgId=4000000002",
 "PROFILES_FOLLOW_PAGESIZE": 100,
 "COMMUNITIES_USER_NAME": "wasadmin",
 "COMMUNITIES_USER_PASSWORD": "pqgtdlwk",
 "COMMUNITIES_FOLLOW_API_PATH": "/communities/follow/atom/resources",
 "COMMUNITIES_DSX_USER_NAME": "wasadmin",
 "COMMUNITIES_DSX_USER_PASSWORD": "password",
 "COMMUNITIES_MEMBERSHIP_API_PATH": "/communities/service/atom/dsx/membership",
 "COMMUNITIES_FOLLOW_PAGESIZE": 100,
 "POPULATE_MIGRATION_PEOPLE_PAGESIZE": 500,
 "MAX_REQUEST_NUMBER_PER_MINUTE": 20000,
 "MAX_CONCURRENT_USER": 50,
 "STEP_POPULATE_CODE": false,
 "STEP_MIGRATE_PEOPLE_INFO": false
}
```

c). From the people-migrate pod, run the following command:

```
npm run start migrate:clean
```

d).Check the outputs for any errors: from the command window, or the migration.log at: /usr/src/app

Follow the same steps for other organizations that need to migrate their data for OrientMe.