Synchronization Engine guide for GNU Health

V 0.0.1

Scope of this document

The purpose of this document is to have a general purpose guide that covers the synchronization process among the health centers across Jamaica, and their relation to the central instance.

The intended audience of this guide are project managers and system administrators. It's quite general, and avoids getting too technical, although some topics and tasks require knolegde of Operating Systems, Networking , Database administration and Python programming.

Nomenclature : When writing the guide, I propose some specific naming convention, so it will make the implementation and documentation process easier.

Definitions :

GNU Health instance : Stand-alone GNU Health installation. It contains the database , tryton kernel and, at least, the core health module.

Satellite : Each GNU Health instance that is part of the synchronisation

Central Instance : The main GNU Health instance at the Ministry of Health. This is the instance that contains the aggregated information, and gets the synchronization requests from the satellite instances.

Rabbit-mq : The message broker. It is executed at the operating system and run as a daemon.

Celery : The asynchronous task / job manager used by Tryton. It uses Rabbit-mq as the middleware. The tasks for synchronization are periodically lauched from the operating system at a pre-defined fixed interval (eg, each 5 minutes).

Work In Progress : This document, is work-in-progress, and the best place for it is at a wiki, and in wiki format. We need to be updating it constantly, and make it a collaborative guide, where we all can contribute to it.

In the near future, I will document the process of GNU Health implementation at the GNU Health wikibook, but that won't cover anything specific to our Jamaica implementation.

Modules to be Installed in the SATELLITES health centers

Name	Currently Installed	Notes
health	yes	Core module
health_jamaica	yes	Main localization module
health_jamaica_jiss	yes	Jamaica Injury Surveillance System
health_pediatrics	yes	Pediatrics, including neonatology information
health_archives	yes	Tracks the paper-based clinical history archives
health_gyneco	yes	GYN/OB
health_inpatient		Hospitalization
health_lifestyle	yes	Person Lifestyle, habits, ...
health_history		Report of the patient clinical record
health_crypto		Adds digital signing / verification / hash
health_nursing		Nursing – both outpatient and inpatient
health_socioeconomics	yes	Person socioeconomic indicators
Tryton modules installed as dependencies		
Party	Yes	Core module of party, addresses, contact,...
Company	Yes	Company,
Currency	Yes	Data for currencies
Country	Yes	Data for countries and subdivisions
Product	Yes	Product model definition

Additional modules to be installed in the Central Intance

Name	Currently Installed	Notes
health_icd10	yes	ICD10 Disease List
health_icpm	No	International Coding Procedures in Medicine

Note : You might see a "test_synchronisation" module in the module list . DO NOT click on install, since it has been installed at Operating System level already.

Installation of Satellites and Central instance

Satellites

Install rabbitMQ messaging system server (current version 3.3.4)

pkg install rabbitmq

Install Celery-tryton (locally, with "gnuhealth" server Operating System user)
It will install Celery as a dependency

Set the PYTHONPATH variable to your current gnuhealth installation !

From GNU Health v 2.6, the PYTHONPATH variable is included in the user environment.

```
$ export PYTHONPATH="/home/gnuhealth/gnuhealth/tryton/server/trytond-3.2.0"
$ pip install --user celery_tryton
```

Uncompress and install the Tryton synchronisation package for Satellites

```
$ tar -xzvf tryton_synchronisation.tar.gz
$ python ./setup.py install –user
```

Install and patch Proteus

$ pip intall –user proteus

Optional : If using Proteus, apply following patch to proteus http://hg.tryton.org/proteus/raw-rev/d9bd28fde9f9

Satellites and Central Instance

Apply patches for Trytond

```
~/gnuhealth/tryton/server/trytond-3.2.0/trytond $ patch -p2 < trytond-patch-
register_wire_types-3.2.patch
```

Running the synchronization engine on SATELLITES

(Notes taken from FreeBSD. The specifics vary from operating systems)

1) Start Rabbit-mq service

```
# service rabbitmq onestart
```

2) Create a configuration file "celeryconfig.py" with the following entries

```
TRYTON_DATABASE = "name_of_your_satellite_instance"
TRYTON_CONFIG = "/home/gnuhealth/gnuhealth/tryton/server/trytond-
3.2.0/etc/trytond.conf"
```

This celeryconfig.py file should be stored in a place available in $PYTHONPATH.
Let's use /home/gnuhealth/gnuhealth/tryton/server/tryton-3.2.0 (which is the same value as $PYTHONPATH)

Nomenclature:
TRYTON_DATABASE : I used "satelliteams" . No "-" (ams as the city code for Amsterdam). The name of the health center would be ok also (like "santacruz"). In any case, it will be beneficial to have a nomenclature for the databases in the island.

3) Start the Tryton GNU Health instance

```
$ cdexe
$ ./trytod
```

4) Start Celery Broker

```
~/.local/bin $ ./celery --app=celery_synchronisation worker --config=celeryconfig
```

5) Execute the tasks

```
./celery call celery_synchronisation.synchronise_new --config=celeryconfig
```

or

```
./celery call celery_synchronisation.synchronise_pull_all --config=celeryconfig
```

or

```
./celery call celery_synchronisation.synchronise_push_all --config=celeryconfig
```

This tasks will be called from a cron job . We will start by calling them every five minutes.

TEST ENVIRONMENT LAYOUT

Central Instance : health.gnu.org

Sync id	City, Town	Operational	OS, DB	hostname:port, dbname	Remarks
0	Atlanta(GA), United States	Yes	Ubuntu 12 PG 9.1	health.gnu.org:9500 xml-rpc : 7500 dbname : jamcentralinstance	

Satellite instances

Sync id	City, Country	Operational	OS, DB	hostname:port, dbname	Remarks
1	Amsterdam, Holland	Yes	Archlinux, PG 93	health.gnusolidario.org:9500 dbname : satelliteams	Issues reported on the celery broker.
2	Las Palmas, Spain	Yes	FreeBSD 10 PG 94		Currently firewalled.
3	Buenos Aires, Argentina	No			
4	Vietnam	No			
5	Kingston, Jamaica	No	CentOS		
6	Las Palmas, Spain	No	Archlinux, PG		

Note:
The "Sync id" correspond to the synchronisation_id parameter on the trytond.conf file. The Central Instance will always have the synchronisation_id = 0

Proposed Port numbering

JSON-RPC (to connect with the clients)	9500
XML-RPC (Central Instance)	7500

Technical documentation

Tryton Synchronisation

Adds record synchronisation support to Tryton.

Each server instance has a unique ID (`synchronisation_id`) set in the configuration file.

The number of instance is limited to 512 starting from 0 to 511.

A "Celery" instance provides 3 main tasks:

* **synchronise_push_all :** pushes to the main server all the instances modified since its last synchronisation.

* **synchronise_pull_all** : pulls all the known instances that have changed on the main server.

* **synchronise_new** : fetch all the non-synchronised instances from the main server.

The tasks communicate with the main server using the XML-RPC protocol defined by "synchronisation_url" in the configuration file.

`celery_synchronisation` uses `celery_tryton`_ to integrate with Celery.

Developers “mini-guide” to the syncrhonization engine

The health_jamaica_sync module contains the classes

Synchronization models :

There are two main models used to synchronise records from objects.

- **SyncMixin** : Synchronises using an **existing** unique key on the model
- **SyncUUIDMixin** : It uses a Universal Unique Identifier (UUID) on each record

SyncMixin

If the model has a unique code, then we should use the SyncMixin method to synchronize the records. A good example for SyncMixin would be the gnuhealth.patient or the party.party models. Both have a unique id attribute (field) and this is the field that will be used by the syncronisation engine.

Examples of SyncMixin use:

```
class Party(SyncMixin):
    __name__ = 'party.party'
    __metaclass__ = PoolMeta
    unique_id_column = 'ref'


class PatientData(SyncMixin):
    __name__ = 'gnuhealth.patient'
    __metaclass__ = PoolMeta
    unique_id_column = 'puid'
```

Note that in the SyncMixin model, the unique_id_column must always be present, and assigned a field that is unique (in this case “puid“)

SyncUUIDMixin

This syncrhonisation method is used for models that represent dynamic events. For example, a patient appointment (

```
class Appointment(SyncUUIDMixin):
    __name__ = 'gnuhealth.appointment'
    __metaclass__ = PoolMeta
```

Note that there is no “unique_id_column” on class using the SyncUUIDMixin