Table of Contents

[Checkins](#_ra1zrc23evc3)

[Schema changes](#_jgfhc0j00tau)

[Release versioning](#_f6t1lxdbeqkj)

[Lookup tables](#_z1f222rsn00x)

[Unit Testing](#_ppcz5b2znhp9)

[Build environment](#_gh4wu5g1dpop)

[Building](#_jiv0d668off3)

[Code Freeze](#_n0m4rrpy34l)

[Acceptance Testing](#_ivd3xeuswbcl)

[Release Notes (Change Document)](#_sxmfge59oib2)

[Deployment](#_4ez8o53fphew)

[Consolidated site upgrading](#_solm7w82aysc)

### 

### 

### Checkins

It is assumed that all members of the development team have sufficiently unit tested their work such that a checkin never impacts the stability of what is already in the repo.

As the time comes closer to the release date, it is expected that checkins are increasing sparse as well as coordinated and carefully planned among team members.

### Schema changes

iSanté manages the schema by comparing the current version stored in the **schemaVersion** table with the version used in the upgrade. Any logical feature additions to the schema tables or columns must be specified in a unique, numbered file located in ~/support/schema-updates/. The README in this directory states:

This directory contains database schema update files. Schema changes are tracked in the schemaVersion table. During an upgrade any schema update file that has not been applied to the database will be.

The schema update files have a special naming convention.

$number-$name.sql

$number is a four digit, zero padded, unique integer that expresses the order the change should be applied. $name is a small descriptive name for the schema change. Ideally each schema update file would only contain a single logical change to the database schema and have a name that expresses the intent of that single logical change.

All you have to do is put files in this directory and they will automatically be picked up. The format of the files is not quite standard SQL. Statements in these files are processed through the SQL translator and use a delimiter of 'go'.

It is generally a very bad idea to make any edits to schema update files that have been included in a production release. Edits to schema update files that have not yet been included in a release are no problem.

The initial schema must be kept in sync with these changes. The initial schema file is located in support/**schema-tables.sql**. The initial record inserted into the schemaVersion table within this script must match the most recent file name in the ~schema-updates/ directory.

So, for instance, the schema changes necessary for malaria reporting are in the file:

~support/schema-tables.sql/0031-Add\_malariaDataWarehouse.sql

When doing a new install, the schemaVersion table must be populated with a version record corresponding to the highest numbered script above. Edit this in ~/support/schema-tables.sql, immediately after the creation of the schemaVersion table.

One additional aspect of schema change involves replication. If any of the core tables are altered, this could affect replication of old or new data from source sites. Unfortunately, it is not possible to predict what changes might be necessary in replication without considering specifics. Fortunately, this is usually not necessary for the following reasons:

* Most new data elements in iSanté are implemented via the concept schema, which is already successfully handled by replication in a generalized way.
* Replication automatically handles new columns in existing tables, assuming that it is not trying to load new data into an old schema.
* Most replication changes in the past were the result of columns migrating from one table to another or table rows migrating from one table to another.

The migration to OpenMRS is likely to involve significant changes to the replication processing, since an entirely separate schema for OpenMRS is involved. This needs careful consideration in the design plan for the migration.

### Release versioning

The software version displayed in the iSanté banner which corresponds to the whatever release level is run in an installation is controlled by the file ~/support/version.ini. The rules for specifying version are contained within that file:

; control displayed version number

; format (optional in []): $version.$planned.[$unplanned] [$type] ($revision)

; bump this once per calendar year

version = 13

; bump this each time we have a scheduled release, starts at 1

planned = 1

; bump this for each non scheduled release (hot-fix, bug-fix, etc), starts at 0

; when equal to 0 this is not shown to the user

unplanned = 0

; indicate the kind of release

; empty = normal production bound release (build script should set this to empty)

; "dev" = currently being developed version

; "test" = testing versions

type = "dev"

; SVN revision number, will be dynamically calculated if not defined here

;revision = 1234

### Lookup tables

Lookup tables are used for validation, localized display, and general overall control of iSanté. The records in them typically change as the system evolves. Lookup tables may be added as features are added to iSanté. For instance, the malaria reporting adds this lookup table to the schema:

dw\_malariaReportLookup

Lookup tables are added to the schema as described above. In addition, their content is automatically updated during upgrades.

During development, lookup tables are modified as necessary. A replication gz file named **lookups-Data.csv.gz** can be generated once the lookup tables are finalized by running ~/replication/create-lookups.sh. This script unloads all the lookup records to a gz file and reports the differences between the previous gz file and the new one. Once your inspection validates that the new gz file is good, rename it lookups.csv.gz and check it in like any other file.

With one exception, the final lookups.csv.gz is generated with create-lookups.sh. The exception is clinicLookup, which is maintained on the UGP consolidated server and should be broadcast automatically on a daily basis to the I-TECH consolidated server and the ASP server.

### Unit Testing

iSanté is still tested manually. As a general rule, this means running through every menu, every form, and every report to be sure it works. It is usually imperative to test in older versions of IE (though the definition of how old is always open to question), because IE is the most likely to dislike small punctuation problems in JavaScript, especially trailing commas.

Fingerprint and smartcard testing involves obtaining the scanner and the smartcard reader/writer (the I-TECH Haiti office currently has them) and making sure that this functionality works.

It is also critical, once the final build becomes available, to test installs and upgrades to make sure they work and to make sure they have included all new features added since the previous release. This is done via VirtualBox. See Steve if you need an existing VM installation in order to do an upgrade test. In the past, we have tested not only upgrades from the previous release, but also upgrades that jumped 2 or more releases, since many sites do not adopt a release by the time another release becomes available.

### Build environment

1. Check out trunk (or other branch if building a specific version) of the debian-cd subproject.

svn co svn+ssh://svn.cirg.washington.edu/svnroot/itech/debian-cd/trunk isante-cd

cd isante-cd

1a. WARNING: make sure that this is a pristine checkout. The build generates packages that are stored in the directory and NOT rebuilt if present. Run this command to clean up a previous build:

svn status | awk '/^\?/{print $2}' | xargs rm -r

1b. See README.txt for a list of dependencies that must be installed for this to work. Run:

sudo apt-get install gettext ghc6 libghc6-parsec-dev libghc6-cgi-dev libxml-twig-perl ant-optional g++ libboost-date-time-dev libmysqlclient15-dev fakeroot debhelper php5-dev makeself simple-cdd imagemagick libc6-pic

2. Make a symbolic link to your checkout source:

sudo ln -s /home/websites/haiti-dev.cirg.washington.edu/isante ./isante-source

Or check out isante source. (trunk or another branch)

svn co svn+ssh://svn.cirg.washington.edu/svnroot/itech/branches/lamp70 isante-source

### Building

3. To build development, test or production media use the following commands respectively.

./build.sh (for development builds)

./build.sh test

(./build.sh release 2>&1) | tee build.out

4. Output will appear in the ~/images subdirectory.

### Code Freeze

[to be written]

### Acceptance Testing

[to be expanded]

Acceptance testing involves non-developer (user) testing of [especially] new features, but also prior features [regression testing]. Because Haiti acceptance testers are remote from our development and testing environments, a VM should be preconfigured for them with the final, code frozen build which they can install and work with locally.

### Release Notes (Change Document)

Once the contents of the release are finalized, release notes should be written as a Google Doc, then published as pdf in both English and French. These should contain any new and modified features in iSanté, written for both the end user and system administrator audiences, typically with sections for forms, reports, and system administration. Typically, the various items are authored by the developer most responsible for them. An example is here:

[ChangeDocument12.3.en (iSanté 12.3 New Features)](https://docs.google.com/document/d/1vbKfvEqqAHQWz0zbZlb7qq2MPuYvr_Iq5MDo1roJK74/edit)

### 

### Deployment

The files generated during the build process can be burned to CD, transferred to a site via sftp, and should be placed in the haiti01 user account on almond.cirg.washington.edu.

These files should have checksums, so that transfers can be verified as valid. Both md5 and sha checksums can be generated:

md5sum build.sh

shasum build.sh

When announcing a release, we want to provide the checksums for these files and explain how to use them.

### Consolidated site upgrading

iSanté currently has two consolidated sites and the ASP server (used by sites in Haiti that do not have their own server).

Traditionally, these sites were upgraded before any in-country sites were upgraded. In-country sites were upgraded piecemeal, hopefully after they were trained on the release.

Lately this has not happened. So far it has caused no problems because no structural changes affecting the core tables has occurred between recent releases. Additional reporting functionality has been added to these sites outside of the normal upgrade process in order to respond to stakeholder demand.

It is not a good situation when the consolidated sites are running old versions of iSanté. This has occurred for two reasons:

* There is disagreement about whether to do in-place upgrades of the consolidated sites (as we do with in-country sites) vs. a clone/run parallel/cut over strategy.
* Particularly with the ASP site, but also to some extent with the consolidated sites, we have been hesitant to upgrade prior to user base training being completed. The recent site visits in Haiti make it clear that such training has historically been inadequate.

Thus, this topic needs discussion and resolution.