Development Notes
for SaberNet DCS 2.0

Title:	SaberNet DCS Development Notes
Author:	Seth Remington <sremington@saberlogic.com>
Date:	2009-04-21
Revision:	1.5.2.4
Description:	Documentation for developers of SaberNet DCS.

Contents

Database Changes and Upgrades

If a database change needs to be made there are several things that must be done to ensure that upgrades will be smooth. But first, a little background:

All manipulation of the database is done through MiddleKit which is a "domain object" middle layer between the front end and the database store. MiddleKit defines the database schema in a csv file which is used not just during development but also at runtime. MiddleKit will generate all of the SQL to create the database which works great for a fresh install but if you have a production database there is no method for upgrading the structure without dropping the database and installing a clean schema which is not an option.

To get around this there is a method inside of SaberNet DCS to upgrade the database. Starting with DCS 2.1.0 each revision of the database will get an incremental integer version number. Prior to 2.1.0 the version is assumed to be 0... 2.1.0 is version 1, etc... This version number is stored inside the database itself.

In the code at sndcs_common/__init__.py there is a variable named DATABASE_VERSION which is the revision level that the database needs to be at to support the code. This will be compared with the revision level stored in the database to determine if the database needs to be upgraded. If it does need upgrading there is an upgrade directory that contains a python module for each revision level (i.e. 2.py will bring the database up to revision level 2, etc....). The upgrade_sndcs.py script in the upgrade directory will handle determining what modules need to be ran to bring the database up to the necessary level.

The upgrade modules need to conform to have the following contents:

```
1. A class named the same as the module that inherits from the sndcs.upgrad
#. The Upgrade Database class provides functions that should be overridden
    a. tablesToAdd() - If there are new tables to add to the database then
```

When development requires that the database must be changed (including adding a new INDEX) the following **MUST** be performed to ensure a smooth upgrade process:

```
1. Increment the DATABASE_VERSION number in sndcs_common/__init__.py
#. Increment the number in the "database" field of the "Version objects" ta
#. Provide an XX.py upgrade module in the upgrade directory (where XX repre
    a. The module should contain a class of the same name as the module tha
```

Table Indexes

MiddleKit doesn't allow us to create additional indexes in the Classes.csv file so there is a bit of code in configure_sndcs.py to take the contents of the sndcs.mkmodel/indexes.sql file and append it to the end of the GeneratedSQL/Create.sql file. Any additional indexes that are needed should be added to the indexes.sql file.

Running the Test Suite

To run the test suite you will probably want to create a blank database using the configure_sndcs.py script. Start up the DCS Server pointing to the new blank database. Change into the 'tests' directory and run the 'run_test_suite.py' script.

Future TODO

- The TreeModel --> TreeModelFilter --> TreeModelSort chain in the available activity list is slow. Removing the TreeModelSort gives a dramatic speed increase. Look for a workaround (perhaps a custom TreeModel? http://www.pygtk.org/pygtk2tutorial/sec-GenericTreeModel.html)
- Add spell checking (perhaps with PyEnchant)