ChildCount+ Documentation

Release 2.0(g)

Earth Institue and Millennium Villages Project

Sep 27, 2017

Contents

Introduction

CHAPTER 1

Introduction

Target Audience

This documentation is meant for **programmers** and **project managers** who are interested in deploying ChildCount+. Project managers and health professionals might focus on the *Human Aspects* section. Programmers and medical records specialists can focus on the *Technology* section.

What is ChildCount+?

ChildCount+[1] is a health data management system designed for day-to-day use by community health workers. To be specific, ChildCount+:

1. Collects health information from community health workers (by text message or paper forms),
2. Sends text message alerts to health workers and health managers,
3. Calculates the values of key health and CHW performance indicators using data collected from the CHWs, and

[1] **Why the +?** We call our system ChildCount+ (read: "Child count plus") because it has expanded from a system for collecting data about children to a system for collecting data about people – including adults. The "+" represents the fact that we count children *and* adults too.

4. Produces concise printed performance reports for CHWs and their managers.

Workflow

ChildCount+'s original design revolved around mobile phones: community health workers would submit information to the ChildCount+ server by text message (SMS). Based on the submitted data, the server would then periodically send information and alerts to the community health workers. The ChildCount+ deployment in Sauri, Kenya, where Millennium Villages Project first piloted ChildCount+, uses the mobile-phone-based workflow depicted in the *accompanying figure*.

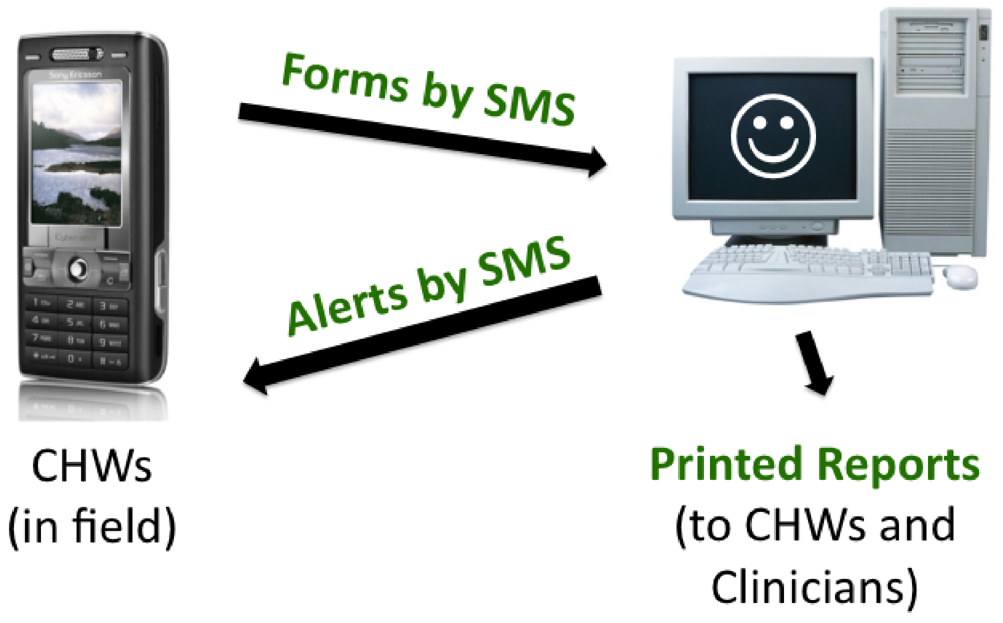


Fig. 1.1: A representation of the mobile-phone-based ChildCount+ workflow.

It is possible to deploy ChildCount+ without mobile phones. In fact, most Millennium Village sites use a *paper-based workflow* for ChildCount+, since managing airtime credit and fleet of mobile phones is sometimes not possible.

What's in the Box

The two major open-source components of ChildCount+ are:

- **Paper Forms**: See *Forms* to check out our paper data collection forms.
- **Software**: See the *Technology* section for details on our software and what you need to get it up and running.

For everything else (community health workers, data entry clerks, programmers, servers, mobile phones, airtime, ...) you are on your own!

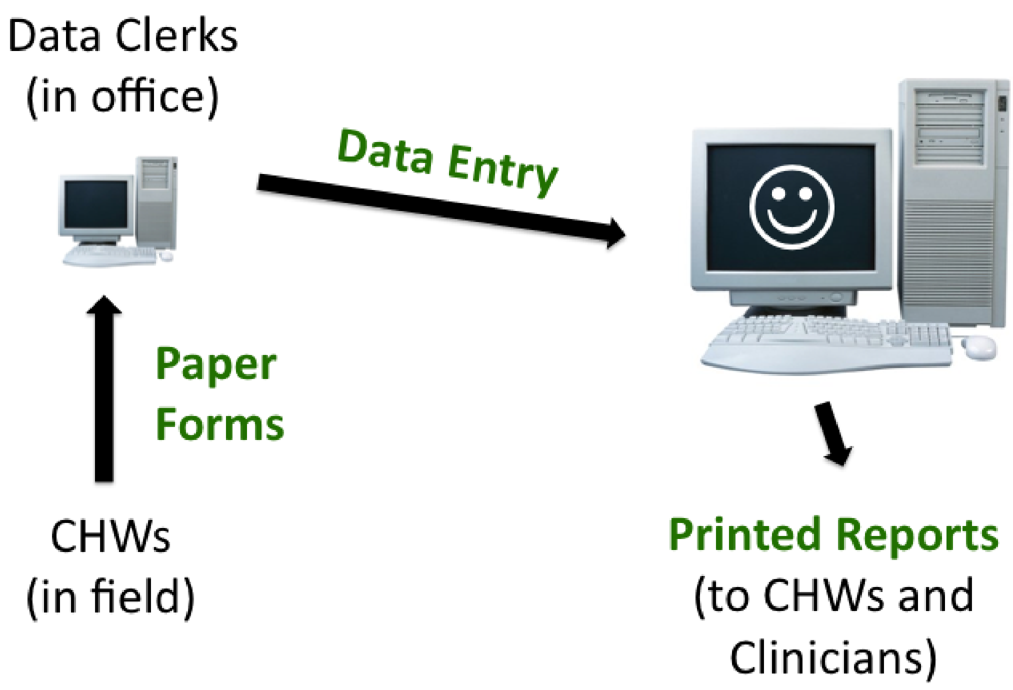


Fig. 1.2: A representation of the paper-based ChildCount+ workflow.

Should You Use ChildCount+?

Here are some questions to consider before embarking on a ChildCount+ deployment:

- **Do you have a community health worker program?** If not, ChildCount+ might not be the best platform for your project. You can customize ChildCount+ to suit your application (tracking levels of drug stocks, for example) but that would require extensive programming and customization. See *Human Prerequisites* for more information.
- **Do you have the means (i.e., transport) to meet regularly with your CHWs?** Many of the Millennium Village Project sites aim to have feedback meetings with the community health workers *every month*. If you have scores of CHWs distributed over a large geographical area, these meetings can take a non-trivial amount of time. Don't bother deploying the system if you don't have time to use the data it produces. See *Human Prerequisites* for more information.
- **Do you have health managers with enough time to maintain the system?** One major purpose of ChildCount+ is to collect and display public health data. If there's no one who has time to look at the data, and act based on what they are seeing, then maybe you should skip ChildCount+ and focus on that problem instead.
- **Do you have a technical team (or at least a technical person)?** ChildCount+ is not a "plug-and-play" solution. In fact, it is more like a "download-and-hack" solution. You will need, at least, one on-call Python programmer with some Linux systems administration experience to install the software and to maintain the server. See *Human Prerequisites* for more information.
- **Do you have money to pay for paper and SMS fees?** As an example: in Uganda (May 2011), an on-network SMS costs US$0.02. If you have 100 CHWs each sending or receiving 20 SMS messages per day, that is:

  100 CHWs * 20 SMS/day * $0.02/SMS * 30 days/month = **$1200.00/month**

Do you have $1200/month for SMS fees?[2] Paper-only deployments are cheaper, but then you miss out on all of good things that come with SMS.

- **Do you have enough cell phones and phone chargers for your CHWs?** ChildCount+ makes the assumption that there is one phone per community health worker. With a bit of engineering you could modify the system to allow CHWs to share phones, but you might lose some of the benefits of real-time CHW-to-server communication.
- **Do you have a system in place to manage a fleet of phones and chargers for your CHWs?** Cell phones break, get lost, and are stolen. If CHWs are using their phones all day every day to send SMS messages to the ChildCount+ server, then you should expect a lot of wear and tear. Make sure you have a policy and a means to replace broken and stolen phones so that CHWs can continue to submit forms even after their phone breaks.
- **Do you have a system in place to manage airtime for CHWs?** If community health workers are spending US$12/month on SMS messages, you will need a reliable way to get money or airtime to them. Millennium Villages Project has tried to negotiate with the local mobile operator for "toll-free SMS" lines, but it's not a quick process.

Deployment Background

Millennium Villages Project has deployed ChildCount+ at its sites across sub-Saharan Africa. As of May 2011, these deployments are the *only* ChildCount+ deployments. For more information on the history of ChildCount+, please see *History / Credits*.

Human Aspects

[2] We are considering a GPRS/EDGE-based alternative to our SMS-based transport. In Uganda, that would bring the monthly data cost down to less than US$10.

CHAPTER 2

Human Aspects

Human Prerequisites

Warning: Even the simplest mobile health platform will probably require a full-time or near-full-time manager to keep it running and to integrate it into your existing health program. Do not spend time and money deploying a mobile health system if you do not have the human resources to make it useful!

Community health workers ChildCount+ is a tool for community health workers to keep track of their patients and for health managers to keep track of their community health workers. ChildCount+ is not a general data collection tool (for that, see ODK) nor is it a tool that would be very useful for clinicians or for general-purpose medical record keeping (for that, see OpenMRS).

Programmers Running ChildCount+ requires access to a software developer or at least a systems administrator. The more programming talent you have at your disposal, the more flexible and useful ChildCount+ will become. ChildCount+ is written in the Python programming language and uses the Django and RapidSMS frameworks, so people with knowledge of those technologies will be particularly useful.

FrontlineSMS might be a good alternative solution for organizations without programmers at their disposal.

Community Health Worker Program

Since we designed ChildCount+ for the Millennium Villages Project, we have incorporated elements of the MVP CHW protocols into the ChildCount+ forms, alerts, and reports.

Health IDs

One important and time-consuming part of the ChildCount+ set-up process is patient registration. Each person who is to be tracked by the ChildCount+ system must be assigned a unique ChildCount+ health identifier. Our deployments use the Childcount form +*NEW* (`childcount.forms.PatientRegistrationForm`) to assign health IDs to patients.

These health idenfiers are like primary keys into a database table – each person's health identifier is unique within the system and is the most common way to reference to their patient record.

The Millennium Villages Project deployments use the OpenMRS IdGen Module to generate the health IDs and we use the Luhn-30 checksums to validate the IDs.

Forms

Here are examples of the ChildCount+ forms we use:

All Sites

- `Danger Signs List`

Sauri, Kenya

- `Form A (Patient Registration)`
- `Form B (Household Visit and Family Planning)`
- Form C (Consultation)
  - `Children`
  - `Pregnant Women`
- `Form HED (Household Associationg)`
- Form P (Pregnancy Forms)
  - `P: Demographics`
  - `P2:  Initial Antenatal Visit`
  - `P2:  Follow-up Visit`
- `Form R (Appointment Reminder Log)`
- `Form V (Clinic Visit Log)`

Ruhiira, Uganda

- `Form A (Patient Registration)`
- `Form B (Household Visit)`
- `Form C (Consultation)`
- `Correction`

Technology

CHAPTER 3

Technology

Technology Overview

Before you “get started,” make sure to take a look at the *Should You Use ChildCount+?* section of the introduction. If you are duly convinced that you need some CC+ action in your life, then read on.

Technology Prerequisites

Warning: ChildCount+ should NOT be deployed on a publicly accessible network. Data between the server and clients is not encrypted the software is not hardened against attacks.

Tip: Wait! Before you read on, check out the *Human Prerequisites* section. Make sure that you’re not missing anything on the human side of things before you jump into the land of config files and list comprehensions.

Hopefully these documentation pages, plus some Googling, will be all the information you need to get ChildCount+ up and running. In the real world, it is unlikely that the ChildCount+ team will be able to keep this documentation up to date forever, but at least we are trying!

All Installs

Linux Server You will need a server to host your ChildCount+ installation. We recommend using Ubuntu versions 10 and up, since that is what we use for development and deployments.

Printer If you need to print paper reports.

Paper-Form-Based Installs

Computers for Data Entry You will need one client machine per data entry clerk. Windows, Max, Linux, OS/2, – anything with a good web browser will do.

Local Area Network To connect your data entry computers to the ChildCount+ server.

SMS-Based Installs

GSM Modem If you want to interact with ChildCount+ via SMS. Any modem that works with PyGSM will work. We use Multitech MTCBA-G-F4 modems.

SMS-Enabled Mobile Phones One per community health worker.

Mobile Phone Chargers If you are deploying in an area without reliable power, make sure you have solar chargers or some other means of charging cell phones.

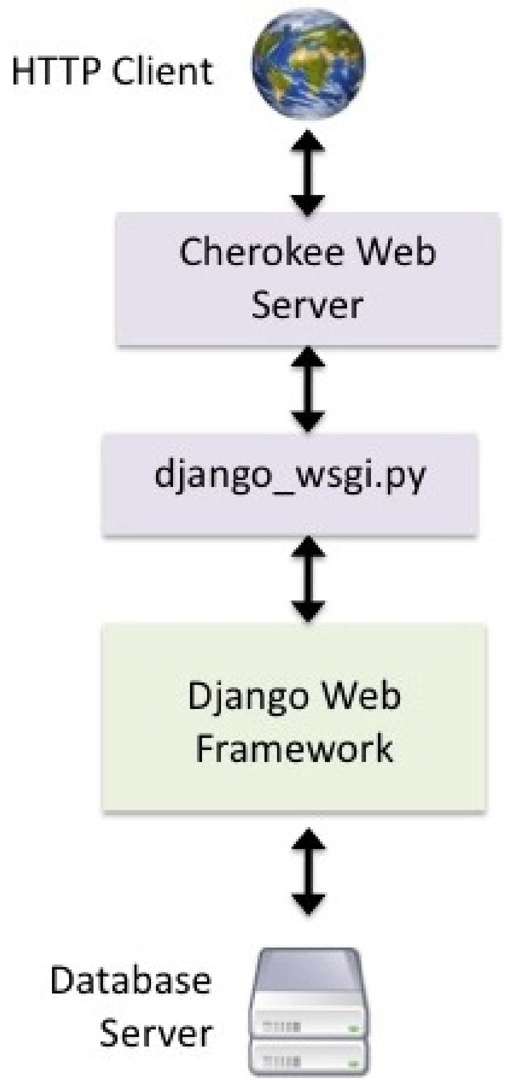
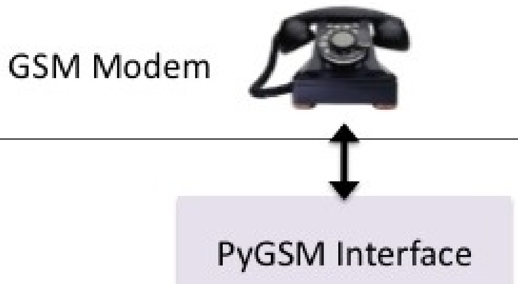
Airtime Credit Distribution System You will need some way to distribute airtime credit to the community health workers. Millennium Villages Project has tried to negotiate with mobile operators to get toll-free SMS lines, but distributing airtime scratch cards by hand is also an option.

Understanding the Components

The ChildCount+ stack is large – it includes many components patched together in non-obvious ways. The following diagrams will hopefully give you a sense of what the components of ChildCount+ are and how they relate to each other.

ChildCount+ runs on top of RapidSMS, but since there is almost no documentation for RapidSMS, we will try to document bits of RapidSMS as we go along.

Table 3.1: **ChildCount+ Component Overview**

Web. How ChildCount+ handles HTTP requests from Web browsers: • *Cherokee Web Server* handles incoming HTTP requests and passes them to... • *django_wsgi.py* – a Python script that sets some environment variables (like the local time zone) and invokes Django's wsgi (Web Server Gateway Interface) handler. • *Django*'s Web framework handles the request from there. The Web portion of ChildCount+ works like Django (with a few caveats). You will find Django-style `views.py` and `urls.py` files in the `apps/*` directories.	 HTTP Client Cherokee Web Server django_wsgi.py Django Web Framework Database Server
	

Configurations

There are a few different places where configurations happen in ChildCount+ and it's useful to know which settings go where.

`settings.py` This is a Python file that holds settings for the Django environment in which RapidSMS and ChildCount+ run. If you have settings for Django plug-ins or need to set environment variables, this is where to do it.

We use it to hold timezone settings, Django cache settings, django-celery settings, and some language settings.

`local.ini` This is a RapidSMS-specific configuration file. It is divided up into sections, with each RapidSMS app getting a single section. RapidSMS passes the values of these settings to the `configure()` method of the `rapidsms.app.App` class in the RapidSMS application `apps/[app_name]/app.py`.

The database login information is here, and the lists of activated ChildCount+ forms and reports are there under the *[childcount]* header.

`rapidsms.ini` I am not sure what this does but I am scared to delete it.

`childcount.models.Configuration` This is a Django model (a database table) that holds some configuration information that doesn't fit well anywhere else. OpenMRS login information (for *mgvmrs*) and lists of enabled dashboard sections are there too.

Dependencies

We use:

- Ubuntu 10.04
- Python 2.6
- Django 1.1
- PyGSM 0.1
- RapidSMS 0 ("old core")
- Django Celery 2.2.4
- Celery 2.2.5
- RabbitMQ Server 1.6
- Kombu 1.0.7
- MySQL 5.1

Installation Instructions

ChildCount+ is not easy to install, so make sure you have a patient Linux-savy software developer or server administration on hand before you get started.

Millennium Villages Project has an internal wiki where we keep the latest documentation on how to deploy ChildCount+. Here are the key pages:

- Installing ChildCount+ and RapidSMS (for the Web interface and database)
- Installing Celeryd and RabbitMQ (for reports and SMS alerts – follow instructions for "The Champion's Way")

There are lots of steps involved in setting up and maintaining a ChildCount+ server. If (by some miracle) you are able to get ChildCount+ running, you can take a look at the various server configuration files used at the Ruhiira, Uganda installation here:

Ruhiira Install Files

The best way to get help with the installation is to contact one of the ChildCount+ developers directly. Since the team is always in flux, check the ChildCount+ Web site to find out how to contact us.

Running the Software

Once everything is installed, you can use the following commands to start your ChildCount+ instance:

```
sudo service rabbitmq-server start
sudo service celeryd start
sudo service celery-beat start
sudo service rapidsms start
sudo service rapidsms-webserver start
```

You then should then open a browser and navigate to:

```
http://your_server_ip/childcount
```

The normal Django administration pages are at:

```
http://your_server_ip/admin
```

SMS

with RapidSMS

ChildCount is built using the RapidSMS framework. The following few sections will introduce you to a few key RapidSMS concepts: the router, backends, and applications.

Router

The [RapidSMS] router is the part of ChildCount+ that handles incoming and outgoing messages, and it operates independently from whatever Web server you are using to view the ChildCount+ dashboard. (Other parts of ChildCount+ serve Web pages and generate analytical reports.) If you are deploying ChildCount+ yourself, you will probably want to learn how to use the router to respond to special SMS keywords or to collect deployment-specific data. See the SMS section of *Understanding the Components* for an overview of the router.

Backends

The RapidSMS router interacts with the outside world via a set of "backends" (whoever chose this terminology must have had a sense of humor...).

An application that uses SMS, Web data entry, and email to interact with the world would have three backends: one for each of these three transport mechanisms. Backends inherit from `rapidsms.backends.backend.Backend` and they use a common interface to tell the RapidSMS router how to send and receive messages. The active backends are specified in the `local.ini` file in the root ChildCount+ directory.

Applications

The router treats incoming messages the same way no matter where they come from (by SMS, email, etc) – every message gets parsed into a `rapidsms.message.Message` object and handed to the active applications.

As described in *Understanding the Components*, RapidSMS steps through the active applications listed in the `local.ini` file and calls `[app_name].App.handle()` on each, with the `rapidsms.message.Message` object as an argument. Each application processes the message and returns `False` if the message should be passed on to the rest of the active applications, and `True` otherwise.

Here is an example `App.handle()` definition that responds to a message **`FLIP`** with the message **`Heads`** or **`Tails`**:

```
class App(rapidsms.app.App):
    def handle(self, message):
        if message.text.strip().upper() == "FLIP":
            response = random.choice(["Heads", "Tails"])
            message.respond(response)
            return True
        else:
            return False
```

As in Django, you can have many RapidSMS apps running on the same ChildCount+ server. The order in which the apps get to handle messages is determined by the order in which they appear in the `local.ini` file.

Some useful SMS-related apps are:

- */apps/childcount* – Handles all ChildCount+ messages
- `/apps/fortune` – Responds to the message **`FORTUNE`** with a Ugandan proverb
- `/apps/logger_ng` – Stores all messages in a message log database table
- `/apps/ping` – When it receives a message **`PING`**, it responds **`PONG`**

with ChildCount+

The body of the ChildCount+ message processing happens in `/apps/childcount/app.py` – ChildCount's RapidSMS application. The following sections describe how components *within* the ChildCount+ application process messages and how you can customize these components.

SMS Forms and Commands

To understand ChildCount+ SMS processing, you must know the difference between a *form*, a *command*, and a *report*.

Caution: We have recklessly overloaded the term "form." The word "form" can refer to the paper paper forms filled out by CHWs (see: *Forms*) or it can refer to SMS forms – the logic that parses and processes messages (described below).

Forms and commands are both means of connecting SMS keywords to bits of application processing logic. The difference is that SMS *forms* are part of a message that begins with a patient health identifier (health ID) and (*commands*) are consist of messages that begin with a keyword.

Table 3.2: Examples of SMS Commands

Message Sent to Server	Action Taken by Server
`CHECKID abc123`	Reply to sender with a message explaining whether or not the health ID `abc123` is valid.
`LOOKUP joe`	Reply to sender with a message listing all of the patients with name `joe`.
`CANCEL`	Cancel the effect of the sender's previous message.

As you see, all of the commands listed in the table begin with a keyword (like `CHECKID`). Commands are useful for situations where the message does not directly relate to a registered patient. Commands inherit from the class `childcount.commands.CCCommand`.

Other commands are listed in the commands API documentation (*childcount.commands*) and in the ChildCount+ source code in the folder `/apps/childcount/commands`.

Messagings containing SMS forms begin with a valid ChildCount+ health ID (see *Health IDs*), followed by a series of +CODE sequences. ChildCount+ checks the validity of the health ID before any of the form processing logic begins.

The SMS forms generally correspond to fields on the paper ChildCount+ forms. For example, the `+V` form below corresponds to the `+V` section of the ChildCount+ household visit form (paper form B). You can look at the paper forms here: *Forms*.

Message Sent to Server	Action Taken by Server
`ABC123 +V Y 2 BN FP`	Record that the CHW who sent the message conducted a household visit at the household headed by the person whose health ID is `ABC123`. The arguments to the `+V` form indicate that there was a household member present (`Y`), that there were two under-fives present (`2`), and that the CHW discussed bednets and family planning (`BN FP`) at the household visit.
`56HG2 +F Y +S FV VM +R B`	Record that the patient with health ID `56HG2` tested positive with a rapid diagnostic test for malaria (`+F Y`), that the patient has fever and is vomiting (`+S FV VM`), and that the CHW made a 24-hour referral for this patient to a health center (`+R B`).

Note that it is possible (and encouraged) to send many forms relating to the same patient within the same message. Combining forms this way cuts down on the number of SMS messages that CHWs need to send per household visit.

SMS forms reside in the directory `apps/childcount/forms` and the API documentation is here: *childcount.forms*. SMS forms inherit from *`childcount.forms.CCForm.CCForm`*.

You enable commands and forms by including them in the list of active commands/forms in the `local.ini` configuration file.

SMS Reports

Caution: We have shamelessly overloaded the term "report:" The word "report" can refer to the printed paper reports generated by ChildCount+ (see *Printed Reports*) or it can refer to Report models (described below).

Reports (in the context of messaging) are Django models for storing information collected from a ChildCount+ SMS form. In general, the form holds parsing and validation logic for the collected data, while the report is where the data ends up being stored. A "report" in this context is a Django model that corresponds to a database table holding the form data.

For example, the `+V` SMS form collects data about household visits. There is a class `childcount.forms.HouseholdVisitForm` that defines the parsing and validation logic for the `+V` form. Once the data has been parsed from the +V form and validated, it is stored using the Django ORM as a `childcount.reports.HouseholdVisitReport` object.

All of the ChildCount+ reports are located in `apps/childcount/models/reports.py`, and they inherit from *childcount.models.CCReport*.

Defining a Command

Say you want to define a new command called *ReverseTextCommand* that users invoke by SMS like this:

```
REVERSE First Second Third
```

To define this command, you must:

1. Look through the existing commands in `apps/childcount/commands` to make sure that the command you want does not already exist. There are lots of useful commands defined there, so please check first.
2. Create a file `apps/childcount/commands/ReverseTextCommand.py`
3. Within this new file, import `childcount.commands.CCCommand` and define a new class that inherits from it:

   ```
   from childcount.commands import CCCommand
   from childcount.utils import authenticated

   class ReverseTextCommand(CCCommand):

       KEYWORDS = {
           'en': ['reverse'],
           'fr': ['inverse'],
       }

       @authenticated
       def process(self):
           ...do actual work here
   ```

   See *childcount.commands.CCCommand* for the definition of the `childcount.command.CCCommand` class.
4. In `apps/childcount/commands/__init__.py`, add the line:

   ```
   from childcount.commands.ReverseTextCommand import ReverseTextCommand
   ```
5. In your `local.ini` file in the root ChildCount+ directory, add `ReverseTextCommand` to the list of active commands:

   ```
   ...
   [childcount]
   commands = WhoCommand, LookupCommands, ReverseTextCommand, ...
   ...
   ```

Adding a New Form

Say you want to define a new form called *DogsForm* that will record the number of dogs a person has in their household. Users will invoke the SMS form like this:

```
HEALTH_ID +DOGS 2
```

...where HEALTH_ID is replaced by the person's ChildCount+ health identifier and 2 is replaced by the number of dogs that person has in their household.

To define this new form, you must:

1. Look through the existing forms in `apps/childcount/forms` to make sure that the form you want does not already exist. There are lots of useful forms defined there, so please check first.
2. If you want to store the form data in the database (and you probably do), then you will need to create a Django model that represents your report data. Since `DogForm` only takes one parameter – an integer number of dogs, this will be straightforward. You need to create a new model that inherits from the `childcount.reports.CCReport` abstract model.

   To do this, edit the file `apps/childcount/models/reports.py`. At the end of the file, add the code:

   ```
   class DogReport(CCReport):
       class Meta:
           app_label = 'childcount'
           db_table = 'cc_dogreport'
           verbose_name = _("Dog Report")
           verbose_name_plural = _("Dog Reports")

       dog_count = models.PositiveIntegerField(_("Number of dogs"))

   reversion.register(DogReport, follow=['ccreport_ptr'])
   ```

   This defines a new model (i.e., database table) that will store your dog data. This is just standard Django model stuff, so you can consult the Django Documentation for details on how it all works. The only trickiness is that we use django-polymorphic and django-reversion to add some extra features to the models.

   Django-polymorphic allows all models that inherit from `childcount.models.reports.CCReport`` to share common database columns. All reports have an associated `childcount.models.Encounter` and django-polymorphic allows us to declare this relationship only once (in `childcount.models.reports.CCReport`) and all other models get those fields too.

   Django-reversion allows some version control on database tables. We use this to implement the `CANCEL` command (`childcount.commands.CancelCommand.CancelCommand`), which performs an "undo" operation for the previously sent SMS. Django-reversion has high overhead and does not always work properly so we may remove it in the near future.
3. Use South to create a new database migration for this report model. From the command line run:

   ```
   # Change to your CC+ directory
   cd ~/sms
   ./rapidsms schemamigration childcount --auto
   ```

   South should detect the new model and create a migration for it.
4. Create the database table. From your command line, run:

   ```
   # Change to your CC+ directory
   cd ~/sms
   ./rapidsms migrate childcount
   ```
5. Now that the database table for storing your data has been created, you have to define the parsing logic in a *`childcount.forms.CCForm.CCForm`* object. To do this, create a file `apps/childcount/forms/DogsForm.py`

6. Within this new file, import *childcount.forms.CCForm.CCForm* and define a new class that inherits from it:

```
from childcount.forms import CCForm
from childcount.models import Encounter
from childcount.utils import authenticated

class DogsForm(CCForm):

    KEYWORDS = {
        'en': ['dogs'],
        'fr': ['chiens'],
    }

    ENCOUNTER_TYPE = Encounter.TYPE_HOUSEHOLD

    @authenticated
    def process(self, patient):
        ...do actual work here
```

   See *childcount.forms.CCForm* for the definition of the *childcount.forms.CCForm.CCForm* class.

7. In `apps/childcount/forms/__init__.py`, add the line:

```
from childcount.forms.DogForm import DogForm
```

8. In your `local.ini` file in the root ChildCount+ directory, add `DogForm` to the list of active commands:

```
...
[childcount]
forms = PatientRegistrationForm, BirthForm, DogForm, ...
...
```

Indicators

Overview

As described in *What is ChildCount+?*, ChildCount+ collects data, runs analysis on those data, then generates reports and alerts based on the results of the analysis. The indicators functionality takes care of the simple data analysis that goes on inside of ChildCount+.

The data analysis features are centered around the idea of "indicators." An indicator is a function that takes two arguments: a data set and a time period and returns a numerical value. For example, an indicator called "Number of Households" would take a list of patients and a time period as arguments, and would return an integer – the number of households heads in the patient list at the given time period – as output.

The standard interface for indicators provides a few benefits to the programmer:

1. We can transparently cache indicator values.
2. Aggregation is simple: the same indicator function can generate per-patient, per-CHW, per-village, and per-site values, depending on the patient (or other) list you provide as input.
3. It is easy to reuse reporting code across indicators.

The indicators code lives in two places. The definition of the indicators interface is in a library directory: `lib/indicator/` while the ChildCount+ indicator definitions live inside the ChildCount+ application directory: `apps/childcount/indicators/`.

Creating an Indicator

All of the ChildCount+ indicator code resides in the `apps/childcount/indicators/` directory. For the most part, one indicator module (e.g., `childcount.indicators.household` directly corresponds to one ChildCount+ SMS form (e.g., `+V` or the `childcount.forms.HouseholdVisitForm`). To create a new indicator, you must add a new class to one of the files in `apps/childcount/indicators/`.

Warning: Calculating indicator values can be very nuanced and tricky. Please make sure to extensively test your indicator code before you deploy it.

For example, you might want to create an indicator that measures the number of households headed by people over the age of 50 (at the end of the time period). Since this indicator relates most directly to registration (the `+NEW` form), we would put it in the file `apps/childcount/indicators/registration.py`.

Warning: If your indicator returns a percentage value, make sure that your indicator class inherits from `indicator.indicator.IndicatorPercentage`. Using the percentage class will save you lots of time and will make caching your life easier!

Adding a time period

Since all indicators take a time period as an argument, ChildCount+ defines a standard set of time periods we can use for indicator and report generation. Every time period has a start and end `datetime` object, plus one or more sub-periods.

A time period might be "One Year" and the sub-periods might be each of the months of the year. These time periods are defined in `apps/reportgen/timeperiods/definitions/`.

To define a new time period, clone one of the files in `apps/reportgen/timeperiods/definitions/`, edit it to do what you want, and make sure to add it to the list of imports in `apps/reportgen/timeperiods/__init__.py`.

Printed Reports

Overview

Printed report generation is arguably the most important thing that ChildCount+ does. To simplify the process, we have created a report generation "framework" of sorts that makes creating reports faster and easier (once you know how it all works).

One key principle of the framework to reduce the duplication of code by abstracting a lot of the common elements of report generation out of the individual report code. The framework treats each report as a function that takes as arguments:

1. A time period (e.g., "12 Months (divided by Quarter)")
2. A file format (e.g., "pdf")
3. [Optional] A variant (e.g., "Bugongi Health Center")

In the definition of the report, the report author only has to write a function that takes these three parameters and spits out a report with the desired properties. The report framework handles the user interface for generating reports and manages the storage as well. Look at the existing reports to see how this all is done.

How to Add Reports

1. Create a report definition file and place it in `apps/reportgen/definitions`. Your best bet is to copy an existing report file and adjust it according to your needs.
2. If your report is called `NewReport.py`, then add `NewReport` to the `__all__` list in `apps/reportgen/definitions/__init__.py`.
3. Open the Django admin interface (at http://your_server_ip/admin), and add an entry to the Reportgen.Report model. The entry should contain the name of your report class (e.g., `NewReport`) and a human-readable title for the report.
4. Restart `celeryd`, `rapidsms` and `celerybeat`.

How to Debug Reports

Debugging reports can be annoying. The one way to simplify the process is this:

```
# Open shell
cd sms
./rapidsms shell

# Load a time period
from reportgen.timeperiods import Month
t = Month.periods()[2]

# Load your new report definition file
from reportgen.definitions.MyNewReport import ReportDefinition

# Test the report using the first variant
ReportDefinition.test(t, 'pdf')

# The generated report will end up in /tmp/test_my_new_report
# (or whatever the name of your report is)

# To re-run your report, you need to quit the shell
exit
```

Save this script in a file to speed up the process.

ccdoc - ChildCount+ Document-Generation Library

The ccdoc library adds another layer of abstraction to reports. Using ccdoc, you can generate an instance of a report as a ccdoc `ccdoc.document.Document` object and ccdoc will handle generation of the report output in HTML, PDF, and XLS file formats. Many of the reports in `apps/reportgen/definitions` use ccdoc to simplify the report generation process and those are good places to look for real-world examples of ccdoc in action.

The file `lib/ccdoc/example.py` contains an example of how to create and render a Document into HTML. Run the example from your rapidsms shell (when you're in your site directory) like this:

```
cd ~/your-site
./rapidsms shell < lib/ccdoc/example.py > test.html
# Ignore the >>> prompts that the python shell adds
```

How It Works

The user creates a `ccdoc.document.Document` object, which is pretty much a collection of `ccdoc.section.Section`, `ccdoc.paragraph.Paragraph`, and `ccdoc.table.Table` objects. The user then passes the `Document` to an object that inherits from `ccdoc.generator.Generator` (for example, `ccdoc.html.HTMLGenerator`). The `Generator` object prints the formatted report to a file – you can either ask for the name of the file where the report has been printed or you can get the contents of the file back as a string.

You can create new `Generator` objects that inherit from `Generator`. If you do, make sure to look in `lib/ccdoc/generator.py` at the bottom and to implement `_*_document()` and the `_render_*()` methods in your subclass.

See the *ccdoc* page for API information.

[U] Dashboard

Adding an element to the dashboard

Development Processes

CHAPTER 4

Development Processes

Bug Tracker

You can find our bug tracker at: http://code.mvpafrica.org/.

Mailing List

We do not have a public developer mailing list (yet). You can reach the ChildCount+ developers on our internal mailing list at dev at mvpafrica dot org.

Repo and Branches

Repositories

The ChildCount+ code repository is online here:

> http://www.github.com/mvpdev/rapidsms

The installation and server configuration files for ChildCount+ installations are online here:

> http://www.github.com/mvpdev/rapidsms-impl

Stages and Versions

Different versions of the ChildCount+ code often require different server configurations. We refer to each successive server configuration as a "stage" while each release of the source code is a "version." We use letters (A, B, C, ...) to refer to stages, and numbers (0, 1, 2, ...) to refer to versions. The version numbers reset with each stage.

So, if the latest code is `stageE-v1` and a developer creates a release that does not require changes to the server configuration, the next release will be `stageE-v2`. Afterwards, if the next release does require changes to the configuration files, the release tag will be `stageF-v0`.

Branches

The ChildCount+ code repository has two two main branches: ccdev and ccstable.

If a software developer wants to create a new ChildCount+ feature, the process generally goes like this:

1. Developer creates a new branch from `ccdev`, let's call this branch `twitter` and pretend that it adds some Twitter functionality to ChildCount+.
2. Developer does the work on the `twitter` branch, merging changes from `ccdev` into `twitter` to make sure that `twitter` stays current.
3. When the developer is satisfied that the Twitter feature works as hoped, she merges `twitter` into the `ccdev` branch.
4. Every few months, the release developer merges all of the changes from `ccdev` into `ccstable`. The release developer tests all of these features and makes sure that the translations and localization works properly for French- and Tigrinya-speaking sites.
5. The release developer adds a git tag of the format `stageX-vY` (where `X` is the stage code and `Y` is the version number) to the `ccstable` branch.
6. If a new stage is being created: The release developer writes up a textual description of how to update the server to accommodate the new code and posts it on http://we.mvpafrica.org.
7. The release developer pushes the `ccstable` code out to the sites. In MVP lingo, this is a "stage".

Documentation

This documentation is hosted on GitHub and is created using Sphinx.

We develop the documentation on the `ccdev` branch, then built HTML documentation files into the root of the `gh-pages` branch. The `gh-pages` branch is a special branch for the GitHub Pages feature. Files pushed there end up being served at http://mvpdev.github.com/rapidsms/. The documentation mirror at http://docs.childcount.org/ should copy the documentation from http://mvpdev.github.com/rapidsms/ after every commit to the GitHub repository.

Who to Contact

The best way to get in touch with the ChildCount+ Developers is to look for the contact information listed on http://www.childcount.org/. Those contacts are the most likely to be up to date. You can also catch us at @childcount.

Misc.

CHAPTER 5

API/ChildCount+ Module Reference

Warning: As of 30 June 2011, the version of our documentation hosted on ReadTheDocs does not display our API documentation properly. Instead, please see the full API documentation at http://docs.childcount.org/ or http://mvpdev.github.com/rapidsms/.

apps

bonjour

bonjour.app

bonjour.dates

bonjour.ethiopian_date

Note: This version of the Ethiopian Calendar tool is a fork from Renaud's version. The only difference is that when converting a date to the Julian calendar, this version returns a (year, month, day) tuple instead of a `datetime.datetime` object.

We cannot use the native Python `datetime` class because the Julian calendar has 13 months, and the Python libraries can only deal with 12 months.

Ethiopian Calendar tool for Python 2.6

Copyright (c) 2010 Renaud Gaudin <rgaudin@gmail.com>

This tool is a python port of Java Code from Ealet 2.0 by Senamirmir Project.

This code is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

class `bonjour.ethiopian_date.`**`EthiopianDateConverter`**

Class methods for converting between Ethiopian and Gregorian

classmethod **`date_to_ethiopian`**(*adate*)

Ethiopian date object representation of provided Gregorian date

Shortcut to to_ethiopian() classmethod using a date parameter

Parameters **`adate`** (`datetime.date`) – Gregorian date to conver to Julian calendar

classmethod **`to_ethiopian`**(*year*, *month*, *date*)

Ethiopian date object representation of provided Gregorian date

Parameters

- **`year`** (`int`) – Gregorian year
- **`month`** (`int`) – Gregorian month
- **`date`** (`int`) – Gregorian day

classmethod **`to_gregorian`**(*year*, *month*, *date*)

Gregorian date object representation of provided Ethiopian date

Parameters

- **`year`** (`int`) – Julian year
- **`month`** (`int`) – Julian month
- **`date`** (`int`) – Julian day

bonjour.utils

childcount

childcount.commands

Please see *with ChildCount+* for information on what forms and commands are.

childcount.commands.CCCommand

Pre-Defined Commands

There are a variety of commands already defined in `apps/childcount/commands`. Consult them before writing a new command.

childcount.dashboard_sections

childcount.exceptions

childcount.fields

childcount.forms

Please see *with ChildCount+* for information on what forms and commands are.

childcount.forms.CCForm

class `childcount.forms.CCForm.`**`CCForm`**(*message*, *date*, *chw*, *params*, *health_id*)

An abstract class to hold the logic for an SMS form.

Parameters

- **`message`** (`rapidsms.Message`) – SMS message being processed by this form
- **`date`** (`datetime.datetime`) – Encounter date of this form
- **`chw`** (`childcount.models.CHW`) – CHW who submitted this form
- **`params`** (*`list`*) – Parameters passed to this form (as in `sys.argv()`)
- **`health_id`** (*`str`*) – Health ID for the encounter's patient

`MULTIPLE_PATIENTS` **= True**

`PREFIX` **= '+'**

The character prefix that should precede the form keyword. We use "+" everywhere to keep things standardized.

`post_process`(*forms_list*)

Processing to be done *after* all *`process()`* has been called on all submitted forms.

Parameters **`forms_list`** (list of instantiated *`childcount.forms.CCForm`* objects) – List of successfully processed forms

`pre_process`()

Processing to be done by this form *before* the patient's health ID is validated.

This method used primarily for patient registration – when the health ID is not valid until the registration has completed. See `apps/childcount/forms/PatientRegistrationForm.py` for an example.

`process`(*patient*)

Processing to be done by this form once the encounter patient has been identified. Most forms implement their validation and DB logic here.

Pre-Defined Forms

All ChildCount+ forms are defined in :file:'apps/childcount/forms/'. Before writing a new form, check to see if someone has already written a form that suits your application.

childcount.helpers

PLEASE DO NOT PUT INDICATOR LOGIC HERE!!! The helpers module is *only* for reusable logic that:

1. Cannot be elegantly expressed in terms of a `childcount.indicator.Indicator`.
2. Are used in many different reports or forms.

Any function that operate on lists of `childcount.models.Patient` objects, or that involves DB heavy lifting should probably be written as a `childcount.indicator.Indicator`. That way you get the benefits of caching and the standardized `childcount.indicator.Indicator` interface.

childcount.helpers.chw

childcount.helpers.patient

childcount.helpers.site

childcount.indicators

Please see *Indicators* for details on the indicators interface and for information on how to define a new indicator. Defines all of the indicators used throughout ChildCount+. Look at the files in `apps/childcount/indicators/` for definitions.

childcount.models

childcount.models.Patient

childcount.models.reports

Hint: See *Adding a New Form* for information on how these `CCReport` objects relate to `childcount.forms.CCForm` and the database.

childcount.tasks

childcount.utils

libreport

Libreport is the legacy code we use to generate some PDF and CSV reports. The PDF code wraps the ReportLab PDF generation library.

mgvmrs

Mgvmrs contains tools for pushing ChildCount+ data into OpenMRS using the XForms module.

If you need this functionality, just take a look at the code in `apps/mgvmrs`.

reportgen

lib

ccdoc

checksum

checksum

Used for validating the Luhn-30 checksums used with the OpenMRS and ChildCount+ health IDs.

`checksum.checksum.`**`BASE_CHARACTERS`** **= u'0123456789acdefghjklmnprtuvwxy'**
Valid characters for health IDs

exception `checksum.checksum.`**`CheckDigitException`**
Raised on invalid input to checksum methods

`checksum.checksum.`**`clean_chars`**(*chars*)
Remove whitespace

`checksum.checksum.`**`get_check_digit`**(*identifier*, *base_chars*)
Compute the check digit for an identifier using the specified legal characters.

Parameters

- **`identifier`** (`unicode`) – String on which to compute the checksum
- **`base_chars`** (`unicode`) – String of legal characters

`checksum.checksum.`**`is_valid_identifier`**(*identifier*, *base_chars=u'0123456789acdefghjklmnprtuvwxy'*)
Use the Luhn-30 checksum to validate a string

Parameters

- **`identifier`** (`unicode`) – String whose validity should be checked
- **`base_chars`** (`unicode`) – String of legal characters

Returns `bool`

indicator

indicator.cache

indicator.indicator

indicator.percentage

indicator.query_set_type

CHAPTER 6

Frequently Asked Questions

Contents

How can I copy the MySQL database from the server to my local machine?

Run the following commands in a local shell/terminal:

```
# Connect to the CC+ server, replacing
# CC_SERVER_IP with the IP address of your
# ChildCount+ server
ssh mvp@CC_SERVER_IP

# Dump CC+ database to a file called "childcount_dump.sql"
# in the mvp home directory
mysqldump -u childcount -pchildcount childcount > ~/childcount_dump.sql

# Quit SSH connection to server
exit

# Now you are on your local machine.
# Copy the SQL file from the server to your
# local machine.
scp mvp@CC_SERVER_IP:~/childcount_dump.sql ~/childcount_dump.sql
```

```
# Load the file into your development database
mysql -u childcount -pchildcount childcount < ~/childcount_dump.sql
```

That's it!

How can I update the translations for my language?

Each app is translated independently, but for ChildCount+ to work, all apps should be translated. The ChildCount+ apps are listed in *apps*.

Here is how you update the translations from an Ubuntu machine:

```
# Make sure you have poedit installed
sudo apt-get install poedit

# Change to the directory of the app that you want to
# translate. For example, if ChildCount+ is installed in
# ~/sms, here is how you translate apps/childcount:
cd ~/sms/apps/childcount

# Make sure that you're on the development branch
git checkout ccdev

# Make sure that the locale directory exists
mkdir locale

# Update message file with new untranslated strings.
# Replace "fr" with the two-letter code for your
# language.
django-admin.py makemessages -l fr -e html,json,py

# Edit the .po file for your language. Replace "fr"
# with the two-letter code for your language.
poedit locale/fr/LC_MESSAGES/django.po

# After saving the .po file, compile the translations.
django-admin.py compilemessages

# Add the files to git and commit them.
git add locale
git commit -m "New translations"

# Push new files to the repository
git push
```

CHAPTER 7

History / Credits

History

Warning: This history is probably incorrect. It is based on one ChildCount+ team member's fuzzy recollection of conversations with other team members.

ChildCount+ v1 came out some time in 2009. Matt Berg along with David Gelvin, Dickson Ukanga, and Renaud Gaudin deployed the software at the Millennium Villages Project Sauri site. The v1 software was a sort of Twitter clone for community health workers – they could message each other and send broadcast messages as well. There were some features for recording patient health data as well.

This same team started to develop ChildCount+ v2 in 2010. This second version of ChildCount+ (the current version as of June 2011), developed the health data collection features and added report generation functionality as well.

MVP deployed the v2 software at Sauri in early 2010 and at other MVP sites in mid-2010. Sauri's community health workers were able to interface with the ChildCount+ system from the v1 deployment onwards. Community health workers at most other MVP sites still (as of June 2011) interact with ChildCount+ via paper forms.

Credits

Current and Past Code Contributors (culled from git logs and in alphabetical order):

- Aboubacar Diarra
- Alex Dorey
- Alou Dolo
- David Gelvin
- Delphia Polle
- Dickson Ukanga

- Elaine Stampfer
- Henry Corrigan-Gibbs
- Ibrahima Fadiga
- Matt Berg
- Moses Katembu
- Renaud Gaudin

Please let us know if you are mising from this list! For a more complete list of people involved with ChildCount+ development and deployment, please see the ChildCount+ Team page.

This documentation was created by Henry Corrigan-Gibbs in 2011 with the goal of letting others benefit from the hard work of the many ChildCount+ contributors.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index