

SAHANA EDEN

Published : 2014-06-08
License : None

INTRODUCTION

1. ABOUT THIS BOOK
2. WHY SHOULD YOU USE SAHANA EDEN?
3. WHAT DOES SAHANA EDEN DO?
4. WHO USES SAHANA EDEN?
5. TECHNICAL OVERVIEW
6. PLANNING A DEPLOYMENT

1. ABOUT THIS BOOK

Everyone can be affected by disasters whether personally or indirectly. Disaster Management professionals are kept busy responding to events of all scales, yet find time to explore the use of more sophisticated toolsets. The Sahana Project was conceived by people on the front lines of the 2004 Sri Lanka tsunami in order to coordinate the rescue efforts. The word "sahana" means "relief" in Sinhala. This has evolved to provide solutions to both prepare for and respond to disasters no matter where they happen.

Sahana Eden is an open source software platform which provides a range of solutions for Disaster Management practitioners to help them reduce the impact disasters have on our communities through tracking the needs of the affected populations & coordinating the responding agencies & their resources.

The latest version of this book is available to either read online or order a printed copy from <http://bit.ly/sahanaedenbook>

WHO IS THIS BOOK FOR ?

This book has been imagined to meet the needs of three kinds of persons: 1) Decision Makers looking for an appropriate solution for disaster management; 2) Deployers who are ready to deploy Sahana Eden; and 3) Developers who are extending Sahana Eden for more specialized solutions or want to contribute to the project.

Decision Makers

Preparing and planning for crisis and disaster scenarios is an important part of every community. Whether it is at a global, national, regional or local level, leaders who understand the complexities of disaster response make better decisions and can quickly respond to changing situations. Decision Makers should read the "Introduction" section. This content covers the Sahana Eden platform overview, capabilities and selected case studies. This material will help decision makers in strategic planning and give them important insight into the deployment process.

Deployers

Those who are thinking about or who are ready to deploy Sahana Eden should read the "Introduction", "Getting Started" and "Administration" sections.

In order to deploy Sahana Eden, a deployer should be comfortable with the following system administration processes:

- Installation of an operating system (such as Debian Linux)
- Command line usage (for package installation)
- System configuration involving editing of text files

Developers

Developers should read the entire book as they should have an overview of how the software can be used and will need to maintain their local deployment.

Basic customization doesn't require any more skills than those required for a Deployer, however, more advanced development will require being familiar with or learning the following skills:

- Python
- JavaScript
- Cascading Style Sheets (CSS)

2. WHY SHOULD YOU USE SAHANA EDEN?

Sahana Eden is an open source software platform which has been built specifically for Disaster Management. It is highly configurable so that it can be used in a wide variety of different contexts and is easy to modify to build custom solutions. Different levels of support are available from both the voluntary Sahana Eden community and professional companies.



Training of Teachers of Pune University on Disaster Management

BUILT FOR DISASTER MANAGEMENT

Sahana Software was initially developed by members of the information technology (IT) community in Sri Lanka to provide solutions for the relief effort following the 2004 Indian Ocean Tsunami. Sahana Eden is the latest evolution of this software and provides a solution to manage organizations, people, projects, inventory and assets as well as collecting information through assessments and providing situational awareness through maps.

Sahana Eden can be accessed from the web or locally from a flash drive, allowing it to be used in environments with poor internet. Local & Web versions can be configured to synchronize to allow data to be shared between them.

These features are designed to help Disaster Management practitioners to better mitigate, prepare for, respond to and recover from disasters more effectively and efficiently. Sahana Eden can provide valuable solutions for practitioners in Emergency Management, Humanitarian Relief and Social Development domains.

COMMUNITY AND PROFESSIONAL SUPPORT

Sahana Eden is a project of the Sahana Software Foundation, whose mission is:

To help alleviate human suffering by giving emergency managers, disaster response professionals and communities access to the information that they need to better prepare for and respond to disasters through the development and promotion of free and open source software and open standards.

The project is supported by a voluntary community of Disaster Management practitioners, students, academics and companies. This community is able to provide a basic level of support to help you deploy and configure Sahana Eden. There are also companies, such as AidIQ, who provide professional services to customize and support Sahana Eden.

HIGHLY CONFIGURABLE AND EASY TO MODIFY

Sahana Eden is designed to be rapidly configured and customized to support the diverse business processes used within Disaster Management. Sahana Eden's modular design allows different pieces of functionality to be enabled and disabled as required providing flexible solutions for changing contexts.

The application can be configured to secure sensitive information, while also making data which needs to be shared available in a variety of different formats including Microsoft Excel and PDF. To ensure that Sahana Eden is accessible to every country, it can be translated into multiple languages.

Finally, Sahana Eden is licensed under the Open Source MIT License (<http://www.opensource.org/licenses/mit-license.php>), making it free to download, customize and modify without any restriction or reliance on any single vendor. The MIT License also places no restriction on the commercial or closed deployments of the software, giving the greatest flexibility for the use of Sahana Eden in sensitive environments.

3. WHAT DOES SAHANA EDEN DO?

Sahana Eden contains a number of different modules which can be configured to provide a wide range of functionality. This chapter gives a brief summary of the core modules and outlines how Sahana Eden can meet some of your needs.

ORGANIZATION REGISTRY

Many diverse organizations are involved in Disaster Management, from responding to disasters to strengthening communities to providing support to people in need. Sahana Eden's Organization Registry can track what organizations are active in different contexts, providing opportunities for collaboration and coordination. After the 2010 earthquake in Haiti, Sahana Eden managed a list of 696 organizations who were all providing assistance to the affected population. This included Government Departments, Non-Governmental Organizations (NGOs), United Nations (UN) Agencies and Corporations.

The Organization Registry also allows organizations to record their Offices, Warehouse and Field Sites including their locations so they can be mapped as well as links to other modules such as Human Resources, Assets and Inventory.

	Name	Acronym	Type	Sector	Home Country	Website
Open Delete	American Red Cross of Greater Los Angeles	ARC	Red Cross / Red Crescent	Health Care and Social Assistance	United States	http://redcrossla.org
Open Delete	CERT Los Angeles	CERT	National NGO	Public Administration	United States	http://www.cert-la.com
Open Delete	Disaster Healthcare Volunteers	DHV	National NGO	Health Care and Social Assistance	United States	http://www.lacountydhv.org
Open Delete	LA Works		National NGO	Public Administration	United States	http://www.laworks.com
Open Delete	Los Angeles Emergency Management Department	EMD	Government	Public Administration	United States	http://www.updateala.com
Open Delete	Salvation Army		International NGO	Health Care and Social Assistance	United States	http://salvationarmy.org
Open Delete	Volunteer Center of Los Angeles	VCLA	National NGO	Other Services	United States	http://www.vcla.net

PROJECT TRACKING

By telling you *Who's Doing What, Where, and When*, Sahana Eden provides a valuable tool to help organizations responding to disasters know where the greatest needs are and coordinate with others who are engaged in similar work.

The Disaster Risk Reduction (DRR) DRR Project Portal (www.drrprojects.net) uses Sahana Eden to provide a coalition of organizations a platform to share information on what projects they are engaged in within the Asia Pacific region. There is information on at least 1250 projects which is publicly available to communities, stakeholders and decision makers to facilitate cooperation and planning and to identify gaps and overlaps.

	Climate Change	Complex Emergency	Early Warning	Environment	GIS & Mapping	Recovery	Total
Cambodia	15	3 <ul style="list-style-type: none"> United Nations Economic and Social Commission for Asia and the Pacific United Nations Office for the Coordination of Humanitarian Affairs Mekong River Commission 	14	4 <ul style="list-style-type: none"> United Nations Economic and Social Commission for Asia and the Pacific Mekong River Commission Korea International Cooperation Agency World Bank 	10	5	25
Lao People's Democratic Republic	12	3	13	6	9	5	19
Thailand	15	3	14	10	7	7	24
Viet Nam	18	3	18	7	11	6	30
Total	28	3	25	13	14	12	

HUMAN RESOURCES

The most important part of Disaster Management is the people. Whether they are community volunteers or staff working for different organizations, the Human Resources module can help manage the people involved. It will track where they are, what skills they have and help ensure that everyone is effectively engaged with the work that needs to be done.

Sahana Eden can also be used to provide a contact list to ensure that the right people can be contacted at the right time.

The Associação Portuguesa dos Bombeiros Voluntários (APBV) - the Portuguese National Volunteer Fighting Association - use Sahana Eden to help manage their various teams including tracking their experience, training and evaluations as a solution for managing credentialing of their volunteers.

Search for Staff or Volunteers

Name: ? HELP

[Advanced Search](#)

[Save Search](#)

Matching Records

Show entries Showing 1 to 2 of 2 entries

	Person	Job Title	Type	Facility
Details	Amy Grant		staff	Santa Clarita Valley District Office (Office)
Details	Amy Barker		staff	EMD Main Warehouse (Warehouse)

Showing 1 to 2 of 2 entries

First Previous **1** Next Last

INVENTORY

Whether organizations are supplying basic essentials of life to people affected by natural disasters or giving communities the tools they need to restore their livelihoods, Sahana Eden can be used to manage inventories of items and match requests for items with warehouses and other facilities which have them available. Operationally, Sahana Eden can be used to record and automate transactions for sending and receiving shipments. Sahana Eden can support multiple Catalogs of Items as well as providing alternative items to ensure more effective use of supplies. Sahana Eden can be configured to load multiple catalogs including a generic list of items and/or the IFRC Emergency Item Catalog.

The HELIOS Foundation is utilizing Sahana Eden to allow different Humanitarian NGOs to share inventory information to improve operational efficiency by facilitating the utilization of surplus items and coordinating procurement of new items.

Warehouse Details

Name: EMD Main Warehouse Type: Warehouse
 Organization: Los Angeles Emergency Management Department Location: [EMD Main Warehouse](#)
 Email: None Telephone: None

Basic Details | Staff | Requests | Match Requests | Commit | **Inventory Items** | Incoming | Receive | Send

[Add Item to Inventory](#)

Inventory Items

Search: Show 10 entries Showing 1 to 10 of 14 entries

	Item	Pack	Quantity	Value per Pack	Currency	Expiry Date	Comments
Open Delete	Body Bags (ea)	-	5000.0	None	1	-	
Open Delete	Bolt Cutters (ea)	-	75.0	None	1	-	
Open Delete	Canned/Dried Food (lbs)	-	75000.0	None	1	-	
Open Delete	Diapers (dtz)	-	2000.0	None	1	-	
Open Delete	Gasoline (gal)	-	9000.0	None	1	-	
Open Delete	ID Badges (ea)	-	12000.0	None	1	-	
Open Delete	Infant Formula (lbs)	-	8000.0	None	1	-	
Open Delete	Non-Potable Water (gal)	-	80000.0	None	1	-	
Open Delete	Potable Water (gal)	-	20000.0	None	1	-	
Open Delete	Radio (ea)	-	50.0	None	1	-	

Showing 1 to 10 of 14 entries

ASSETS

A wide range of assets are needed to respond to disasters, including vehicles to transport people and relief items, radio equipment to provide communication where telecommunication infrastructure has been destroyed, and generators to provide backup power. Sahana Eden is able to manage assets, track where they are, who they have been assigned to, and what condition they are in. This ensures that assets are used effectively and efficiently.

The Resource Management System is a Sahana Eden instance deployed by the International Federation of Red Cross and Red Crescent Societies (IFRC) to provide visibility on the assets of national Red Cross and Red Crescent Societies. This prepares the Red Cross movement to respond more effectively to disasters with the assets they need.

Asset Details

Asset Number: 1001 Item: [Generator](#)
 Condition: Minor Damage Status: Assigned
 Person: Frank Boone Facility: [EOC \(Office\)](#) Location: -

[Set Base Site](#) | [Return](#) | [Assign to Person](#) | [Assign to Site](#) | [Assign to Organization](#) | [Update Status](#)

[Edit Details](#) | [Log](#) | [Documents](#)

Asset Log

Search:

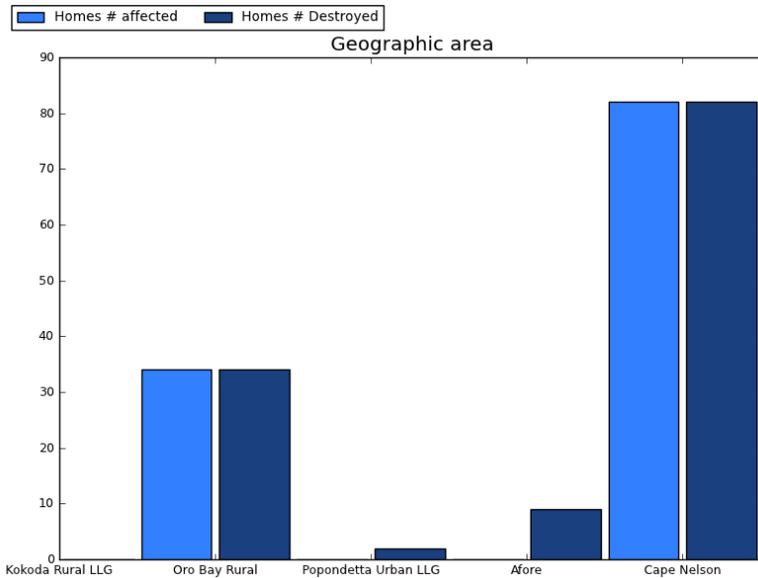
	Status	Date	Date Until	Organization	Facility or Location	Facility	Room
Open	Base Site Set	2009-08-14 17:00:00	-	American Red Cross (ARC)	Site	Head Office (Office)	Garage
Open	Assigned	2011-10-19 21:32:51	-	-	Site	EOC (Office)	-

Showing 1 to 2 of 2 entries

ASSESSMENTS

Sahana Eden can be used to collect and analyze information from assessments to help organizations more effectively plan their disaster management activities. Different assessment templates can easily be designed and imported into Sahana Eden to support assessments for different organizations in different contexts. Data can either be entered into an interactive web form or imported via an Excel template.

In order to help decision makers effectively use the information collected in assessments, Sahana Eden provides a range of analysis including custom reports, graphs and maps.



SCENARIOS & EVENTS

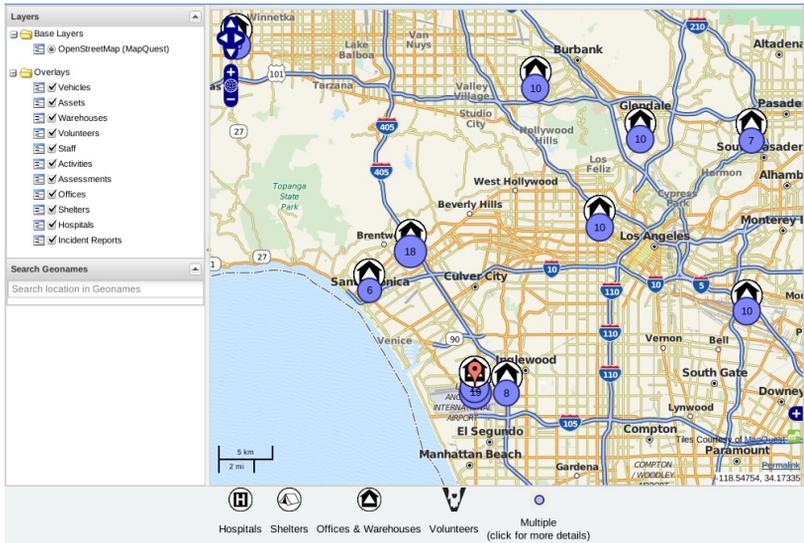
To help organizations better plan for disasters, Sahana Eden can be used to plan for different scenarios, including recording what human resources, assets, facilities and tasks will be needed to effectively respond.

When an incident occurs events can be created from a scenario template to allocate the resources and alert people of the need to respond.

MAP

"Some people need to see a map before they can even start having a conversation about the data." - Sahana Eden User

Sahana Eden has fully integrated mapping functionality which allows any location-based data to be visualized on a map. This information can also be searched using a map-based boundary selection. Maps provide situational awareness which is essential when either planning to prepare for or respond to a disaster. Sahana Eden supports many standard formats for overlaying data on maps from other sources and Geographical Information Systems (GIS), for example natural hazard risks, population or weather.



SHELTER MANAGEMENT

When disasters are widespread and result in population displacement, understanding and tracking the landscape of shelters - and the people in them - is a critical activity. The Shelter Registry provides functionality to list and track information on shelters and on the people arriving and departing. Shelter details include location, services provided, responsible organization and contacts, demographics, and needs. In addition, individual person data includes name, age, relatives, status, health, and many other details to provide a clear understanding of population demographics within the site.

MESSAGING

In the complex domain of Disaster Management, communication is critical. Sahana Eden provides support for messages to be sent by Email, SMS, Twitter and Google Talk. Distribution Groups can be set up to allow messages to be easily sent to many people at once. Users are able to search for specific information and subscribe to receive update messages when new information is added.

Interactive messages can also be set up to allow people to send short message queries to Sahana Eden and receive automatic responses.

4. WHO USES SAHANA EDEN?

Sahana Eden is used by many diverse organizations throughout the world to assist the response to traumatic events such as natural disasters. From hosted deployments by Foundation Team members to on-site deployments within organizations, Sahana Eden's versatility is demonstrated in some of the following case studies and use cases for the software. While some first responders are coming to assist after an earthquake, others are attempting to reduce risk by gathering information and helping to network and raise awareness prior to a disaster. Coordination of resources, understanding the inventory of available resources, raising awareness, and providing early warning all reduce risk and empower responses that can literally save lives. Here are some stories about deployers and responders using Sahana Eden.

APBV - PORTUGUESE VOLUNTEER FIREFIGHTERS

With little to no budget, the president of the Associação Portuguesa dos Bombeiros Voluntários (APBV), was seeking better solutions to manage their limited resources. In the past they had tried proprietary software based solutions which were not maintainable and did not address their needs. These solutions were costly and did not offer a sustainable method for data management or migration.

After seeing a demonstration of Sahana Eden at the Information Systems for Crisis Response and Management (ISCRAM) conference in Lisbon, APBV deployed the Human Resources module to help manage personnel and are planning to deploy the vehicle management system and connecting their GPS-enabled Tetra radios to Sahana Eden's mapping capabilities. They hope that this will become the national standard for disaster planning and crisis management for all of Portugal.

DISASTER RISK REDUCTION PROJECT PORTAL

www.drrprojects.net

The Disaster Risk Reduction (DRR) Project Portal collects information on all multi-country and national level DRR projects and initiatives in the Asia Pacific region implemented since 2005. By facilitating information sharing across the region, the Portal aims to advance the [Hyogo Framework for Action \(HFA\)](#) goals in building the resilience of nations and communities to disasters. This includes making disaster risk reduction a high priority on local and national levels. There are many priorities for action, such as knowing the risks, enhancing early warning systems to reduce vulnerabilities, and building a culture of safety and resilience for all people by strengthening networks and working with the media.

This Sahana Eden-powered portal was deployed at the Asian Disaster Preparedness Center.

The Project Portal:

- Helps effective planning, programming, cooperation, and collaboration of DRR projects and programs in the region by facilitating project analysis to identify gaps and overlaps.
- Is essential for governments, organizations and donors involved in implementing and supporting DRR projects and program in the region.
- Is a useful resource for academics, students and the media for obtaining an overview of DRR projects being implemented in the region.

HAITI 2010 EARTHQUAKE RESPONSE

We all recall the awful event in early 2010 when an earthquake hit the town of Léogâne, near Port of Prince, Haiti. The loss of life, building, and damage estimates were shocking but trained response organizations went into action immediately.

Anticipating the need for overall organization coordination, the Sahana Software Foundation deployed a public emergency response portal site using the Sahana Eden software. The site was hosted at <http://haiti.sahanafoundation.org> and the community managed access for registered users - those from charitable organizations, government agencies and educational institutions were given create/edit permissions to the site; while most of the data remained publicly available (read access) excluding sensitive information (such as personal contact information for agency staff). The site went live the day of the disaster.

In the first 48 hours after the earthquake, responders wanted to know who else was responding, what organizations already had staff in Haiti that could assist, where were they located, and what assets and resources they had available to them. To meet this need, Sahana's Organization Registry (OR) tracked organizations and offices working on the ground in Haiti. The Organization Registry provided a searchable database of organizations responding to the disaster, the sector where they are providing services, their office locations, activities and their contact details. The Sahana database became one of the primary repositories of organization, office and contact information for the relief operation during the first couple of weeks of the response. Organizations were encouraged to self-register and report their office locations or to simply send the Sahana team an e-mail indicating their office locations. Volunteers entered data from pre-disaster lists of organizations working in Haiti available from United Nations Office for the Coordination of Humanitarian Affairs (UN OCHA), as well as active contact lists used by United Nations Disaster Assessment Coordination (UNDAC), InterAction and other sources with official and accurate points of contact.

During the second week of the relief operation, requests came from all directions seeking to identify the location and operating status of hospitals and medical facilities within Haiti. Sahana Eden's Hospital Registry organized a volunteer effort to geo-locate approximately 100 hospitals with names and known coordinates over a 24-hour period. The results of this effort added over 160 hospitals to the Sahana Hospital Registry that had been set up to manage medical and health facility capacity and needs assessment. Because avoiding overcrowding and ensuring medical personnel and equipment availability is crucial to its success, this registry was designed to be compliant with the OASIS EDXL-HAVE interoperability standard that provides a schema for tracking hospital capacity and bed availability data during emergencies. A KML feed built from Sahana's hospital location data provided a visual customizable display of geographic data in Google Earth. This feed remained the most accurate and complete source of operating hospital facilities throughout the first two months of the relief operation and was accessed by thousands of users world-wide.

The technology community's response to the Haitian earthquake was an unprecedented collaborative and cooperative effort on the part of different organizations to come together and to help each other and to not replicate efforts. The Sahana Software Foundation team worked from outside Haiti to deploy and manage the infrastructure being used by local and international responders and contributors.

INTERNATIONAL FEDERATION OF RED CROSS AND RED CRESCENT SOCIETIES

The International Federation of Red Cross and Red Crescent Societies (IFRC) developed a Resource Management System using Sahana Eden. This allows their National Societies to share information on their Inventory, Assets, Staff and Volunteers. Neighboring National Societies and the IFRC can quickly see what is available in the event of a major disaster. This information is blended with data from other Geographic Information Systems (GIS), such as Population Density, Rainfall and Topography to allow for a more informed planning of the response.

The solution allowed agencies to share a common server, yet retain full control over their data and who can have access to it (i.e. a multi-tenancy system). The open source nature of the software was important because it meant there was no vendor lock-in and the software was easy to maintain. For this deployment the Sahana team deployed using Amazon Web Services in the regional data center to guarantee low latency.

HELIOS SUPPLY CHAIN AND INVENTORY SHARING

The HELIOS foundation (helios-foundation.org) in the United Kingdom wanted to create a portal to allow Humanitarian Non-Governmental Organizations (NGOs) working in the field to be able to track and share information on their inventory of relief items. The system is planned to allow them to use up surplus items, avoid items expiring and avoid duplicate procurement. Data can be automatically uploaded from HELIOS instances or via manual data entry or uploading of spreadsheets.

Sahana Eden was chosen to allow various methods of data entry as well as to scale for use by other NGOs. Future iterations will include sharing of information on planned procurements to allow bulk discounts shared freight movement and import costs.

This work was done for the Consortium of British Humanitarian Agencies and funded by the Department for International Development (DfID), part of the UK government.

5. TECHNICAL OVERVIEW

Because Sahana Eden needs to be accessible to users at remote locations, including the public, a browser-based solution was essential. The system also needs to be able to be used on offline laptops, so it needs to run on a lightweight stack.

Python was selected as a suitable high level language allowing the rapid customization of code required for each individual circumstance yet has a large number of powerful libraries available including for Geospatial Information Systems (GIS).

Sahana Eden includes tools for synchronization between multiple instances, allowing for responders or district offices to capture data on victims in the field and exchange the data with other offices, headquarters or responders in the field.

SAHANA EDEN FRAMEWORK

The Sahana Eden Software Platform has been built around a Rapid Application Development (RAD) Framework. This provides a high level of automation to ensure that new solutions can be quickly and effectively developed. Once a database table is defined, the Sahana Eden Framework automatically generates HTML pages to handle CRUD (Create, Read, Update, Delete) as well as Search, Map and Pivot Reports. Web Services are available to import and export in XML, CSV, JSON and EXTensible Stylesheet Language (XSL) transforms are supported to produce other data standards.

The Sahana Eden Framework has flexible authorization policies which can be configured to grant permissions for different modules, tables as well as the ability to have multiple Organizations control their own data on a single Sahana Eden installation.

Sahana Eden can be downloaded and run locally from a flash drive. Synchronization functionality allows data to be entered then keep up to date between different installations, including online servers and local flash drive installations. The Sahana Eden Framework also includes a scheduler for running tasks at a specific time, in regular intervals or as asynchronous tasks which are triggered by users.

SAHANA EDEN ARCHITECTURE

The basic Sahana Eden architecture is as follows:

Web Server	Apache	Other web servers can also be used, such as Cherokee.
Application	Sahana Eden	
Web Application Framework	Web2Py	
Programming Language	Python & Java Script	
Database	MySQL, PostgreSQL, or SQLite	MySQL, PostgreSQL, and SQLite are supported. Other databases should be usable without major additional work since Web2Py supplies many connectors.

Operating System	Linux (Debian recommended)	Windows and Mac OS X are possible, but only recommended for single-user environments.
-------------------------	----------------------------	---

6. PLANNING A DEPLOYMENT

A successful deployment of Sahana Eden must consider the people and business processes involved as well as the technology. This chapter will provide guidance on how to engage with the necessary people and analyze the business processes to ensure that you deploy a successful solution. This planning will help to guide you through the Configuration and Customization that your deployment will require and may also help you to develop the requirements for new modules you may need to develop.

WHO ARE YOUR USERS?

Your deployment of Sahana Eden should be beneficial to its users. To ensure that you achieve this you may need to spend some time consulting your users to find out what their needs are and how Sahana Eden can help them. Identifying key stakeholders and "champions" who will promote the use of Sahana Eden to others can greatly support your deployment.

To effectively deploy Sahana Eden you need to know who the users will be. Some important questions to ask about your deployment are:

- Is it going to be accessible to the public? Is ALL of the data accessible?
- How many users will be accessing the system at once?
- Will it be used only internally within a single organization or across multiple organisations?
- Will there be different types of users? What different data will they be able to access and modify?
- Will new users need to be approved? By who?
- What languages will your users need to access Sahana Eden in?
- Will users need training to effectively use Sahana Eden?
- What support will your users need? How will this support be provided?

Sahana Eden can be easily configured to support a wide range of different answers to these questions.

WHAT SOLUTION IS APPROPRIATE?

It is important to consider how Sahana Eden aligns with existing business processes and what new workflows will need to be performed by your users. If you are introducing new business processes in Sahana Eden, this may require more explanation and training for your users. Sahana Eden can be configured to provide a wide range of solutions from simple tools to complex systems. It is important to ensure that the solution that you deploy is appropriate for your context.

As outlined in the chapter 'What is Sahana Eden', there are a number of different modules available which can be enabled and disabled to provide different types of solutions. The business processes that Sahana Eden is used for will determine which modules are needed. Enabling more modules will increase the complexity and workload of the deployment. Therefore for a successful deployment, it may be recommended to start by supporting a limited number of business processes and ensure they are being used effectively before enabling additional modules.

WHAT'S IN A NAME?

Although you are deploying a solution using Sahana Eden, you are free to give it a name that is appropriate for your context. We do like it if you can leave the "Powered By Sahana Eden" badge within your solution's pages!

CONFIGURATION VS. CUSTOMIZATION

Sahana Eden offers a great deal of flexibility through configuration, however, if you require specific functionality or features which are not already supported you will need to customize the Python code which Sahana Eden is written in. Fortunately Sahana Eden's Framework has been designed to make it easy for you to relabel fields, add new fields, hide existing fields and make fields required. More advanced customization may also be required to add new database tables or even build new Sahana Eden modules.

When Customizing the code, it is recommended to set up a testing and release process to manage this development, especially if you have already deployed a live instance.

WHERE WILL IT BE INSTALLED?

Sahana Eden can be installed on a variety of different infrastructures depending on the needs and resources you have for your deployment.

Local Box/Server

To support multi-user access, it is usual to install Sahana Eden on a server accessible through a Network. This could be using physical hardware in your office, either a server or a computer that can be set up as one. Although specialized server hardware is more expensive it will provide better and more reliable performance.

Hosted Server

One of the more cost effective solutions for installing Sahana Eden is to use a Hosted Service. This allows you to make monthly payments for the use of a Server without having to provide the infrastructure to support a server (location, power back up, air conditioning) and without having to worry about redundancy and maintenance.

Sahana Eden has been effectively installed on Amazon's Cloud service, EC2 (<http://aws.amazon.com/ec2/>).

Flash Drive

Sahana Eden can easily be downloaded and run locally from a flash drive on a Windows computer. It can be configured to be accessed on a local network but for more than 4 users the performance will not be optimal. For instruction on installing a Flash Drive instance, please see <http://eden.sahanafoundation.org/wiki/InstallationGuidelines/FlashDrive>

GOING LIVE!

Once you have installed Sahana Eden there are a number of steps that may be required to make the deployment successful. You may need to train users or, if the deployment is public, promote it to encourage new users. You should also consider operational requirements like user support and backups.

GETTING STARTED

- 7. INSTALLATION
- 8. CONFIGURATION
- 9. IMPORTING DATA
- 10. LOCALIZATION

7. INSTALLATION

Sahana Eden can be installed on any environment which can run Python, including Linux, Windows and OSX. The system supports a number of different databases and has been widely tested on MySQL, PostgreSQL and SQLite. A webserver is optional, but for production installations we have experience of both Apache/mod_wsgi and Cherokee/uwsgi.

For production installations, we would recommend Debian Linux v7 "Wheezy" as this is the environment for which the most support is available. If you don't have a ready server for this, then we'd recommend installing on Amazon's EC2 cloud as this can provide scalable performance with a low setup cost.

Installation scripts and detailed instructions are available on the Wiki:

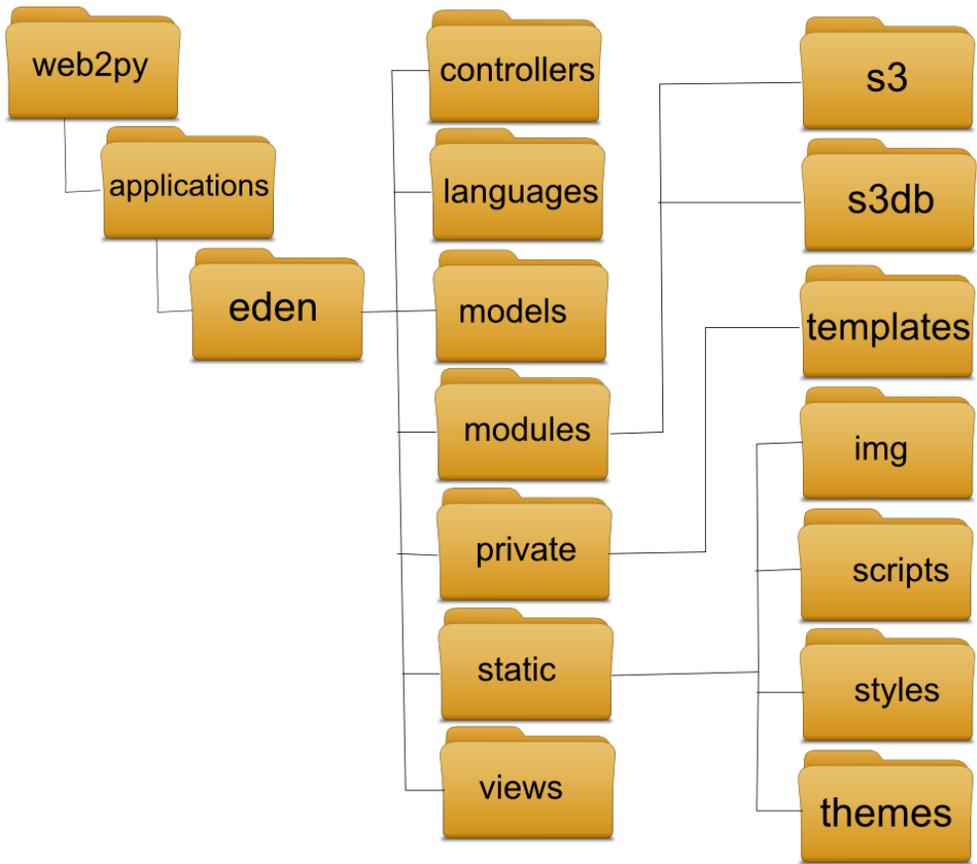
<http://eden.sahanafoundation.org/wiki/InstallationGuidelines>

Note that the first user to register gets administrator privileges for the system.

If you need to customize the code, it is recommended to set up a release process. Ideally, this would include a separate development instance and a User Acceptance Testing (UAT) instance, which can be run on the same server.

DIRECTORY STRUCTURE

After the installation, the typical directory structure of the instance looks like:



Troubleshooting Installation

Initial installation issues are generally due to missed installation steps or a non-standard site configuration. Typical issues are:

Obsolete release packages:

The latest functionality will not be available in packaged releases which are available for download. Currently we advise users to install the latest development version from source instead.

Folder names:

Web2Py does not support hyphens in the application folder name. If you specify a target folder name when cloning, be sure to specify a target folder name without a hyphen in.

Linux permissions:

Web2py needs to be able to write in several Sahana Eden directories: cache, databases, errors, sessions, and uploads. The installation instructions and scripts should set the correct permissions. If you encounter permission errors, refer to the installation instructions and run the commands that set permissions and ownership of the Eden directories.

Apache configuration:

Apache with `mod_wsgi` does not support underscores in hostnames. (Underscores are not a legal hostname character according to the formal W3 URI specification.)

If there are multiple `mod_wsgi` sites enabled, each must have its own `WSGIDaemonProcess` name.

It is possible to run multiple Web2py applications under the same hostname and the same `mod_wsgi` site. However, be careful when setting up `routes.py` or Apache rewrite rules, as these will be shared by the applications.

8. CONFIGURATION

Sahana Eden is a highly configurable system that can be adapted to many different needs and situations. If you've taken the time planning your project and have answered the questions posed in the "Planning a Deployment" section, you are now simply looking for where to enter the answers.

CONFIGURATION THROUGH THE WEB INTERFACE

At the moment only some settings, like SMS, email inbox and Twitter, are editable through the Web UI. Future plans are to add more configuration settings through the Web UI but for now most configuration options require editing text files.

CONFIGURATION THROUGH TEXT FILES

Many configuration options can be changed by editing `models/000_config.py`. This consists of sections of Python code where settings for a particular component of the system can be changed. Most of the changes take effect immediately after saving the file. For a production environment then the system would need to be recompiled.

`models/000_config.py` has to be edited before using Sahana Eden. Once you have edited `models/000_config.py`, change the `FINISHED_EDITING_CONFIG_FILE` variable to `True`.

There are comments placed next to the options which are generally self-explanatory in nature. Users **must not** change the variables (or their names), they just need to change their values to configure the instance.

The following sections of `models/000_config.py` are explained in more detail:

Database Settings

It is recommend that production systems use PostgreSQL or MySQL rather than the default SQLite. For these databases it is more secure to provide the application with a database account with minimal privileges.

This section of the `models/000_config.py` file can be used to configure settings like:

- Database Host: The server where your database is hosted
- Database Name: The name of the database being used
- Username: The username that has been assigned to the user for use with Eden
- Password: Password assigned to the user
- Port: Port at which the database service is available. Set to `None` to use the default setting

Authentication Settings

Administrators can use these settings to implement security policies and to make sure that there is no unauthorized access or data manipulation in the system. These settings are related to creating the first user account of the system and determining how users register and access the system.

Base Settings

Users can configure the system name, the public URL of the system and data pre-population in this section of `models/000_config.py`.

One of the most important system settings would be the selection of the template as this can completely alter how Sahana operates as well as it's look & feel. A list of available templates is in the folder `private/templates`. Any template setting can be over-ridden within `000_config.py` for further fine-tuning as-required.

One of these settings is database pre-population. Users can determine if the database will be pre-populated with sample data or not.

Changing the database migration setting to `False` in production will lead to a performance gain. Migration tries to alter the SQL database schema to match that expected in the code. This works very well for simple cases, but may result in loss of existing data for complex cases, so should be applied with care to Production servers.

Web2Py supports automatic migration, but having this enabled all the time does lead to reduced performance, so enable migration only when necessary.

Mail Settings

Sahana Eden can be configured to use a email service for messaging. This section can help you to set up things like the outbound email server and sender address. Note: Until the Sender address is specified, the system will be unable to send emails!

Front Page Settings

Sahana Eden has a dynamic front page with a capability to display RSS or Twitter feeds. You can change certain aspects of the landing page of the application in the frontpage settings section of the code.

Settings in this section can be used to change which RSS and Twitter feeds are subscribed to and displayed on the front page of the application.

Module-specific Settings

Some settings for the Request Management, Inventory Management and Human Resource Management modules can be accessed here. These settings would generally be very specific to the needs of a certain deployment.

Enabling/Disabling Modules

Sahana Eden supports a range of modules that can be enabled or disabled to support different deployments. The default template (`private/templates/default/`) has all the main modules enabled as standard (you may notice that some other modules are disabled as standard; these tend to be under development or experimental).

Disabling a module has the effect of removing it from the main menu of the application. All modules can be disabled except core modules: Home (default), Administration (admin), Map (gis), Person Registry (pr) and Organization Registry (org).

There are three ways to disable modules. The most direct way to do this is to comment out the relevant lines of code in the configuration file of the default template:

`private/templates/default/config.py`. To turn a line into a comment, simply make sure it begins with a `#` symbol.

For instance, consider the Shelter Registry (named "cr"). The following code section in `private/templates/default/config.py` applies to the Shelter Registry:

```
("cr",
Storage(
    name_nice = T("Shelters"),
    #description = "Tracks the location, capacity and breakdown of
victims in Shelters",
    restricted = False,
    module_type = 10,
)),
```

To disable this module, just make sure that each line in this section starts with a hash (`#`) symbol:

```
#("cr",
#Storage(
#    name_nice = T("Shelters"),
#    #description = "Tracks the location, capacity and breakdown of
victims in Shelters",
#    restricted = False,
#    module_type = 10,
#    )),
```

The module is now disabled and will no longer show up in the application menu.

The drawback of this approach, however, is that the default template will be updated whenever you update your code, and any changes you have made risk being lost. For a long-term solution, it is recommended that you create a new template for your implementation.

Most implementations of Sahana Eden involve the creation of a template folder specific for that project. This will be placed within `private/templates`, as an alternative to the default. The `settings.base.template = "default"` line within `models/000_config.py` will then be changed to reflect the name of the new template folder. Eden will initially look within this folder for a `config.py` file, and if one is present, it will use the module list defined there rather than the one within the default template. To disable unwanted functionality, create a custom version of `config.py` within your template folder, with unwanted modules commented out as described above.

There is a third option for disabling modules that can be useful in some cases. When testing, for example, or when demonstrating a subset of functionality, for example, it may be useful to disable modules without altering the templates. For this, `models/000_config.py` can be used, and a section of that file is provided for adding in overrides to the template. Add to this section a new line of code for each unwanted module:

```
settings.modules.pop("unwanted-module-name", None)
```

This will remove the module from the list that was created by the template. For example:

```
settings.modules.pop("cr", None)
```

While the above line is present, the Shelter Registry will be disabled, just like in the previous example. Because updates to the code do not touch `models/000_config.py`, this change will also be safe from unwanted modification.

For more information on templates, see the [Customisation](#) section of this book.

Updates to `000_config.py`

So that your configuration settings are not changed when you update the code for your implementation, your local copy of `models/000_config.py` is not updated with the rest of the code. Very occasionally, however, updates to `000_config.py` are necessary. If you do experience problems following an update, it is worth checking your copy of `000_config.py` in the `models` folder with the current version. The current version can be found on your system in `private/templates/000_config.py`.

Further information on configuring Sahana Eden can also be found at <http://eden.sahanafoundation.org/wiki/ConfigurationGuidelines>

9. IMPORTING DATA

Adding data is a common activity and Sahana Eden offers a variety of ways to do both batch imports and manual data entry. This section covers importing data from CSV files and some basic troubleshooting.

IMPORT FROM SPREADSHEETS

If you have existing data available in a spreadsheet format it can be imported into Sahana Eden to populate the database.

Resources which support spreadsheet import, have an "Import" menu item in the module menu:

The screenshot shows two parts of the Sahana Eden interface. On the left is a sidebar menu with three sections: 'Warehouses' (containing 'New', 'List All', 'Search', and 'Import' with a red arrow), 'Inventories' (containing 'Search Inventory Items', 'Search Received Shipments', and 'Import' with a red arrow), and 'Received Shipments' (containing 'New' and 'List All'). On the right is the 'Inventory Management' page, which has a header 'Inventory Management' and a description: 'This module allows Inventory Items. Inventory Items include both consumables and non-consumables.' Below this, there are sections for 'Inventories:' and 'Shipments:'. The 'Inventories:' section contains a list of links: 'Warehouses', 'Offices', 'Shelters', 'Hospitals', and 'Search Inventory items'. The 'Shipments:' section contains a link 'Receive'.

Step-by-step

1. Download a CSV Template

Go to the module (e.g. Inventory management), and find the *Import* menu item for the resource you want to import data to (e.g. Warehouses):

By clicking *Import* you get to the upload page which contains a form to upload a new CSV file, and a list of prior imports (this list may be empty):

The screenshot shows the 'Upload Warehouse Data' form in Sahana Eden. The top navigation bar includes 'Home', 'Staff and Volunteers', 'Requests', 'Assessments', 'Inventory Management', 'Map', 'more', and 'Administration'. The sidebar menu on the left is the same as in the previous screenshot, with a red arrow pointing to the 'Import' link under 'Warehouses'. The main content area is titled 'Upload Warehouse Data' and contains a 'Download Template' link, an 'Import File:' field with a 'Blåddra...' button, an 'Organisation:' field, and an 'Upload Data File' button. A '? HELP' link is also present. At the bottom, it says 'No Records currently available'.

If you click on the *Download Template* link in the upload form, you can download an empty CSV file for this data resource in the required format (this CSV file will just have column headers):

	A	B	C	D	
1	Name	Organisation	Country	State	District
2					

2. Fill in the CSV template

Fill in the CSV template with your data or re-format your existing spreadsheet data to match this template:

	A	B	C	D	E
1	Name	Organisation	Country	State	District
2	Example Warehouse	Example.org	Germany	Baden-Wuerttemberg	Karlsruhe
3					

Note that you may change the order of the columns, but do not rename or change the column headers!

3. Upload the CSV file

After you filled in the CSV file with your data, go back to the upload page in Sahana Eden, choose your CSV file and click *Upload Data File*.

4. Review the records and confirm the Import

After uploading the CSV file, Sahana Eden will show a list of records to import ("Import Items") from your CSV file, along with any validation errors:

Details of the selected import job

Total records in the Import Job: 1 Records selected: 1

List of import items

Search: Show 10 entries Showing 1 to 1 of 1 entries

Deselect All	Select All	Element	Error
<input type="checkbox"/>	<input checked="" type="checkbox"/>	name: Example Warehouse	None

Submit First Previous 1 Next Last Showing 1 to 1 of 1 entries

Review the records displayed on the list. You can expand the record details by clicking on *Display Details*.

Records can be selected for import or de-selected by clicking into the *Element* column of the respective row. Selected rows turn green and de-selected rows turn gray, while rows with errors are shown in red (those cannot be selected for import).

Once you have selected which rows shall be imported, then click *Submit* to import the selected rows into the database

Troubleshooting

Sahana Eden reporting validation errors:

- Correct any red colored rows (invalid data) according to the error message displayed in the error column by making the corrections in the CSV file.
- From the review page choose the *Import* menu item and re-upload the CSV file with the corrected data.

Data not being imported as expected:

Note: This refers to data which passed Sahana Eden's validation but does not correspond to the user's expectation of correctness (e.g. *latitude* ending up in a *name* column). Here are some common things to check or try to address the errors:

- You may change the order of the columns, but not rename the column headers
- Data must use UTF-8 character encoding
- Export/Save as CSV file (.xls and .xlsx are also supported although require an up to date version of the python xlwt library. Using CSV is the safest method)
- CSV must use comma as value separator and double quotes (i.e. ") as quoting character.
- A common problem is that cells containing whitespace or commas aren't enclosed in quotes
- All data must be in a single worksheet
- Duplicates will be resolved automatically. Where this fails you might need to check spelling, remove any leading or trailing whitespace in your cells, and make sure the CSV is using UTF-8 character encoding

10. LOCALIZATION

By default Sahana Eden displays all information in US English. However, the system is fully internationalized, which means that all text elements of the user interface can be displayed in any language, including right-to-left languages.

The process of "localizing" Sahana Eden (adapting it to specific language and locale) involves translating the text elements of the user interface into whatever language is needed.

Many translations are already available for Sahana Eden, although they may not be complete or not up-to-date. These include:

- Arabic
- Bosnian
- Chinese (Simplified)
- Chinese (Traditional)
- Dari
- English (UK)
- French
- German
- Italian
- Japanese
- Khmer
- Korean
- Nepali
- Pashto
- Portuguese (Brazil)
- Portuguese (Portugal)
- Spanish
- Russian
- Tagalog
- Tetum
- Vietnamese

UPDATING AN EXISTING TRANSLATION

If you need to update an existing translation, either because it is incomplete or to add customized strings specific to your installation, then you need to update a text file in the languages folder (e.g. `languages/de.py` for the German translation). This file contains a Python dictionary to map the original US English strings to their translated counterparts.

There are 2 approaches that you can take to generate an empty language file for translation:

1. If you have just a small number of modules that you wish to translate quickly then you can remove all untranslated strings from an existing language file. Then navigate through these modules - this will add any untranslated strings that the system encounters to the language file (assuming the relevant file permissions allow this).
2. If you wish to translate the entire application as part of a Preparedness project then you can update all the language files in `languages` by doing the following:

```
cd web2py
python web2py.py -S eden -R
applications/eden/static/scripts/tools/languages.py
```

There are 3 approaches you can take to do the translations:

Note: Inform all translators to not translate the variables within strings (e.g. %(name)s), but just move around the surrounding text to ensure that the word order makes sense.

1. If you have a small number of strings to translate then it is possible to do this using the Web2Py Admin Interface (this assumes that you have a local branch on your machine to work on):

<http://127.0.0.1:8000/admin/default/design/eden#languages>

2. If you want to send these strings to be translated by a professional translation company, then they will typically expect the strings in spreadsheet format. You can create a CSV of strings using the Translate Toolkit:

```
web2py2po -i language.py -o language.po
po2csv -i language.po -o language.csv
```

Tip: Excel has a nasty habit of corrupting strings with quotation marks or other special characters, so avoid this if possible & be prepared to clean-up if not.

3. If you want to use a community of translators then you can use Pootle (see below).

ADDING A NEW TRANSLATION

This can be done via the Web2Py admin interface:

<http://127.0.0.1:8000/admin/default/design/eden#languages>

Create a new file using the [ISO 639-1 Code](#) of the Language plus ".py" as the filename. If it is a national variation of a language, eg. New Zealand English, add a suffix to the language code: "en_nz.py".

The same process then applies as for updating an existing language.

USING POOTLE TO MANAGE TRANSLATIONS

[Pootle](#) is a web-based tool to manage translations by a group of translators which includes the ability to have alternate suggestions reviewed before being selected.

There is a Sahana instance at <http://pootle.sahanafoundation.org> which is available for you to manage the translation for your language.

To use Pootle you need to convert the .py version of your translation to/from the PO format, which can be done using web2py2po from the [Translate Toolkit](#).

ADMINISTRATION

11. MAINTENANCE

12. DATA EXPORT

11. MAINTENANCE

When Sahana Eden has been deployed, then you need to ensure that the system Availability is maintained through any upgrades and that the Data Integrity isn't compromised by ensuring regular Backups are taken.

BACKUPS

Backups are generally done by dumping the SQL to the filesystem & then copying to tape from there. Also remember to backup the contents of the `uploads/` folder

```
# Schedule backups for 02:01 daily
echo "1 2 * * * root /usr/local/bin/backup" >> "/etc/crontab"
```

SCRIPTS

There are a number of useful maintenance scripts which are added to `/usr/local/bin` by the installation scripts.

(Examples shown are for Apache/MySQL, variants are available for Cherokee and/or PostgreSQL. Check the Installation Guidelines section of the Wiki for the latest versions of these scripts.)

clean

This script is used to reset an instance to default values, which may include 'prepopulated' data specific to this deployment.

```
#!/bin/sh
/usr/local/bin/maintenance
cd ~/web2py/applications/eden
rm -f databases/*
rm -f errors/*
rm -f sessions/*
rm -f uploads/*
sed -i 's/deployment_settings.base.migrate =
False/deployment_settings.base.migrate = True/g' models/000_config.py
sed -i 's/deployment_settings.base.prepopulate =
0/deployment_settings.base.prepopulate = 1/g' models/000_config.py
rm -rf compiled
mysqladmin -f drop sahana
mysqladmin create sahana
cd ~/web2py
sudo -H -u web2py python web2py.py -S eden -M -R
applications/eden/static/scripts/tools/noop.py
cd ~/web2py/applications/eden
sed -i 's/deployment_settings.base.migrate =
True/deployment_settings.base.migrate = False/g' models/000_config.py
sed -i 's/deployment_settings.base.prepopulate =
1/deployment_settings.base.prepopulate = 0/g' models/000_config.py
/usr/local/bin/maintenance off
/usr/local/bin/compile
```

w2p

This script is used to open a Python shell in the web2py environment. This allows database migration scripts to be developed interactively.

```
#!/bin/sh
cd ~/web2py
python web2py.py -S eden -M
```

compile

This script is used to compile the Python code so that changes are visible to users (until this time, changes to `.py` files aren't seen by users).

It is called automatically from the 'pull' and 'clean' scripts.

```
#!/bin/sh
cd ~web2py
python web2py.py -S eden -R
applications/eden/static/scripts/tools/compile.py
apache2ctl restart
```

maintenance

This script is used to put the site into 'maintenance' mode & restore it to normal operations. It is usually called from the clean & compile scripts.

```
#!/bin/sh
if [ "$1" != "off" ]
then
    a2dissite maintenance
    a2ensite production
    cd ~web2py && sudo -H -u web2py python web2py.py -K eden -Q
>/dev/null 2>&1 &
else
    killall -u web2py python
    a2ensite maintenance
    a2dissite production
fi
apache2ctl restart
```

backup

This does a dump of the SQL database so that it can be backed-up to tape. It is usually called from Cron.

```
#!/bin/sh
NOW=$(date +"%Y-%m-%d")
mysqldump sahana > /root/backup-$NOW.sql
OLD=$(date --date='7 day ago' +"%Y-%m-%d")
rm -f /root/backup-$OLD.sql
```

MAINTENANCE SITE

This is an alternate Webserver configuration which blocks user access to the application so that upgrades can be done safely. Users see a simple holding page which asks them to try again later. This is (de-)activated by the 'maintenance' script, which is usually called from the 'pull' script.

Tip: It is still possible for administrators to access phpMyAdmin for MySQL database administration whilst the application is offline.

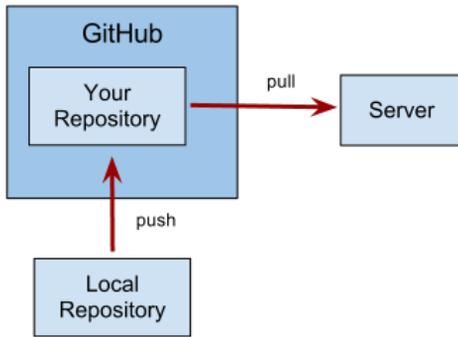
UPGRADES

Simple upgrades can be done by running

```
git pull upstream
```

If there is a database migration required then this will require extra work. It is highly recommended that Production instances use a Development (and ideally a User Acceptance Testing) instance to practice data migration scripts on. This migration should be done using a copy of the Production database.

When making code customizations, it is best to do this in a branch of the code and then pull that code to the server, rather than editing files directly on the server:



TROUBLESHOOTING UPGRADES

Upgrading the version of Sahana Eden or enabling more modules may require updating configuration settings or installing/upgrading library dependencies.

Update configuration settings file

A new version of Sahana Eden may have new settings in `000_config.py` that need to be merged with your current choices. After updating Sahana Eden, compare the copy of `000_config.py` in deployment-templates with the site's copy in models. Merge in added and modified lines.

Missing software packages

A new version of Sahana Eden or a newly-enabled module may require software packages that were not included in the original installation. Optional packages may be needed to make use of new features. The latest list of required and optional packages is on the wiki:

<http://eden.sahanafoundation.org/wiki/InstallationGuidelines/Windows/Developer/Manual>

For optional features that require missing packages, warnings will be printed when the Web2py server is started that list the features and the packages they need. If you don't need this functionality, then these can be safely ignored.

If missing packages are required then attempting to run the application will result in an error ticket with a message saying that this package was not found.

Web2py version

Since Sahana Eden extends Web2Py, and the two are both undergoing rapid development, the revision of Web2Py can be critical. Whilst the latest 'bleeding edge' version of Web2Py is usually stable, some Web2Py revisions have bugs which break a part of Sahana Eden. You can try upgrading to the latest revision of Web2Py or else downgrading to an older version which does not exhibit this bug.

Sometimes a new version of Sahana Eden may use features from a more recent Web2py than the currently installed version. This typically leads to an error ticket with a message indicating that some item was not found. Update to either the latest Web2py, or the latest known-stable Web2py revision, the version number for which can be found in `private/update_check/eden_update_check.py`

It is also sometimes posted in the #sahana-eden IRC channel topic (see the Community chapter for connecting to IRC).

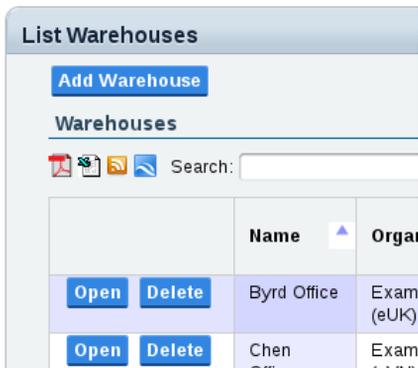
12. DATA EXPORT

There are many reasons why there may be a need to export data. Whether it is for sharing data with another application or organization, or whether it involves physically moving data to another Sahana Eden installation such as from a Test to a Production environment.

Sahana Eden can export data in a variety of data formats, e.g. as spreadsheet (XLS, CSV), PDF, KML and a variety of XML formats.

Some export formats are limited to specific resources. For example, hospital records would support EDXL-HAVE export whereas person records would support PFIF export.

Common export formats can be accessed from the icon buttons above the list view. Export the data by clicking on the respective icon.



Other export formats can be accessed by appending the extension to the URL, e.g.:

For an EDXL-HAVE feed:

`/eden/hms/hospital.have`

For a PFIF feed:

`/eden/pr/person.pfif`

For more details see: Appendix "Web Services".

EXTENDING SAHANA EDEN

13. INSTALLING A DEVELOPER
ENVIRONMENT

14. CUSTOMIZATION

15. BUILDING A NEW MODULE

16. FURTHER READING

13. INSTALLING A DEVELOPER ENVIRONMENT

When making code customizations, it is best to do this in a local copy of the code, test thoroughly, and then pull that code to the server, rather than editing files directly on the server. Keeping the code modifications under revision control greatly eases the process of upgrading the software while keeping your customizations intact.

A development environment can be installed on Windows, Linux or Mac - we have active developers using all three platforms. Detailed instructions can be found on the Sahana Eden wiki:

<http://eden.sahanafoundation.org/wiki/InstallationGuidelines>

To keep the setup simple, we recommended using the default SQLite database and the default Web2py internal web server.

Note: Unless using a set of prepopulate data which includes an Admin user, the first user to register on a new Sahana Eden installation gets administrator rights. You will need this to be able to view any error tickets that are generated, and to examine the database using Web2py's interface.

GITHUB

In order to contribute code you should have your own repository on [GitHub](#), a community collaboration platform based on the Git distributed version control system.

Start by registering here:

<https://github.com/signup/free>

Add your SSH Keys:

<http://help.github.com/win-set-up-git/>

You can then fork the Eden trunk:

https://github.com/flavour/eden/fork_select

You can then clone this branch down locally to work on:

```
cd web2py/applications
git clone git@github.com:mygitusername/eden.git
```

Note: If you already have a local clone of trunk, then you can use this instead of requiring a fresh clone by modifying the source URL in eden/.git/config:

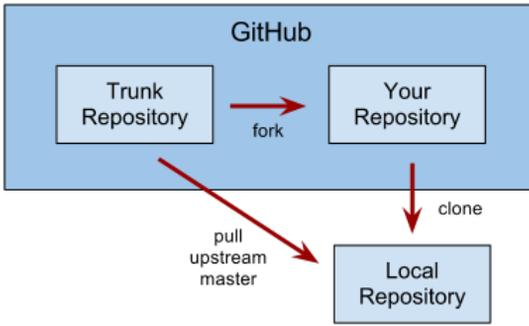
```
[remote "origin"]
  url = git@github.com:mygitusername/eden.git
```

Then set the Trunk branch as the 'upstream' remote:

```
cd eden
git remote add upstream git://github.com/flavour/eden.git
```

This allows you to pull the latest changes to trunk using:

```
git pull upstream master
```



RECOMMENDED DEVELOPMENT TOOLS

For easy inspection and debugging of both CSS and JavaScript when working on UI improvements, use the [Firebug](#) plugin for Firefox.

For serious developers, we recommend [Eclipse](#) as a graphical debugger because of the enhanced visibility gained by setting breakpoints and stepping through the server-side Python code.

VIRTUAL MACHINE

A pre-configured Virtual Machine is available to allow a developer to get operational quickly, which is great for training events to both reduce time on installation and it also gives all participants a consistent environment. The Virtual Machine instructions can be found here:

<http://eden.sahanafoundation.org/wiki/InstallationGuidelinesVirtualMachine>

14. CUSTOMIZATION

Sahana Eden is developed using Python and JavaScript. However, amazing as it may sound, simple customization can be done without any knowledge of either programming language. In fact there is just a single concept which is essential to understand before diving in: whitespace matters.

WHY WHITESPACE MATTERS

Unlike many programming languages, Python code is sensitive to whitespace. To avoid bad results as an impact of this sensitivity, make sure that you indent the code in .py files properly. This is easiest if you configure your text editor to replace Tabs with 4 spaces:



The screenshot is from [Notepad++](#), a recommended editor on Windows, where you can find this by going to Settings -> Preferences... -> Language Menu/Tab Settings -> Tab Settings (group)

DEBUG MODE

As a developer, you should normally run the application in Debug mode.

This means that JavaScript & CSS files are loaded in separate, uncompressed versions and also automatically reloads the core models in `modules/s3db`. Changes to other files in `modules/` still require a restart of Web2Py to become visible.

models/000_config.py

```
settings.base.debug = True
```

WHICH FILE DO I EDIT?

Sahana Eden runs as a Web2Py application. The code is in the folder:

```
web2py/applications/eden
```

Inside that folder are folders for Models (define the data structure), Controllers (provide URLs to enable access to the data) & Views (HTML templates).

Each module within Sahana Eden will normally consist of one of each of these files:

- Model: `modules/s3db/modulename.py`
- Controller: `controllers/modulename.py`
- View: `views/modulename/index.html`

In order to know which file to edit in order to change a particular function, you need to look at the URL. The Web2Py web framework maps URLs as follows:

```
http://host/application/controller/function
```

So, if you want to edit the Home page with the URL:

`http://host/eden/default/index`

This implies that you should look at the file `eden/controllers/default.py` and the `index` function within it which can be found by searching for the function title "`def index():`"

Tip: Sahana Eden makes heavy use of integrated resource controllers so the typical mapping is:

`http://host/eden/module/resource`

The resource refers to a table with the name `module_resource` in the file `modules/s3db/<module>.py`

TEMPLATES

When making customizations, it is better if you can retain compatibility with the upstream trunk code, so that you can 'pull' updates with less chance of conflicts. For this reason it is better if all your customizations are contained within your own folder inside `private/templates/` eg. `private/templates/template_name`.

If, on the other hand, you are fixing a bug or developing functionality useful to other Sahana users, then that should be done in the core code & your changes submitted upstream via a 'pull request'. See the Git chapter.

To set the template used by Sahana Eden, edit this line in the file `models/000_config.py`

```
settings.base.template = "template_name"
```

Once you have created a folder for your template, you can place some of these files in:

- Module loader:
`private/templates/template_name/__init__.py`
- Main Configuration:
`private/templates/template_name/config.py`
- Prepopulate Configuration:
`private/templates/template_name/tasks.cfg`
- CSS
Configuration: `private/templates/template_name/css.cfg`
- Custom Controllers:
`private/templates/template_name/controllers.py`
- Custom Menus:
`private/templates/template_name/menus.py`
- Custom Layouts:
`private/templates/template_name/layouts.py`
- Custom Views:
`private/templates/template_name/views/`

Not all of these files are needed in every template. We will now go through what each of these are used for:

`__init__.py` is needed if there are custom controllers, menus or layouts. It is normally empty.

`config.py` is needed if there are any custom `deployment_settings` which are common to all instances of this deployment (e.g. production, Test, Demo &/or Training) or customizations to standard controllers. Most templates will contain one of these unless the folder is just a collection of pre-populate files or just a theme.

`tasks.cfg` is a collection of pre-populate CSV files. These are used to configure the base system with lookup lists (e.g. Organization Types) and Map configuration. They can also contain Demo data. The CSV files can be in this folder, another folder or even downloaded from a remote server.

`css.cfg` is a collection of CSS files which are loaded in every page for this Theme (a template folder is often linked to a theme of the same name, however this is not mandatory. Many templates can share a common theme). In non-debug mode the files are compressed together and downloaded as one file, whereas in debug mode they are downloaded individually. The CSS files can be from libraries, such as jQueryUI, from other templates, such as the default template, or custom ones for this theme.

`controllers.py` allows the creation of fully custom controllers. The most common usage is to provide a fully custom homepage (default/index). Additional pages may be created as `default/custom/mycustompage`.

`menus.py` allows custom menus.

`layouts.py` allows custom design for menus.

`views/` folder allows custom views. For some core system pages which are commonly changed (such as the homepage, About, Contact, Help), it is sufficient to simply drop an HTML file into this folder. A Theme will normally have a customized `layout.html` here.

HOME PAGE AND OTHER SIMPLE VIEWS

One of the common changes that needs to be made is modifying the main home page, individual module home pages and the Contact/About pages.

Some of these customizations can be done in pure HTML by editing the View files. For example, you can edit the contact page at URL `/eden/default/contact` by copying the file:
`views/default/contact.html` to
`private/templates/template_name/views/contact.html`

This is a normal HTML page other than the part(s) inside double curly braces `{{ . . . }}` which indicate Python code.

```
{{extend "layout.html"}}
```

This part should not be edited as it loads the generic page layout from `views/layout.html` or, if using a custom theme, from `private/templates/template_name/views/layout.html`.

Most pages, such as the main home page, include the results of Python code executed in the controller file. This is executed before the view template is parsed and dynamically generates content which is inserted in the curly brackets inside the HTML. The controller file for the main home page is `controllers/default.py`

The simple way to start customizing this page is to ignore some or all of this dynamic content and replace it with simple HTML content in the view template:

```
private/templates/template_name/views/index.html
```

Tip: Sahana Eden doesn't make use of many custom views in the core modules – normally the generic view templates within the `views/` folder are used, such as `_create.html` and `_list.html`. Modifications are made by configuration in the Model and Controller files. However it is possible to customise these views by adding these into `private/templates/template_name/views/`

EDIT A FIELD LABEL

This can be done by editing the 'label' attribute in the field properties within the model.

Example:

Change the text 'Year' to 'Year Founded' in the form at URL: `/eden/org/organisation/create`



Year: ? HELP

You can lookup the fieldname in the main model `modules/s3db/org.py`:

```
tablename = "org_organisation"
table = define_table(tablename,
    ...
    Field("year", label = T("Year")),
```

Since we are putting our customizations into our Template, we need to customize the controller in

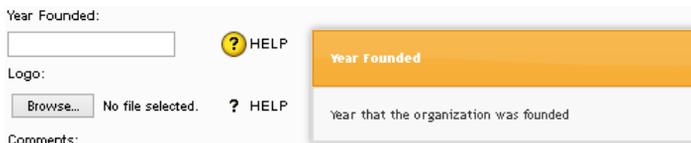
`private/templates/template_name/config.py`:

```
def customise_org_organisation_controller(**attr):
    table = current.s3db.org_organisation
    table.year.label = T("Year Founded")
    return attr
settings.customise_org_organisation_controller =
customise_org_organisation_controller
```

Tip: The strings are internationalized by wrapping them inside `T()`. If you change the labels then you need to update any translation files that you are using, even if the translation remains the same. (Note the UK English spelling of variable names. Labels and other text that appears in the user interface are standardized to US English as they are used as identifiers for translated text.)

For consistency, you should also edit the heading of the help tooltip, and maybe the body, if the meaning is being changed. This is found in the 'comment' attribute:

```
table.year.comment = DIV(_class="tooltip",
    _title="%s|%s" % (T("Year Founded"),
        T("Year that the organization was
founded."))),,
```



Year Founded: ? HELP

Logo: No file selected. ? HELP

Comments:

Year Founded

Year that the organization was founded

HIDE A FIELD

It is common to want to hide a field to simplify a form to allow efficient collection of the data that you need. Hiding a field is safer than removing it completely from the database as it eases upgrades by eliminating the need for database migration.

Example:

Hide the 'Code' field from the form at URL: /eden/org/office/create

Add Office

*** Required Fields**

Name: *

Code:

Organization: *

Since we are putting our customizations into our Template, we need to customize the resource in

private/templates/template_name/config.py:

```
def customise_org_office_resource(r, tablename):
    table = current.s3db.org_office
    table.code.readable = False
    table.code.writable = False
```

settings.customise_org_office_resource = customise_org_office_resource

Tip: Python variables are case-sensitive, as are the boolean values 'True' and 'False'.

Name: *

Organization: *

[Add Organization](#)
 ? HELP

ADD A NEW FIELD

If you need to collect extra data for an existing resource, then it isn't currently possible to do this within a template. However it is simple and relatively safe to add an extra field to the model for that table.

Example:

Add the ability to store a Facebook page for Organizations.

Look at **modules/s3db/org.py** and add the new field to:

Year:
 ? HELP

Twitter:
 ? HELP

```
tablename = "org_organisation"
table = define_table(tablename,
    ...
    Field("facebook", label=T("Facebook Page")),
```

Facebook Page:

Twitter:

Tip: Be careful to add trailing commas to each line in the table definition as this is a common source of errors.

EDIT THE MENUS

If you wish to edit the left (second-level) menus (e.g. relabeling, reordering or hiding entries) then create the file **private/templates/*template_name*/menus.py**. We can copy from `modules/s3menus.py` and override the menu for just the modules that we need.

Example:

In the Organization Registry Module, re-name 'Offices' as 'Venues' and hide the 'Search' & 'Map' options:

Organizations
Add Organization
List All
Search
Import
Offices
New
List All
Map
Search
Import

```
from s3layouts import *
try:
    from .layouts import *
except ImportError:
    pass
import s3menus as default

class S3OptionsMenu(default.S3OptionsMenu):
    """ Custom Controller Menus """
    def org(self):
        """ ORG / Organization Registry """

        return M(c="org")(
            M("Organizations", f="organisation")(
                M("New", m="create"),
                M("List All"),
                M("Search", m="search"),
                M("Import", m="import")
            ),
            M("Venues", f="office")(
                M("New", m="create"),
                M("List All"),
                #M("Map", m="map"),
                #M("Search", m="search"),
                M("Import", m="import")
            )
        )
```

You can see this change in the Organization Registry Module at URL: </eden/org>

Organizations
New
List All
Search
Import

Venues
New
List All
Import

Tip: The # character comments out the text after it on that line so that Python will ignore it.

15. BUILDING A NEW MODULE

This chapter walks you through the process of adding a new module. This may sound like a big project, but the Sahana Eden Framework supports Rapid Application Development (RAD), which allows simple functionality to be added easily.

EXAMPLE: TRAINING COURSES

Imagine that we want to add the capability to manage training courses from within our Sahana Eden instance. Instead of installing a separate package (such as Moodle) for this, we have decided to integrate this into our Sahana Eden instance so that:

- The Human Resource Management (HRM) module can use training records of personnel
- We don't need to define data (e.g. users, locations, training courses) in multiple systems
- We can use Sahana Eden's messaging, scheduling and mapping capabilities for the training courses

NOTE: This Training Course module is only an example as Sahana Eden includes functionality for managing trainings within the HRM module.

IDENTIFY THE RESOURCES

The first step in constructing a new module is to identify the *Resources* involved.

Here, the primary *Resource* for the Training module will be a 'course'. Each course includes:

- the date, time and site where the course will be held
- the facilitator
- participants
- course materials

DEFINE THE BASIC DATA MODEL

We'll start by defining a database table with a few simple fields:

Tip: By convention, database tables in Sahana Eden are named as 'module_resource'. Here, the module is 'training' and the resource is 'course'.

Create a new file in the `models/` folder called `training.py` and add the following code (you may leave out the comments after the `#` characters).

`models/training.py`

```
tablename = "training_course"
db.define_table(tablename,
    # A 'name' field
    Field("name"),
    # The start time
    Field("start"),
    # The facilitator
    Field("facilitator"),
    # This adds all the metadata to store
    # information on who created/updated
    # the record & when
    *s3_meta_fields()
)
```

Add New Record

Name:

Start:

Facilitator:

Tip: If your configuration in `models/000_config.py` has `settings.base.migrate = True` then Web2py will automatically 'migrate' your database: creating and modifying tables according to your model changes.

ADD A CONTROLLER

Next we add a *Controller*, which provides access to this resource.

Create another new file, this time in the `controllers/` folder:

`controllers/training.py`

```
def course():
    return s3_rest_controller()
```

The `s3_rest_controller` function provides all the Sahana Eden framework support needed to access the resource, including automatic loading of the respective model definitions. You should now have a working module. You can see the *CRUD* (Create, Read, Update, Delete) user interface here:

<http://127.0.0.1:8000/eden/training/course>

Tip: You will need to register for a login to be able to create new courses. The 1st user to register gets the administrator role.

All resources can be accessed in other formats, such as XLS, XML or JSON, just by appending the representation name to the URL, e.g

- <http://127.0.0.1:8000/eden/training/course.xls>
- <http://127.0.0.1:8000/eden/training/course.xml>
- <http://127.0.0.1:8000/eden/training/course.json>

REPORTS

Pivot table reports with bar charts and pie charts can be generated for all resources, by appending the method name to the URL, e.g.

<http://127.0.0.1:8000/eden/training/course/report>

FIELD TYPES

By default fields are created with type `string`, however we may wish to use other data types. All fields have both client-side widgets & server-side validation automatically added based on their data type.

`models/training.py`

```
tablename = "training_course"
db.define_table(tablename,
                Field("name"),
                # A date type field (includes widget & validation)
```

```

s3base.s3_date(),
Field("facilitator"),
# This is a file attachment that contains
# a welcome pack that will be sent to each
participant:
Field("welcome_pack", "upload"),
*s3_meta_fields()
)

```



Add New Record

Name:

Date:



FIELD LABELS

Field labels are automatically generated from the field names, however we are able to customize these by adding a 'label' attribute.

models/training.py

```

tablename = "training_course"
db.define_table(tablename,
    Field("name"),
    s3base.s3_date(label="Start Date"),
    Field("facilitator"),
    Field("welcome_pack", "upload"),
    *s3_meta_fields()
)

```

Add New Record

Name:

Start Date:

Facilitator:

Welcome Pack:
 No file selected.

INTERNATIONALIZE FIELD LABELS

By wrapping a string in the `T(...)` function they will be added to language files which can be translated, allowing the system to be localized into other languages. To localize the field labels we need to provide a 'label' attribute with the string wrapped in `T(...)`, even if the English version of the label is the same as the automatically generated one.

models/training.py

```
tablename = "training_course"
db.define_table(tablename,
    Field("name",
        label=T("Name")),
    s3base.s3_date(label=T("Start Date")),
    Field("facilitator",
        label=T("Facilitator")),
    Field("welcome_pack", "upload",
        label=T("Welcome Pack")),
    *s3_meta_fields()
)
```

ADD LINKS TO OTHER RESOURCES

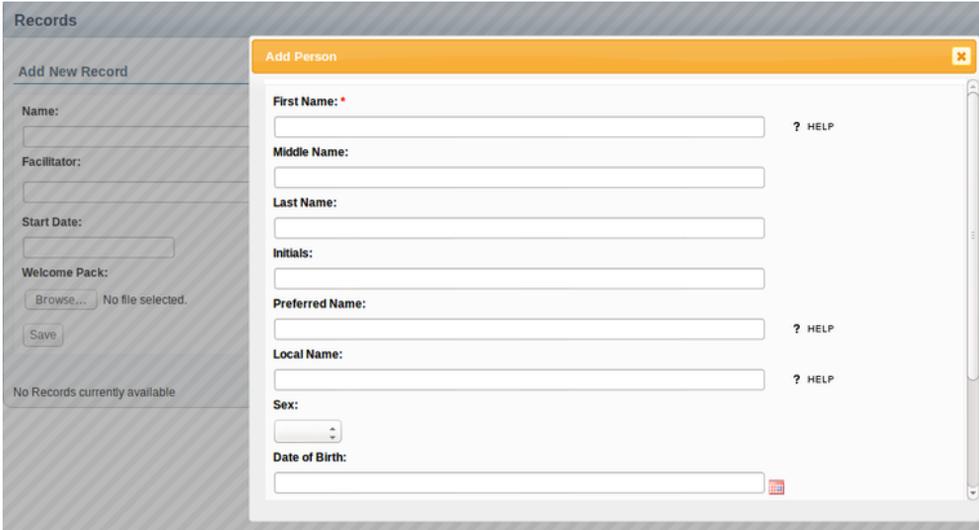
The course resource needs connections to existing resources: people & sites./p>

These are represented in SQL databases as 'Foreign Keys' which are usually defined in Sahana Eden by using 'Reusable Fields' to make the process simple.

We can link to a person by adding a `person_id` to the table definition in the model:

models/training.py

```
tablename = "training_course"
db.define_table(tablename,
    Field("name",
        label=T("Name")),
    # Link to the Person Resource
    s3db.pr_person_id(label=T("Facilitator")),
    s3base.s3_date(label=T("Start Date")),
    Field("welcome_pack", "upload",
        label=T("Welcome Pack")),
    *s3_meta_fields()
)
```

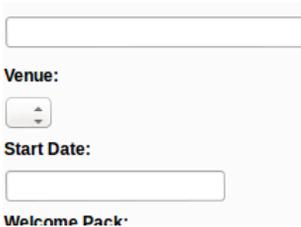


Note how we over-ride the default label to be more appropriate to this context.

The site link is a little more complex as this is a [Super Entity](#):

models/training.py

```
tablename = "training_course"
db.define_table(tablename,
    Field("name",
        label=T("Name")),
    s3db.pr_person_id(label=T("Facilitator")),
    # Link to the Site resource
    s3db.super_link("site_id", "org_site",
        label = T("Venue"),
        # superlink fields are normally
        readable = True,
        writable = True,
        # we want users to see the site name
        # rather than just the ID
        represent = s3db.org_site_represent,
    ),
    s3base.s3_date(label=T("Start Date")),
    Field("welcome_pack", "upload",
        label=T("Welcome Pack")),
    *s3_meta_fields()
)
```



Note that we make use of a 'represent' function so that users see site names in the drop-down and not just integer IDs

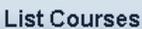
Tip: You will need to create a site (e.g. through /eden/org/office/create) to be able to see this

CRUD STRINGS

You can replace the default strings within the *CRUD* user interface with custom strings for your resource.

models/training.py

```
s3.crud_strings[tablename] = Storage(
    label_create = T("Create Course"),
    title_display = T("Course Details"),
    title_list = T("List Courses"),
    title_update = T("Edit Course"),
    title_upload = T("Import Courses"),
    subtitle_list = T("Courses"),
    label_list_button = T("List Courses"),
    label_delete_button = T("Delete Course"),
    msg_record_created = T("Course added"),
    msg_record_modified = T("Course updated"),
    msg_record_deleted = T("Course deleted"),
    msg_list_empty = T("No Courses currently registered"))
```



MODULE INDEX PAGE

The "course" controller we created earlier, controls just a single page within our new module. More pages like it can be created by adding new controller functions. One such special controller function is the index function which handles the index page of the module. Any links we create to the module will be directed at this page.

controllers/training.py

```
def index():
    return dict()
```

This is a minimal controller function which passes control to a *View* template.

The default view template which is called has the same name as the function and is located in the folder named after the controller, so create this new file, and add the following code:

views/training/index.html

```
{{extend "layout.html"}}
<h2>Welcome to the Training Module</h2>
<ul>
  <li>
    <a href='{{=URL(f="course")}}'>
      List Training Courses
    </a>
  </li>
</ul>
```

Note that this is normal HTML code, apart from sections which are enclosed within `{{ . . }}`, which are normal python code, other than the special terms 'extend' & 'include' which allow HTML template fragments to be reused within each other.

Tip: The `URL()` function is an HTML helper which is used to generate a URL to access the course resource without hardcoding the application name.

MENUS

There are two levels of menu within the system:

- The top-level 'Modules Menu' is visible in all modules.
- Underneath that, each module has its own menu for for module-specific navigation.

To change the top-level Modules menu, edit the following file and add a new entry for the Training Course module inside the 'modules' data structure:

models/000_config.py (at end of file)

```
settings.modules["training"] = Storage(  
    name_nice=T("Training"),  
    module_type=2)
```

Note: `models/000_config.py` is not in the version control system, and so is not changed when the software is updated. It is this instance's working copy, with local settings, of the configuration file. The template for this file is: `private/templates/000_config.py`

Adding the module to `settings.modules` through `000_config.py` is just a quick way of making a small modification to the template currently in use. Most implementations of Sahana Eden will define their own template, in which the modules to be used will be listed. See the customisation and configuration chapters of this book for further details. When a new module is ready to be added to the main Eden code base, the above code should be removed from `000_config.py` and inserted into the sequence of similar operations in the default template (`private/templates/default/config.py`). By convention, the default template has all modules enabled (except experimental ones and those under development). Implementations then disable unwanted modules when they define their own templates.

To add a menu for use within the new training module, edit the following file and add a new function within the `S3OptionsMenu` class, which provides access to the 'course' controller. Note that the `S3OptionsMenu` class already exists - you just need to add a new function within it.

modules/s3menus.py

```
class S3OptionsMenu(object):  
  
    def training(self):  
        return M(c="training")(  
            M("Courses", f="course")(  
                M("Create", m="create"),  
                M("Import", m="import"),  
                M("Report", m="report"),  
            )  
        )
```

You also need to add the training module to your list of active modules:

private/templates/default/config.py

```
settings.modules = OrderedDict([  
    ...  
    ("training", Storage(  
        name_nice = T("Training"),  
        module_type = 10,  
    )),  
    ...  
])
```

COMPONENTS

Note: This section is significantly more advanced than the previous example, so should only be tackled if you're feeling comfortable with the material so far.

We'd like to be able to record information relating to each participant in the course, such as whether they actually attended and what grade they attained.

To do this, we need to build a 'link' table between the participants and the course.

The natural way to do this within Sahana Eden is to make the link table a 'component' of the course. The course is the 'primary resource', and participants are a 'component' of the course.

Tip: See the Resource Model chapter in the appendices for an explanation of the resource and components concept.

Model

Edit the following file and add this after the existing code:

models/training.py

```

represent = S3Represent(lookup = tablename)
course_id = S3ReusableField("course_id", "reference %s" % tablename,
                           label = T("Course"),
                           ondelete = "RESTRICT",
                           represent = represent,
                           requires = IS_ONE_OF(db,
                                                "training_course.id",
                                                represent),
                           )

```

This defines a 'reusable field' which can be added to other table definitions to provide a foreign key reference to the course table:

Note that this uses a 'represent' function to allow a record in the course table to be represented by its name (The S3Represent class allows bulk lookups for scalability).

It also adds a 'requires' validator function. This provides both server-side validation and a client-side widget (in this case a dropdown of records in the course table).

Define a set of options for the course grade attained by each participant:

models/training.py

```

course_grade_opts = {
    1: T("No Show"),
    2: T("Failed"),
    3: T("Passed")
}

```

These options associate a number, which is what will be stored in the database, with a label meaningful to the users.

Define the participant component resource, making use of the course reusable field and grade options we just defined. (Note that we make use of another validator -- the client-side widget is again a dropdown, although here the options come from the grade options dictionary rather than a database table.)

models/training.py

```
tablename = "training_participant"
db.define_table(tablename,
                course_id(),
                s3db.pr_person_id(label=T("Participant")),
                Field("grade", "integer",
                    label=T("Grade"),
                    requires=IS_IN_SET(course_grade_opts),
                ),
                *s3_meta_fields()
            )
```

Note, that unlike before, no "represent" parameter is required to specify the mapping from grade labels to numbers. This is because the IS_IN_SET requirement (a part of Web2py) automatically does this for you if you give it a dictionary.

s3.meta_fields() is a helper that provides a set of fields commonly needed in each table, such as what user created the record and when it was created.

Tip: Functions, classes, and values that start with 's3' or 'S3' are part of the Sahana Eden framework - have a look for more of these.

Tell the framework that a participant is a component of a course:

models/training.py

```
s3db.add_components("training_participant",
                   training_course = "course_id")
```

Controller

There is no need to create a separate REST controller to manage the component, since it will always be accessed via the existing course controller, however we must then extend the controller with 2 new elements to allow the Sahana Eden framework to display the component: 'tabs' and an 'rheader'.

Tabs are how the framework provides access to the different components in a web page for the primary resource.

The 'resource header' is a section of HTML that provides a summary of the primary resource record, in this case the course. This is displayed above the tabs so that when each component record is being viewed, its parent record is also visible at the same time.

Edit the following file, adding this content above the course controller:

controllers/training.py

```
def course_rheader(r, tabs=[]):
    if r.representation != "html":
        # RHeader is a UI facility & so skip for other formats
        return None
    if r.record is None:
        # List or Create form: rheader makes no sense here
        return None

    tabs = [(T("Basic Details"), None),
            (T("Participants"), "participant")]
    rheader_tabs = s3_rheader_tabs(r, tabs)

    course = r.record

    rheader = DIV(TABLE(
        TR(
            TH("%s: " % T("Name")),
```

```

        course.name,
        TH("%s: " % T("Start Date")),
        course.start,
    ),
    TR(
        TH("%s: " % T("Facilitator")),
        s3db.pr_person_represent(course.person_id),
    )
), rheader_tabs)

return rheader

```

Modify the previous course controller with this code:

controllers/training.py

```

def course():
    return s3_rest_controller(rheader=course_rheader)

```

Tip: `rheader` is simply a variable passed through the REST controller unaltered & then serialized as `rheader.xml()` in the views.

FURTHER OPTIONS

The following are some possible directions for this module, although they are currently beyond the scope of this tutorial. Please feel free to experiment with implementing them!

Instance-Specific Components

If a course is offered multiple times, most of the course details should be the same between instances, so courses could be refactored into a generic course (e.g. in a course catalog) with static information (e.g. name, and maybe the course materials), and course instances representing each offering of the course (date / time, site, and participants would be associated with course instances). The generic course would be a primary resource, and course instances would be its components. With this, we would have two levels of resource and component: a generic course has instances, and each instance has participants.

Authorization

If we need to define a 'role' to manage the training courses, so that only people who have that role can modify courses, or a facilitator role that is allowed to set grades. That can be done by editing the file: `private/templates/default/auth_roles.csv`

Messaging

We could add a button to a course's web page, to mail the course materials to the participants. See how the `dispatch()` custom method does this within the Incident Reporting System (IRS) by calling `msg.compose()`.

Scheduler

We could set a reminder to mail the facilitator two weeks before the course start so they can make sure the course materials are up to date and mail them out to the participants.

The Scheduler API is defined in `modules/s3/s3task.py` and tasks are defined in `models/tasks.py`.

Mapping

We could display a map of all upcoming training courses. This is done by calling `gis.show_map()` from `modules/s3/s3gis.py`. There are further instructions on the wiki - search for the Developer Guidelines on GIS.

Conditional Model Loading

Not all of the Eden data models may be needed for the processing of a particular request. For optimum performance, the S3 framework provides a mechanism to only load those models which are needed.

Data models are implemented as Python modules in `modules/s3db`, which contain the database table definitions. Apart from the table definitions, modules can also define global functions and variables.

In `modules/s3db/skeleton.py` you can find a comprehensively documented example for how to implement such a module.

Tip: When doing this we need to ensure that the model is loaded when-required, such as in our `represent` function, by accessing it as `s3db.training_course`, or in order to detect cases where it has been disabled:

```
table = s3db.table("training_course")
if table is not None:
    # Code that depends on training_course
    ...
else:
    # Alternative code
    ...
# Independent code
...
```

Database Abstraction Layer

When getting deeper into the code, you'll notice that we use Web2Py's Database Abstraction Layer (DAL) to do a SQL query. The variable `db` is an instance of the DAL class, which represents a database. Queries are written in a syntax that is much like a Python expression, but not quite. Look at the Web2Py book (<http://web2py.com/book>) for more on the DAL.

16. FURTHER READING

This manual is not designed to teach you Python, Web2Py, or Javascript as we encourage you to dive straight into Sahana Eden code to learn as you go. However this chapters provide additional reference resources for those who wish to get additional skills for more significant code changes.

PYTHON

Python is a high-level scripting language suitable for rapid application development. It has a wealth of powerful libraries available. If you're interested in learning more about Python outside of our code, refer to these excellent Python resources:

- Dive Into Python (<http://diveintopython.org>)
- How to Think like a Computer Scientist (<http://openbookproject.net/thinkcs/python/english3e/>)

WEB2PY

Web2Py is a simple yet powerful framework to allow people to rapidly develop secure real world applications. We like the [Official Web2Py book](http://web2py.com/book) (<http://web2py.com/book>) as a resource.

Here are a few tips about Web2Py:

1. All Models are executed during every request in alphabetical order within web2py environment
2. The Controller is executed
3. The View template is parsed
4. (HTML) page returned to client

Tip 1: Python Modules are not reloaded for every request, so if changes are made to these files then you would need to restart Web2Py to see the differences.

Tip 2: Because all the models are executed during every request, the code added there should be optimized - search for 'conditional model loading' in the code for guidance on how to do this.

JAVASCRIPT

Sahana Eden uses two JavaScript libraries: jQuery and ExtJS.

jQuery offers a simple way of adding unobtrusive client-side interactivity to widgets. It has a wealth of plugins available (some of which we copy to static/scripts/S3, the Amazon Simple Storage Service) and excellent documentation at <http://docs.jquery.com>

ExtJS provides some very advanced UI components that are primarily used for the Map. It has a wealth of plugins available (some of which we copy to static/scripts/S3, the Amazon Simple Storage Service) and excellent documentation at <http://docs.sencha.com/ext-js/3-4/>.

SAHANA EDEN BUILD AND DEBUG TIPS

For end-user performance gains, Sahana Eden minimizes and compresses the CSS and JavaScript. While this approach works well for optimized end-user performance, to debug the CSS and JavaScript you should enable debug mode in `models/000_config.py` in your Sahana Eden server with the following setting:

```
settings.base.debug = True
```

Once any changes to the CSS and JavaScript are working, then you can minimize and compress the CSS and JavaScript using:

```
static/scripts/tools/build.sahana.py
```

Although this uses a web service, you get better results by downloading a local version of the [Closure Compiler](#) (a tool for making JavaScript download and run faster) to `static/scripts/tools`.

Tip: It is also possible to quickly view a single page in debug mode by adding the `?debug=1` variable to the end of a URL.

MEETING THE SAHANA COMMUNITY

17. SAHANA SOFTWARE FOUNDATION

18. GETTING HELP

19. GETTING INVOLVED

20. CONTRIBUTING CODE

21. WHERE TO GO NEXT

17. SAHANA SOFTWARE FOUNDATION

Sahana Eden is a project of the Sahana Software Foundation (SSF), which is dedicated to the mission of saving lives by providing information management solutions that enable organizations and communities to better prepare for and respond to disasters. SSF provides different software tools to supplement the the process of disaster mitigation and response. SSF provides governance and direction to the Eden project among others.

For more, visit: <http://www.sahanafoundation.org>

MEMBERSHIP

The SSF is supported by members who have contributed and continue to support the to SSF's projects. These members include:

- Disaster Management practitioners who have used Sahana software tools.
- Commercial companies that develop and support Sahana software.
- Academics who use Sahana software in teaching and research.
- Volunteer software developers, translators, testers and others.

If you are interesting in becoming a member please contact community@sahanafoundation.org

EVENTS

SahanaCamps

SSF runs SahanaCamps to bring disaster management and relief professionals together with software deployers and developers to explore the use of Sahana software through simulations and participatory discussions. SahanaCamps also provide technical training for software developers wanting to deploy or contribute to the Sahana Software Foundation. SahanaCamps have been held in India, Taiwan, Vietnam, Portugal and the USA.

Google Summer of Code

Google Summer of Code (GSoC) allows college and university students to learn professional software development skills and develop code that will be used in live products. SSF has participated in the Google Summer of Code internship program every year since 2006. Experienced SSF software developers mentor the students to provide a rich learning experience.

For more information on Google Summer of Code, go to: <http://code.google.com/soc/>

Google Code-in

Google Code-in (GCI) is a program to give high school students the opportunity to participate in open source projects. SSF participated in the 2010 and 2011 Google Code-in, in which students from all over the world completed small tasks including testing, code reviews, bug fixing, translation, and writing documentation.

For more information on GCI, go to:

<http://code.google.com/opensource/gci>

18. GETTING HELP

There are several options for getting the support you need to use Sahana Eden, ranging from basic assistance from the voluntary community through professional support.

MAILING LIST

You can contact the mailing list to ask any questions you may have about Sahana Eden. This is the best way to engage with the entire Sahana Eden community, get answers to any questions you may have and share the work you are doing with Sahana Eden.

Details on the list are found

here: <http://eden.sahanafoundation.org/wiki/MailingList>

WEB CHAT

Real-time communication can help facilitate allow more rapid discussion of ideas. You can join the Sahana Eden Internet Relay Chat (IRC) channel to chat directly with members of the community.

If you have an IRC client you can join #sahana-eden channel on irc.freenode.net. Otherwise you can go to <http://webchat.freenode.net/?channels=sahana-eden&uio=d4>

Please be aware that people on the Web Chat may be busy or otherwise occupied and it may be useful to also send an email with any queries you have to the mailing list.

For more information,

see: <http://eden.sahanafoundation.org/wiki/Chat>

MONTHLY COMMUNITY CALL

The Sahana community also gets together on monthly voice calls to discuss ongoing work, new projects, upcoming events and other topics. This call is open to everyone.

For more information,

see: <http://wiki.sahanafoundation.org/doku.php/community:call> or contact: community@sahanafoundation.org.

WIKI

The Sahana Eden wiki contains detailed technical documentation, instructions for deploying the software, guides to contributing to the project and blueprints for future work. It is maintained by community member. You are welcome to register for an account to help improve and extend the wiki

The wiki is at: <http://eden.sahanafoundation.org>

REPORTING BUGS

You can help improve Sahana Eden by reporting any bugs you find in the software. To report a bug go to: <http://eden.sahanafoundation.org/newticket>

This ensures that these bugs can be fixed in future releases of the Sahana Eden.

Some things to remember while filing a bug report:

1. Please be explicit about what you were trying to do when you encountered the bug and what module you are using. A URL (website address) is very useful.
2. Be sure to write in detail the steps to reproduce the bugs. It would also be helpful if you could write out what your trying to achieve when you encountered the bug. For instance, if you were filling out a form and the application crashed as soon as you hit submit, it would helpful to put the data you were trying to put in and the module you were in the bug report.
3. You could also check the wiki for some information, perhaps the bug you encountered is unresolved issue which the developers are aware of. It also be possible that the feature might not be complete yet.
4. If you are a developer, make sure you have the latest version of the code. It is possible that the issue you are facing has been resolved.
5. Describe the environment you are running the system on in the bug report. If you can, please include the operating system, the browser, the version of Python you have, the Web2Py version, Sahana Eden revision and Python libraries installed. Listing out everything may be a little tedious, but the more specific you can be, the easier it will be to fix the bug.

Refer to <http://eden.sahanafoundation.org/wiki/BugReportingGuidelines> before reporting a bug.

PROFESSIONAL SUPPORT

Unpaid volunteers from the Sahana Eden community can provide a basic level of support for deploying and using Sahana Eden. If you require more comprehensive support and guaranteed response, professional companies are available. Both AidIQ (www.aidiq.com) and Respere (www.respere.com) provide deployment, customization, hosting, training and support for Sahana Eden solutions.

19. GETTING INVOLVED

Getting involved in Sahana Eden is a great way to support work in Disaster Management. You can contribute in a variety of ways.

SOFTWARE DEVELOPMENT

If you are a software developer who is interested in contributing to Sahana Eden there are a wide range of projects which you can get involved in.

See:

<http://eden.sahanafoundation.org/wiki/Projects>

DOCUMENTATION

The wiki and this book form the backbone of the documentation available for Sahana Eden. There is always scope for improving and extending the documentation. Contributing to the documentation it is a good way for developers to to better understand Sahana Eden. If there is something in the documentation which is not clear to you or is missing, you can improve it.

You can also produce screencasts which are a quick and easy way for new users and developers to understand Sahana Eden.

TESTING

By testing Sahana Eden you can help to make it more reliable. This can be done by writing test cases, performing manual tests and automating tests with Selenium.

BLUEPRINTS

Blueprints allow users to share their requirements and developers to document their ideas for Sahana Eden. Its a good way for users to engage with the Sahana Eden community communicate their needs.

SYSTEM ADMINISTRATORS

System Administrators are essential for deployments of Sahana Eden. They also help us manage the servers where websites, wikis and demos are hosted. The system administration team has the opportunity to develop their skills while assisting the community.

TRANSLATORS

Translators can help to make Sahana Eden more accessible by translating it into multiple languages.

DESIGNERS

Designers can help to make Sahana Eden more usable by adding a clear graphics, layout and icons to improve the user experience.

GIS EXPERTS

GIS Experts can provide data and tools to improve GIS functionality in Sahana Eden, to ensure that key geographical knowledge is available.

20. CONTRIBUTING CODE

We welcome contributions to the Sahana Eden code. Before any contributions can be accepted, contributors must sign a Contributor License Agreement to ensure that the code can be provided as open source software. This protects both you as a contributor as well as the Sahana Software Foundation. This can be downloaded at http://wiki.sahanafoundation.org/doku.php/foundation:start#contributor_license_agreement

Sahana Eden code is hosted on GitHub at <https://github.com/flavour/eden> which uses the Git distributed version control system.

The bug tracker, which uses Trac, is at <http://eden.sahanafoundation.org/report>.

USING GITHUB

The initial GitHub configuration (1 & 2) is covered in the 'Installing a Developer Environment' chapter.

Commit your code to your local branch (3 & 4):

```
git commit -a
git commit -a
```

Before submitting it, you should ensure that it runs with the current version of the 'Trunk' branch (5):

```
git pull upstream master
```

You should also rebase it to keep the revision history cleaner and in more logical chunks for review (6):

```
git rebase -i
```

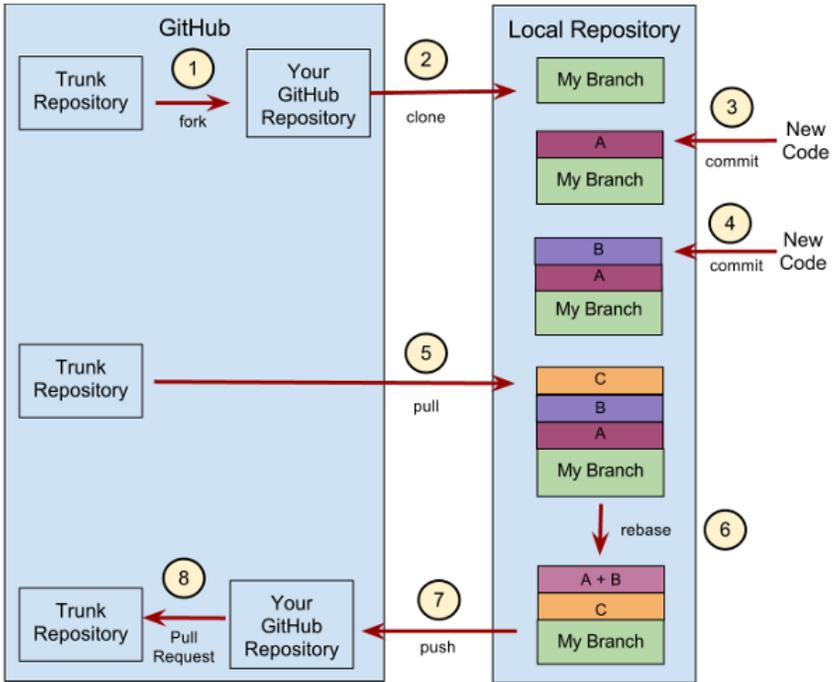
Push your code to your GitHub branch (7):

```
git push
```

You can then use GitHub to submit a 'Pull Request' to Trunk (8).

<https://github.com/mygitusername/eden/pull/new/master>

People subscribed to Trunk can then review the changes, request any necessary amendments before it can be accepted, and then merge your work into Trunk so that other users can benefit from your work.



21. WHERE TO GO NEXT

Thank you for learning more about the Sahana Eden project. We look forward to seeing you become more involved in our community. For the latest news about the project please see the Sahana Software Foundation Website (<http://sahanafoundation.org>).

APPENDICES

- 22. RESOURCE MODEL
- 23. MAPPING & GIS
- 24. SCHEDULER
- 25. SYNCHRONIZATION
- 26. WEB SERVICES
- 27. S3XML
- 28. GLOSSARY
- 29. CREDITS

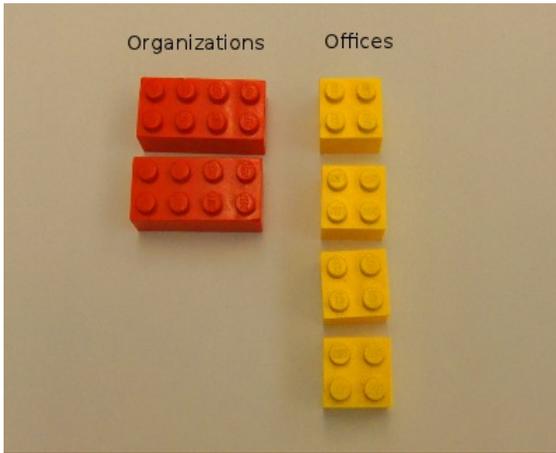
22. RESOURCE MODEL

To manage all the complex information in a broad variety of business processes in a flexible yet consistent way, the Sahana Eden framework uses the concept of "resources".

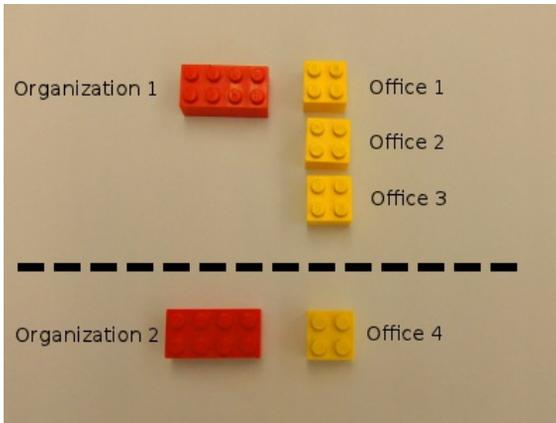
A *resource* is a set of all database records which describe a complex entity in the business process (such as a person or an organization, or a request for items). *Resources* also provide all necessary methods to represent, modify and analyze the data.

Records to Resources

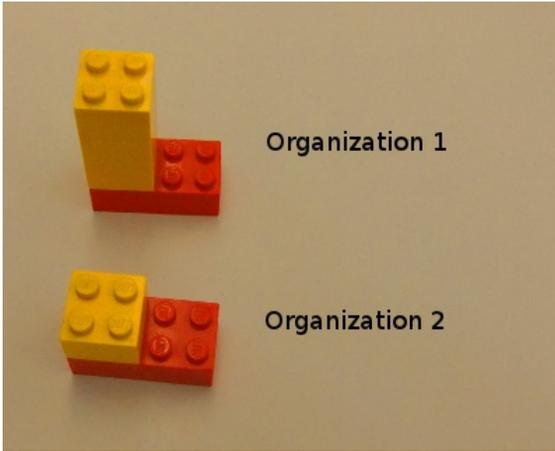
The following pictures illustrate the relationship between records in database tables and *resources*. Each Lego piece represents a single record in the database, while the different colors and sizes symbolize different database tables:



The next picture shows the relationship between the records - every "organization" can have one or more "offices".



The last picture shows how each set of related records forms an instance of the "organization" resource.



The database tables which are involved in a *resource* are called "components" of the *resource*. There is always one "master" component, typically containing the basic details, for example, a name or identification of the entity which the resource describes.

In the user interface, the resource concept is typically represented by navigation tabs:

Organization: Example.co.uk **Sector(s):** Telecommunications
Type: International NGO

Basic Details | Staff & Volunteers | Offices | Assessments | Projects | Activities | Tasks

Delete Organization Last up

*** Required Fields**

Name: *

Acronym:

[? HELP](#)

This shows the "Organization" *resource*, with its *components* as navigation tabs. Each tab handles a set of records in a different database table. All the records in all of the tabs together form the "Organization" resource.

The Sahana Eden Framework provides a set of integrated tools for the implementation of resources and resource methods such as CRUD (create, read, update, delete), Search, Mapping and Reporting, Data Export and Import as well as Web Services. This allows developers to rapidly develop new solutions or adapt existing modules to new requirements.

23. MAPPING & GIS

Seeing data located on a Map helps decision makers to be able to make more meaningful decisions. Sahana Eden's Mapping Client can combine data from both it's own database & a range of external sources to provide a rich environment for display and analysis.

The GIS community within Sahana which has it's own home on the wiki:

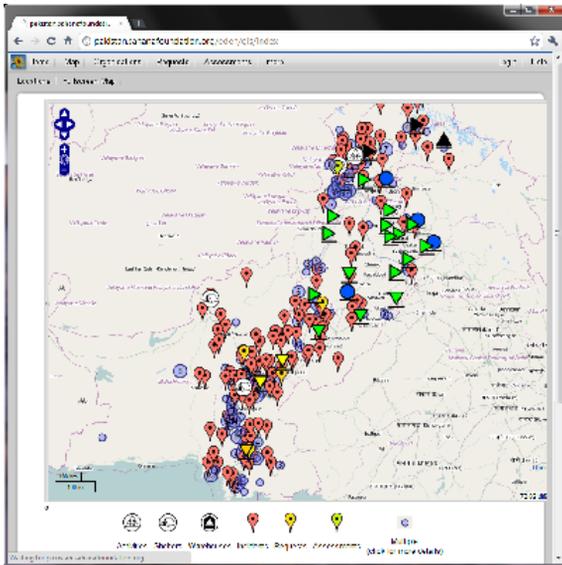
<http://eden.sahanafoundation.org/wiki/GIS>

MAP VIEWING CLIENT

Sahana Eden's mapping client is based on [OpenLayers](#) & [GeoExt](#).

OpenLayers provides access to a wide range of data sources, from public services like OpenStreetMap, Google Maps & Bing Maps through to GIS services based on OGC standards like WMS & WFS or feeds from other systems exposed as KML or GeoRSS.

GeoExt provides UI widgets to allow the user to interface with the map.



MAP SERVICE CATALOG

These different 'Layers' are defined in Sahana Eden's 'Map Service Catalog', from which users can select which layers should be active in the client.

Map Layers using data from within the database simply requests the data through Web Services formatted as GeoJSON.

If multiple markers appear at the same location then they are 'clustered' together.

A 'refresh' strategy is available to reload the layer periodically so that a wall-mounted display can just reload the active layer(s) rather than the static basemap.

SPATIAL INFRASTRUCTURE

Sahana Eden's mapping features are even more powerful when coupled with a [GeoServer](#) installation as this can expose many different data sources, such as Shapefiles or Topography Rasters, as WMS or WFS for ready display in Sahana.

Another very useful tool to complete an infrastructure is [MapProxy](#) as this allows WMS layers from external sources to be reprojected to be compatible with other data sources (typically allowing a WGS84 service to be accessible as an overlay with a Spherical Mercator basemap, such as OpenStreetMap or Google Maps).

Tip: Sahana Eden supports multiple Projections, but can only display one at a time!

This combination is what is used for the IFRC's Resource Mapping System.

CONFIGURATION

The initial configuration is defined in `models/000_config.py`. This then populates the `gis_config` table which is where subsequent modification should be done for this instance.

The system configuration can be inherited for both Personal configurations & Event configurations. There is also the option for Country-based configurations to store the labels for the different levels of the hierarchy.

Currently the selection of active layers from the Catalog is global, but this is planned to be made per-Config.

LOCATION HIERARCHY

All locations are stored in the `gis_location` table, to which other resources link through the `location_id()` reusable field (foreign key). The location records are hierarchical through the use of 'parent' & 'path' fields. There are optimised routines in `modules/s3/s3gis.py` to populate & search these fields. This hierarchy is flexible to accommodate different countries:

- L0: Country
- L1: State or Province
- L2: District or County
- L3: City, Town or Village
- L4: Neighborhood

LOCATION SELECTOR

A Location Selector widget allows a simple `location_id()` reusable field in a source to provide an inline form to be able to select existing locations or create new ones, which includes the hierarchy & the ability to pinpoint the location on a Map, the rough position for which can be obtained through either GeoCoding (lookup of the entered street address or hierarchy) or GeoLocation (detection of the current user's location by the browser).

The widget is defined in `modules/s3/s3widgets.py`.

This makes use of JavaScript in `static/scripts/S3/s3.locationselector.widget.js`. (Remember that any edits won't be visible unless you are running in debug mode or run the build script & refresh your browser cache)

API

As well as the main Mapping Client, there is an API to allow developers to be able to display customised data output relevant to a specific module on the Map.

This functionality is available via the GIS module's `show_map()` function.

A simple example for a controller function would be:

```
table = db.mymodule_myresource
query = (table.id > 0) & \
        (db.gis_location.id == table.location_id)
rows = db(query).select()
queries = [{name: "MyLayer",
            query: rows}]
map = gis.show_map(feature_queries=queries)
return dict(map=map)
```

& the view would include:

```
{{XML(map)}}
```

Full documentation for this API can be found in the source code or on the wiki:

<http://eden.sahanafoundation.org/wiki/DeveloperGuidelinesGIS>

FUTURE PLANS

There are plans to make use of optimised Spatial Queries when PostGIS is available by extending Web2Py's DAL. This will open up a range of possibilities for deeper analysis.

The Cube (pivot table) output is planned to be displayable on Maps - as both Shaded Polygons and Popups within centroid Markers containing the charts for that area.

The connection to GeoServer could be made more transparent by making use of it's REST API.

We could include a tool to be able to browse & select WMS services, auto-configuring an associated MapProxy to reproject, if-necessary.

24. SCHEDULER

The Scheduler allows non-interactive tasks to be run at specific times, repeatedly at regular intervals or once asynchronously for responsiveness of the user interface. This is required e.g. for Synchronization and Messaging.

The Scheduler is run as a separate web2py "worker" process.

It is normally run from `/etc/rc.local` as:

```
cd ~web2py && python web2py.py -K eden -Q  
>/dev/null 2>&1 &
```

Tip: This should normally be set to run automatically by the Installation Script.

Scheduler logs can be checked using the appadmin interface:

```
http://host.domain/eden/appadmin/select/db?  
query=db.scheduler_run.id%3E0
```

25. SYNCHRONIZATION

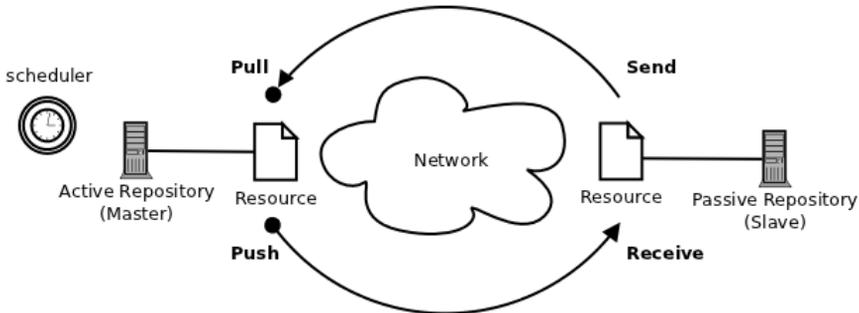
The Synchronization module allows the synchronization of data resources between Sahana Eden instances. Synchronization jobs can be configured to be run automatically in the background and at regular intervals, without disrupting the current operation of the sites.

This module is part of the site administration module, and requires administrator privileges to view or modify its configuration. The synchronization module requires web2py revision 3566 (1.99.0) or newer.

OVERVIEW

The synchronization process is controlled entirely by the "active" Sahana Eden instance (master instance).

The active Eden instance runs the scheduler process, and initiates the update requests when they are due, while the passive repository (slave instance) merely responds to these requests.



The active Eden instance first downloads the available updates from the passive repository (*pull*) and imports them into the local database, and then uploads all available updates from the local database to the passive repository (*push*).

Both *pull* and *push* are each a RESTful HTTP-request, using S3XML as data format.

SYNCHRONIZATION HOMEPAGE

Login as administrator and open the *Administration* menu. In the left menu, you will then find the following entries:

Synchronization
Settings
Repositories
Log

Click on *Synchronization* here to open the homepage of the Synchronization Module:

Synchronization

The synchronization module allows the synchronization of data resources between Sahana Eden instances.

Settings

- See the universally unique identifier (UUID) of this repository
- Configure the default proxy server to connect to remote repositories

Repositories

- Configure connection details and authentication
- Configure resources to synchronize, update methods and policies
- Schedule synchronization jobs
- View log entries per repository

Log

- View all log entries

User Guidelines Synchronization

- See a detailed description of the module on the Sahana Eden wiki

CONFIGURATION

Follow this checklist to configure synchronization:

1. Check the **Prerequisites**
2. Make sure the *passive* site is up and running, and reachable over the network
3. Login as administrator at the *active* site and
 1. Configure the default proxy server in **Synchronization Settings** as needed
 2. Register the passive site in **Repository Configuration**
 3. Configure the resources to synchronize in **Resource Configuration**
4. Set up the **Synchronization Schedule**
4. Ensure you have a **Worker** process running at the *active* site

Prerequisites

Both sites must have Sahana Eden installed and running. To avoid problems with different database structures, both Sahana Eden instances should always use the same version of the software.

Important: It is important that the **system clocks** in both sites are **synchronized** with each other, which can best be achieved by synchronizing both sites with the same NTP service.

Decide which one is the *active* and which one is the *passive* instance. The *passive* instance is typically a permanently and publicly accessible Sahana Eden instance, while the *active* instance could be a protected Eden installation (e.g. behind a firewall), or one with only temporary network access (e.g. on a notebook).

While performing synchronization jobs, the "active" site must be able to establish a connection to the "passive" site over the network using HTTP (or HTTPS).

If a proxy server has to be used for the HTTP connection, this can be configured in the Synchronization Settings (proxy authentication is currently not supported).

Check that both instances have the synchronization module enabled in the `private/templates/<templatename>/config.py` file. If the `sync` section is missing from the `settings.modules` dict, then add it as follows:

```
settings.modules = OrderedDict([
    ...
    # Add or uncomment this section, if it is missing or commented:
    ("sync", Storage(
        name_nice = T("Synchronization"),
        description = T("Synchronization"),
        restricted = True,
        access = "|1|", # Only Administrators can see this
        module in the default menu & access the controller
        module_type = 0 # This item is handled separately for
        the menu
    )),
    ...
])
```

Synchronization Settings

Go to the [Synchronization Homepage](#) and click *Settings* to open this page:

Edit Synchronization Settings Last updated on 2011-09-20 07:24:17 by admin@example.com

Proxy Server URL:
 ? HELP

UUID:
 ? HELP

This page shows you the *UUID* (universally unique identifier) of the repository you are logged in at. You will need this identifier to register the repository at a peer site. The *UUID* is created during the first run of a Sahana Eden instance, and cannot be changed.

If needed, enter the complete URL of the proxy server (including port number if not 80) that is to be used when connecting to the *passive* site (this is only necessary at the *active* site). Click *Save* to update the configuration.

Repository Configuration

Go to the **Synchronization Homepage** and click *Repositories*. This will show you a list of all currently configured repositories:

The screenshot shows a web interface titled "Repositories". At the top left is a button "Add Repository". Below it is the section "Currently Configured Repositories". There is a search bar and a "Show 10 entries" dropdown. A table lists the repository "Gemma" with its UUID, "Accept Pushes" set to "True", and "Last Synchronization" on "2011-09-20 12:15:36". Below the table are navigation buttons: "Open", "Delete", "First", "Previous", "1", "Next", "Last".

To view and/or modify the configuration for a repository, click the *Open* button in the respective row in the list.

By clicking *Add Repository*, you can register a new repository:

The screenshot shows the "Add Repository" form. It has a section for "* Required Fields" with the following fields: "Name: *" (text input), "URL: *" (text input), "Username:" (text input), "Password:" (text input), "Proxy Server URL:" (text input), "Accept Pushes:" (checkbox), and "UUID: *" (text input). Each field has a "? HELP" link to its right. A "Save" button is at the bottom.

Fill in the fields as follows:

Field	Instructions	at the active site	at the passive site
Name	Enter a name for the repository(for your own reference)	required	required
URL	Enter the URL of the repository (base URL of the Sahana Eden instance, e.g. http://www.example.org/eden)	required	
Username	Enter the username to authenticate at the repository	required	
Password	Enter the password to authenticate at the repository	required	
Proxy Server	Enter the URL of a proxy server to connect to the repository, if different from the Synchronization	fill in as needed	

Settings			
Accept Pushes	check this if the repository is allowed to push updates		set as needed
UUID	Enter the UUID from the Synchronization Settings of the repository	required	required

Resource Configuration

Go to the **Synchronization Homepage**, click *Repositories*, then *Open* the repository you want to configure a resource for, and change to the *Resources* tab:

Repository Configuration

Name: Gemma
URL: http://eden.hq.nursix/eden

Configuration Resources Schedule Log

Add Resource

*** Required Fields**

Resource Name: *
dvi_recreq ? HELP

Last synchronized on:
None

Mode:
pull and push ? HELP

Strategy:
 create ? HELP
 update ? HELP
 delete

Update Policy:
NEWER ? HELP

Conflict Policy:
NEWER ? HELP

Save

Fill in the fields as follows:

Field	Instructions	Example
Resource Name	Fill in the name of the master table of the resource. Details can be found in the documentation for the data model of your Sahana Eden application	req_req
Mode	Select the synchronization mode you wish to activate - pull, push or both. See Method Overview to understand the mode	pull and push
Strategy	Choose the import methods you wish to allow for the synchronization of this resource	create, update, delete
Update Policy	Choose in which situation records shall be updated, see explanations below	NEWER
Conflict Policy	Choose in which situation records shall be updated in case of conflicts, see explanations below	NEWER

Update Policy

If a record has been modified in one of the repositories, then the synchronization process has to decide whether to update the other repository with the new data or not. For this decision you can define a policy:

Policy	Meaning
THIS	Always update the remote repository with the local version of the record (overwrite remote updates)
NEWER	Update both repositories to the newest version of the record (keep the newer data)
MASTER	Update the record on either side only if the other side has originated the record (keep the master data)
OTHER	Always update the local repository with the remote version of the record (overwrite local updates)

Usually, you would choose "NEWER" here unless you have a good reason to do otherwise.

Conflict Policy

If a record has been modified both in the local repository *and* the remote repository since the last synchronization time, then this is called a *conflict* situation, in which two concurrent record updates are available at the same time. You can define a policy for which of the updates to apply, similar to the **Update Policy**.

If you do not know what to select here, it is reasonable to choose the same option as for the **Update Policy**.

Policy Transfer

In most situations, you would want both repositories to apply the same policies. This is the default behavior - the policies from the active site are reported to the passive site during the synchronization, and are applied there as well (THIS and OTHER are replaced by the respective opposite at the passive site, of course).

If for some reason you need to define different policies at the passive site, then you have to configure the same resource at the passive site as well, and choose the policies explicitly.

Synchronization Schedule

Go to the **Synchronization Homepage**, click *Repositories*, then *Open* the repository configuration you want to schedule a synchronization job for and change to the *Schedule* tab. If there are already jobs configured for this repository, you will see a list of those jobs. Otherwise (or by clicking *Add Job*), you get to this form:

With every *Job*, all resources configured for this repository will be synchronized.

Fill in the fields as follows:

Field	Instructions	Example
Enabled	Set to True if the job shall actually be run, or set False to disable the job	True
Start Time	Select date and time for the first run of this job (UTC)	2011-09-21 08:30
End Time	Select date and time after which the job shall not be run anymore (UTC)	2012-09-21 08:30
Repeat <i>n</i> times	Select how often the job shall be run, set to 0 to set no limit	0
Run every	Select the time interval after which to repeat the job	5 minutes
Timeout	Set a maximum time after which to abort the action	600 seconds

If you need to switch between jobs (e.g. for maintenance periods, low-traffic periods), you can set up multiple schedules, and disable/enable them as needed.

To consider:

You should choose meaningful time interval and timeout settings: the more resources are to be synchronized, the longer it will take (in this regard, also note that THIS- and OTHER-policies will always exchange all records in a resource, thus taking significantly longer).

How many records have to be exchanged per run depends on the average update frequency and the time interval between synchronizations: e.g. if there are on average 100 record updates per minute, and you set a 2-minute interval, then there would be 200 records on average to be transmitted every run. The import rate on a small server has been tested at on average 18 records/second, which means, the synchronization process would take around 11 seconds in this case. To be on the safe side, choose a timeout value at least 10 times as high as that - e.g. 120 seconds.

Note that the network traffic arising from synchronization does not mainly depend on the frequency of synchronization, but on the record update rate at the sites. Smaller synchronization intervals would increase the traffic only slightly, but reduce the rate of conflicts and the risk of network-related problems. However, too small intervals (below the update rate of the site) may cause unnecessary network traffic with just empty transmissions.

Worker

The scheduled synchronization jobs are performed by a separate asynchronous `web2py worker` process at the *active* site. Make sure the worker process for the Scheduler is running at the active site, see chapter on Scheduler.

SYNCHRONIZATION LOG

Go to the **Synchronization Homepage** and click *Log*. This shows you a list of all prior log entries for all repositories.

If you instead want to see the log entries only for a particular repository, go to the Synchronization Homepage, click *Repositories*, then *Open* the respective repository configuration and go to the *Log* tab:

Repository Configuration

Name: Gemma
URL: http://eden.hq.nursix/eden

Configuration Resources Schedule **Log**

Synchronization Log

Search: Show 10 entries Showing 1 to 10 of 324 entries

	Date/Time	Resource Name	Mode	Action	Result	Remote Error	Message
Details	2011-09-20 12:15:37	dvi_recreq	pull/outgoing	none	success	False	data imported successfully
Details	2011-09-20 12:15:37	dvi_recreq	push/outgoing	none	success	False	data sent successfully
Details	2011-09-20 12:13:37	dvi_recreq	pull/outgoing	none	success	False	data imported successfully
Details	2011-09-20 12:13:37	dvi_recreq	push/outgoing	none	success	False	data sent successfully
Details	2011-09-20 12:11:38	dvi_recreq	push/outgoing	none	success	False	data sent successfully
Details	2011-09-20 12:11:37	dvi_recreq	pull/outgoing	none	success	False	data imported successfully
Details	2011-09-20 12:09:38	dvi_recreq	push/outgoing	none	success	False	data sent successfully
Details	2011-09-20 12:09:37	dvi_recreq	pull/outgoing	none	success	False	data imported successfully
Details	2011-09-20 12:07:38	dvi_recreq	push/outgoing	none	success	False	data sent successfully
Details	2011-09-20 12:07:37	dvi_recreq	pull/outgoing	none	success	False	data imported successfully

First Previous **1** 2 3 4 5 Next Last Showing 1 to 10 of 324 entries

Note: the newest entries are shown on top of the list.

Click on *Details* for a log entry to see the complete entry:

Log Entry

Date/Time
2011-09-20 12:15:37

Repository
Gemma

Resource Name
dvi_recreq

Mode
pull/outgoing

Action
none

Result
success

Remote Error

Message
data imported successfully

Read the entries as follows:

.....

Item	Explanation
Date/Time	Date and time of the transaction
Repository	Name of the repository synchronized with
Resource Name	Name of the resource synchronized
Mode	Transaction mode (pull or push) and direction of transmission (incoming or outgoing)
Action	Action performed to resolve problems (if any)
Result	Result of the transaction
Remote Error	Was this error at <i>this</i> site or at the repository synchronized with?
Message	The log message

26. WEB SERVICES

Web Services in Sahana Eden are implemented as a RESTful API (Application Programming Interface).

This API allows other applications to access and manipulate Sahana Eden data resources directly over the web using the HTTP protocol, which means:

- URLs to address resources
- HTTP requests to perform actions and transfer data
- HTTP method verbs (GET, PUT, POST, DELETE) to specify the actions to be performed
- HTTP status codes to report status and errors

A powerful query language is available to address particular data elements:

```
/eden/pr/person?person.first_name__like=Miriam
```

(all person records where the first name contains "Miriam")

...or to specify method parameters:

```
/eden/org/office/analyze?  
col=location_id&row=type&fact=name&aggregate=group_concat
```

(a pivot table of offices grouped by facility type vs. location)

The RESTful API uses Sahana Eden's native S3XML format for data exchange. Other XML or JSON formats are supported using on-the-fly XSLT transformation - Sahana Eden provides built-in XSLT stylesheets for a variety of XML standards (e.g. KML, EDXL-HAVE), and can also accept custom stylesheets.

URL FORMAT

Basic URL Syntax

Example of a URL to address a resource in the Sahana Eden RESTful API:

```
http://vita.sahanafoundation.org/eden/hms/hospital/1/bed_capacity/create
```

The basic URL format is:

(Parts in { } are optional, [A | B] indicates alternatives)

```
http:// server / path / prefix / name { /<arguments> } { ?<query> }
```

- **server** (*vita.sahanafoundation.org*) is the server domain name
- **path** (*/eden*) the path to the application
- **prefix** (*/hms*) is the name (prefix) of the Sahana Eden module
- **name** (*/hospital*) is the resource name

The **<arguments>** list consists of:

```
{ /id } { / [ method | component { /component_id } { /method } ] }  
{ .format }
```

- **id** (/) is a record ID in the master table of the resource
- **component** (/bed_capacity) is a component name
- **component_id** is a record ID in the component
- **method** (/create) is a method of the resource
- **format** specifies the requested data format (e.g.: ".xml" for S3XML)

For the <query> syntax, see the following section.

Basic Query Format

Example of a URL query:

```
?hospital.name__like=Example%20Hospital
```

...as part the complete URL:

```
http://vita.sahanafoundation.org/eden/hms/hospital?
hospital.name__like=Example%20Hospital
```

The basic query format is:

```
?resource.{foreign key$}field{operator}=value(s)
```

- **resource** (*hospital*) is the name of the component, followed by a period (.)
- **foreign key** is the name of the foreign key field followed by a dollar sign (to filter against a value in the table referenced by this foreign key)
- **field** (*name*) is the name of the field in the target table
- **operator** (*_like*) is the operator
- **value(s)** (*Example%20Hospital*) is the value or a comma-separated list of values to test against (a comma will be treated as an OR)

Note that special characters in values must be properly URL-encoded (the %20 in this example stands for a blank).

Supported operators:

Operator	Method	Comments
__eq	equal, =	can be omitted
__ne	not equal, !=	
__lt	less than, <	numeric and date/time types only
__le	less than or equal, <=	numeric and date/time types only
__gt	greater than, >	numeric and date/time types only
__ge	greater than or equal, >=	numeric and date/time types only
__like	wildcard comparison, LIKE(%value%)	string/text types only
__unlike	negative wildcard comparison, NOT LIKE(%value%)	string/text types only
__in	containment, contains(value)	list types only
__ex	negative containment, excludes(value)	list types only

Other Queries

Boundary Box Queries

For resources with location references (e.g. Hospitals), you can use boundary box queries to select records. The general format of the query variable is:

```
?bbox=minLon,minLat,maxLon,maxLat
```

You can also specify the foreign key field name of the location reference the query relates to (e.g. in case there are multiple location references in that resource):

```
?bbox.FKFieldName=minLon,minLat,maxLon,maxLat
```

Examples:

```
/hms/hospital?bbox=123,13.5,124,13.7
```

```
/hms/hospital?bbox.location_id=123,13.5,124,13.7
```

URL Examples

Interactive (HTML) Format

All "person" records in the Person Registry module (pr):

```
http://localhost:8000/eden/pr/person
```

Non-interactive Formats

All "person" records in the Person Registry module (pr), in other data formats:

```
http://localhost:8000/eden/pr/person.pdf
```

```
http://localhost:8000/eden/pr/person.rss
```

```
http://localhost:8000/eden/pr/person.xml
```

Record by ID

```
http://localhost:8000/eden/pr/person/1
```

```
http://localhost:8000/eden/pr/person/1.pfif
```

```
http://localhost:8000/eden/pr/person/1.xml
```

Record by UUID

```
http://localhost:8000/eden/pr/person?person.uuid=urn:uuid:839bab5a-a401-4be3-8616-27fbc1810ef4
```

URL Queries

```
http://localhost:8000/eden/pr/person?person.id=1,2,3
```

```
http://localhost:8000/eden/org/office?
```

```
office.type=None,1,2&office.obsolete=False
```

```
http://localhost:8000/eden/org/office?
```

```
office.modified_on__gt=20110926T10:00:00
```

Note that in URL queries, date/time values must be in UTC and use the ISO-8601 combined format YYYYMMDDThh:mm:ss.

STANDARD METHODS

Standard methods include:

- interactive create, read, update and delete (CRUD) including list views
- data export/import in various formats, including on-the-fly transformation

Note:

- in XML and JSON, resources are always exported/imported including all their components and referenced resources.
- in all other formats, the components need to be addressed separately

GET

Interactive Formats

Interactive formats are HTML (Extension ".html"), or PLAIN (Extension ".plain"). If no format extension is specified, HTML format is assumed.

- without **method** specified in the URL:
 - if no record ID in the URL: **list** view of the resource
 - with a record ID in the URL: **read** view of the specified record (if the user is permitted to update the record, an **update** form returned instead)

Example: read view of the person record #1:

`http://localhost:8000/eden/pr/person/1`

- with **method** specified in the URL:
 - method **create** returns a create-form
 - method **read** returns a view of the specified record (other than with blank method, no update form is returned in this case)
 - method **update** returns an update form for the specified record
 - method **delete** returns a delete confirmation form together with a list of the specified records (if there is exactly one record identified by its ID in the URL, then the record will be deleted instead of a form being returned, see POST)

Example: create form for a new person record:

`http://localhost:8000/eden/pr/person/create`

- some resources support other methods, e.g.
 - **search** returns a search form for the resource

Example: search for a person by name or ID:

`http://localhost:8000/eden/pr/person/search`

Non-interactive formats

Any format extension that is not listed under the interactive formats, is treated as non-interactive.

- without **method** in the URL:
 - returns all matching records in the specified format

Example: all person records the user is allowed to read, in S3XML format:

GET `http://localhost:8000/eden/pr/person.xml`

- with **method** in the URL:
 - method **create** or **update** returns a schema document of the resource
 - method **create** or **update** together with a source imports the data into the specified resource (see chapter S3XML)
 - method **options** returns a field options document of the resource
 - other methods are not supported

Example: get a schema document of the person resource:

```
GET http://localhost:8000/eden/pr/person/create.xml
```

- XLS and PDF formats work read-only (create/update/delete being ignored)

POST

Interactive Formats

- performs the respective **method** (if specified in the request)
 - method **create** creates a new record
 - method **update** updates the specified record
 - method **delete** deletes the specified record
- expects the form data as multi-part request body

Non-interactive Formats

- enters an interactive review of the source data before importing the data into the resource

PUT

Interactive formats

- see POST

Non-interactive formats

- import data from the request body (which must be in the specified format) into the resource
- records being matched by the UIDs specified in the data, while any record IDs in the URL restrict the selection

DELETE

- deletes those of the addressed records which are deletable by the current user

AUTHORIZATION

It is possible to access privileged resources by providing the username/password within each request, rather than the usual method of having the login stored within the session.

The RESTful API supports HTTP Basic Authentication (http://en.wikipedia.org/wiki/Basic_access_authentication).

Note: for some command-line tools like *wget* or *RESTClient* you might need to additionally activate an option for pre-emptive authentication (unsolicited sending of credentials). E.g. for *wget*, use the *--auth-no-challenge* option.

AJAX Examples

Here some examples how to add the HTTP Basic Authentication header to AJAX requests:

```
function make_base_auth(user, password) {
    var tok = user + ':' + pass;
    var hash = Base64.encode(tok);
    return 'Basic ' + hash;
}

var auth = make_basic_auth('username@example.com', 'password');
var url = 'http://host.domain/eden/controller/function?vars';

// RAW
request = new XMLHttpRequest();
request.setRequestHeader('Authorization', auth);
request.open('PUT', url, true); // async=true
request.send(bodyContent);

// ExtJS
Ext.Ajax.request({
    url : url,
    method : 'GET',
    headers : { Authorization : auth }
});

// jQuery
$.ajax({
    url : url,
    method : 'GET',
    beforeSend : function(req) {
        req.setRequestHeader('Authorization', auth);
    }
});
```

ERROR HANDLING

The HTTP status code in the response indicates the success or failure of a request:

Status Code	Cause	Response Body
200 OK	Success	results or JSON message
400 BAD REQUEST	Syntax error or method not supported for the specified resource	JSON message
401 UNAUTHORIZED	Authorization required	Clear text error
403 FORBIDDEN	Insufficient permissions	Clear text error
404 NOT FOUND	Non-existent Resource	Clear text error
50x	Unrecoverable internal error	Ticket issued or clear text error

Where a JSON message is returned, it has the following structure:

```
{
  success= "True" | "False",
  statuscode = "XXX",
  message = "clear text error message",
  tree = {
    /* element tree */
  }
}
```

If there was an input element tree and it contained any errors, a subtree with the invalid elements will be added to the JSON message ("tree"). This subtree is expressed in [JSON Format](#). Invalid elements will have an additional `@error` attribute containing a clear-text error description.

Skipping invalid records at import:

By default, an import request will be rolled back (completely) and an HTTP 400 BAD REQUEST error be returned if the source contains any invalid data.

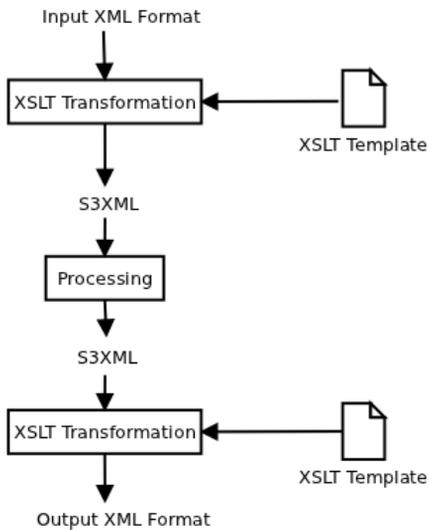
You can override this behavior by using the `ignore_errors` URL variable (`?ignore_errors=True`). Invalid records will then be skipped, while the valid records will be committed to the database and the request returns a HTTP 200 OK. The JSON message in the response body would however contain the error message and the element tree with the invalid elements.

Note that `ignore_errors` applies to Validation Errors only. Any other type of error (e.g. XML syntax error) will be handled as usual (=rollback + error message).

The `ignore_errors` option is meant for "dirty" data, e.g. cases where you need to import from a source but do not have permission and/or means to clean it up before import. In all other cases, where possible, you should avoid `ignore_errors` and rather sanitize the source.

S3XML ON-THE-FLY TRANSFORMATION

The Sahana Eden RESTful API can perform XSLT transformation of XML sources into the S3XML format on-the-fly when exporting or importing data.



The XSLT stylesheets to use for this transformation can be:

- a static file on the server or a web URL
- a file attached to the request (at import)
- an integrated stylesheet of Eden (folder static/formats)

Note:

- You cannot use the .xml or .json extension if the source is to be transformed. By these extensions, the interface assumes the source is already S3XML.
- Non-XML formats such as PDF or XLS do not support on-the-fly transformation.

Integrated Transformation Stylesheets

Sahana Eden provides a number of internal XSLT stylesheets for various formats. These will be automatically used if no other stylesheet is specified (fallback). The internal stylesheets reside in:

- static/formats/<format>/export.xml to transform S3XML into another format
- static/formats/<format>/import.xml to transform another format into S3XML

Example:

- static/formats/have/import.xml transforms EDXL-HAVE (*.have) into S3XML
- static/formats/have/export.xml transforms S3XML into EDXL-HAVE

XSLT Stylesheets on the Server

You can use the URL variable *transform* to specify a transformation stylesheet at a file system location on the server, e.g.:

`http://127.0.0.1:8000/eden/pr/person.pfif?transform=/home/dominic/stylesheet/pfif.xml`

...or at another location on the web, e.g.:

```
http://127.0.0.1:8000/eden/pr/person.rss?  
transform=http://pub.nursix.org/eden/formats/rss.xsl
```

If you use *transform* to specify the stylesheet location explicitly, any existing internal stylesheet for the format extension would not be used.

Note:

- the Eden web server must be permitted to access the stylesheet without authentication
- the request must not use ".xml" as data format extension, otherwise no transformation will be performed at all (*transform* would be ignored then)

This feature is especially useful to create custom feeds to integrate into remote sites, e.g. RSS:

- create your own *rss.xsl* to transform S3XML into RSS
- place it on a public (e.g. your own) web server, e.g.:

```
http://www.example.org/eden-access/rss.xsl
```

- provide a feed link to Eden resources using your RSS stylesheet:

```
http://edensite.org/eden/hms/hospital.rss?transform=http://www.mysite.org/eden-  
access/rss.xsl
```

This does work with any XML format, e.g. KML - if you wanted to provide a map rather than a feed link.

Attached XSLT Stylesheets

For data import, an XSLT stylesheet to transform foreign XML into S3XML can be attached to the request.

The filename of the attached stylesheet is expected to be `<resourcename>.xsl`, where `<resourcename>` is the name of the target resource (without module prefix, e.g. `person` or `hospital`).

Note:

- `document()`, `xsl:include` and `xsl:import` will need absolute paths in this case (safer to avoid these)

27. S3XML

S3XML is a data exchange format for Sahana Eden.

S3XML is a meta-format and does not specify any particular data elements. The interface is entirely introspective to the underlying data model, thus the specific constraints defined in the data model also apply for S3XML documents.

CONVENTIONS

Name Space

In the current implementation of S3XML, no name space identifier shall be used. Where a name space identifier for the native S3XML format is needed (e.g. when embedding S3XML in other XML), it shall be:

```
xmlns:s3xml="http://eden.sahanafoundation.org/wiki/S3XML"
```

Character Encoding

Generally, XML documents can specify their character encoding in the XML header:

```
<?xml version="1.0" encoding="utf-8"?>
```

Sources in non-XML formats (JSON, CSV) used with S3XML on-the-fly conversion/transformation are expected to be UTF-8 encoded.

All exported data are always UTF-8 encoded.

IMPORT SOURCES

There are 3 different ways to specify or submit data sources for import:

Files on the Server

A source file in the server file system can be specified using the *filename* URL variable:

```
PUT http://<server>/<controller>/<resource>.xml?filename=<path>
```

Multiple files can be specified as list of comma-separated pathnames:

```
PUT http://<server>/<controller>/<resource>.xml?filename=<path>,<path>
```

Remote Files

A source file can be specified by its URL using the *fetchurl* URL variable:

```
PUT http://<server>/<controller>/<resource>.xml?fetchurl=<url>
```

Multiple files can be specified as list of comma-separated pathnames:

```
PUT http://<server>/<controller>/<resource>.xml?fetchurl=<url>,<url>
```

Supported protocols are http, ftp and file, where file is interpreted in the server file system context. URLs of different protocols can be mixed.

The specified URLs must be accessible either without authentication, or (if you specify credentials in the URLs) they must support unsolicited HTTP basic authentication - HTTP 403 retries are not handled by the interface.

The URLs must be properly quoted (see http://www.w3schools.com/tags/ref_urlencode.asp for more details), and must not contain commas.

Request Attachments

Source files can also be attached to a multipart-request. In this case the file extension of the source file must match the request URL file extension. Multiple files can be attached.

Multiple Sources

Where multiple sources are specified or attached, they are first converted and transformed one-by-one and then combined into a single element tree before import.

DUPLICATE RESOLUTION

The S3XML Importer does not handle duplicates within the same source. As the order of elements in the resulting element tree is not defined, and the last update time attribute is not mandatory in source elements, there is no predictable rule of precedence.

Records in the source must not be fractionated, but submitted in one element. Fractions of records will not be merged by the Importer, and which of the fractions finally would be imported is not predictable

Source elements using unique keys are automatically matched with existing records. Where the match is ambiguous (e.g. a set of keys matching multiple existing records), the import element will be rejected as invalid. For certain resources, the server may have additional duplicate finders and resolvers configured. How duplicates are handled by these resolvers, can differ from resource to resource.

The duplicate resolution strategy in standard import mode is to update the existing record with the values from the source record. In synchronization mode the default strategy is to accept/keep the newest data (the last update time attribute is mandatory in this case).

XML FORMAT

Document Types and Structure

S3XML defines 3 types of documents:

Document Type	Description
Schema Documents	describe the data schema for a resource
Field Option Documents	describe the currently acceptable options for fields in a record
Data Documents	provide the current contents (data) of resources

Schema Documents

Schema documents describe the data schema for a resource. Clients can use these documents e.g. for automatic generation of forms.

Schema documents can be retrieved from Sahana Eden by sending an empty GET request (i.e. without source) to the *create.xml* method of a resource, e.g.:

```
GET http://localhost:8000/eden/pr/person/create.xml
```

Document Tree:

```
<s3xml>
  <resource>
    <field>
      ...
    <resource>
      <field>
        ...
      </resource>
    </resource>
  </s3xml>
```

or (if requested from the *fields.xml* method):

```
<fields resource="name">
  <field/>
  <field/>
  <field/>
  ...
</fields>
```

Note:

- These documents can only be requested (GET), but not submitted for import
- Schema documents support on-the-fly transformation (see chapter **Web Services**)
- the URL query parameter *?options=true* adds a list of field options to those fields where options are defined, and combined with the parameter *&reference=true*, even options for foreign key references will be included
- the URL query parameter *?meta=true* will include the meta fields (as *<meta>* elements). In data documents, the meta fields appear as attributes of the *<resource>* element

Field Options Documents

Field options documents describe the currently acceptable options for fields in a record. Clients can use these documents e.g. for automatic generation and/or client-side validation of forms.

Field options documents can be requested from Sahana Eden by sending a GET request to the *options.xml* method of a resource, e.g.:

```
GET http://localhost:8000/eden/pr/person/options.xml
```

Document Tree:

```
<options>
  <select>
    <option>
    <option>
    <option>
    ...
  </select>
  <select>
    ...
  </select>
  ...
</options>
```

Note:

- the *field* URL variable can be used to specify a particular field in the resource, the enclosing `<options>` element would then be omitted (i.e. `<select>` becomes root element)
- on-the-fly transformation of field options documents is not supported
- Field option documents can only be requested (GET), but not submitted for import

Data Documents

Data documents provide the current contents (data) of resources.

Data documents can be requested from Sahana Eden by sending a GET request to the URL of the resource, e.g.:

```
GET http://localhost:8000/eden/pr/person.xml
```

Data documents can be submitted to Sahana Eden by sending PUT requests to the URL of the resource, e.g.:

```
PUT http://localhost:8000/eden/pr/person.xml
```

Note that sending data with POST will enter an interactive review of the source data before importing them, thus POST cannot be used by merely non-interactive clients.

Document Tree:

```
<s3xml>
  <resource> <!-- primary resource element -->
    <data> <!-- field data -->
    <data>
    ...
    <resource> <!-- component resource inside the primary resource -->
      <data>
      <data>
      <reference/> <!-- reference -->
      ...
    </resource>
  <reference/> <!-- reference -->
  <reference> <!-- reference with embedded resource element -->
    <resource>
      <data>
      ...
    </resource>
  </reference>
</resource>
</s3xml>
```

Components

Components of resources are `<resource>` elements nested inside the master `<resource>` element. Component records will be automatically imported and the required key references be added (=no explicit reference-element required).

Foreign key references of component records to their primary record will not be exported, and where they appear in import sources, they will be ignored.

Components of components are not allowed (maximum depth 1), and where they appear in import sources, they will be ignored.

References

Foreign key references (except those linking components to their primary record) are represented by `<reference>` elements.

Foreign keys can be importable UIDs (*uuid*-attribute, which will be both imported and used to find and/or link to existing records in the DB) or temporary UIDs (*tuid*-attribute, which will not be imported but only used to find records within the current tree). If a <resource> element with a matching UID key attribute is found in the same tree, it will be automatically imported.

References inside referenced elements will be resolved (unlimited depth) and also be imported. Circular references will be detected and properly resolved.

Multi-references (list:reference type in web2py) use a list of UID keys separated by vertical dashes like `uuid=|uid1|uid2|uid3|`. The leading and trailing vertical dashes must be present.

If a <resource> element is nested inside the <reference>, either or both of the UID keys can be omitted. Where both keys are however used, they must match. Multiple embedded <resource> elements are allowed for multi-references.

Element Descriptions

<s3xml>

This is the root element (in schema and data documents).

```
<s3xml success="true" results="2" domain="mycomputer"
url="http://127.0.0.1:8000/eden" latmin="-90.0" latmax="90.0" lonmin="-
180.0" lonmax="180.0">
...
</s3xml>
```

Parent elements: *none (root element)*

Child elements: <resource>

Contents: *empty*

Attributes:

Name	Type	Description	mandatory?
domain	string	the domain name of the data repository	no
url	string	the URL of the data repository	no
success	boolean	true if the page contains any records, otherwise false	no
results	integer	the total number of records matching the request	no
start	integer	the index of the first record returned (in paginated requests)	no
limit	integer	the maximum number of records returned (in paginated requests)	no
latmin, latmax, lonmin, lonmax	float	geo-location boundary box of the results	no

<resource>

This element represents a record (in data documents) or a database table (in schema documents).

```
<s3xml>
  <resource name="xxx_yyy">
    ...
  </resource>
</s3xml>
```

Parent elements: <s3xml>, <resource>, <reference>

Child elements: <resource>, <data>, <field>

Contents: *empty*

Attributes:

Name	Type	Description	mandatory?
name	string	the name of the database table	<i>yes</i>
uuid	string	a unique identifier for the record	<i>no*</i>
tuid	string	a temporary unique identifier for the record	<i>no*</i>
created_on	datetime	date and time when the record was created	<i>no**</i>
modified_on	datetime	date and time when the record was last updated	<i>no, default: time of the request** ***</i>
created_by	string	email-address of the user who created the record	<i>no</i>
modified_by	string	email-address of the user who last updated the record	<i>no</i>
owned_by_user	string	email-address of the user who owns the record****	<i>no</i>
owned_by_role	string	name of the user group who collectively own the record****	<i>no</i>
mci	integer	master-copy-index	<i>no, default: 2*** ****</i>

- (*) Records will be identified within the input file by their *uuid*, or, if no *uuid* is specified, by their *tuid*.
- (**) as YYYY-MM-DDTHH:mm:ssZ, always UTC
- (***) the last update date/time and *mci* are required in synchronization
- (****) the master copy index specifies how often a record has been copied across sites, see below
- (*****) record ownership will be retained if the record owners can be matched against existing users/user groups

The *uuid* will be stored in the database together with the record. If *uuid* is present and matches an existing record in the database, then this record will be updated. If there's no match or no *uuid* specified in the *resource* element, then the importer will create a new record in the database (and automatically generate a *uuid* if required).

The `mci` - master-copy-index - indicates how often this record has been copied across sites:

- when importing a new record the `mci` value is always **imported** as-is from the source
- when updating a record, the `mci` of the database record remains unchanged
- the `mci` of a record is **exported** as its current database value + 1.
- the repository first creating a record sets `mci=0` in the database record, which appears as `mci=1` in the exported XML.
- a copying site then imports `mci=1` into its database, which appears as `mci=2` in its export XML, and so forth...

The `mci` can be used to filter records for whether they have been originated at a repository or not. If there's a fixed set of synchronization paths between a number of Sahana Eden instances, the `mci` can be used for conflict resolution. If the `mci` is not specified, it defaults to 2.

MCI handling is optional for non-synchronizing peers.

<data>

This element represents the value of a single field in the record.

```
<s3xml>
  <resource>
    <data field="fieldname" value="value">...</data>
  </resource>
</s3xml>
```

Parent elements: resource

Child elements: none (*leaf element*)

Contents: Text

Attributes:

Name	Type	Description	mandatory?
field	string	the field name in the record	yes
value	JSON	the native field value	no
url	URL	the URL to download the contents from*	no
filename	filename	the filename of the attached contents*	no

(*) If the field is for file upload, a `url` attribute should be provided to specify the location of the file. The importer will try to download and store the file (file transfer) from that URL (*pull*). It is also possible to send the file together with the HTTP request - in this case the `filename` must be specified instead of the `url` (*push*). The *push* variant for uploads is meant for peers which do not support pulling for some reason (e.g. mobile phones). Normal servers would always provide a URL for download in order to allow the consuming site to decide which files to download and when (saves bandwidth).

The text node in the `data` element provides a human-readable representation of the field value.

The *value* attribute contains a JSON representation of the field value, retaining the original data type (i.e. strings must be double-quoted) except for *date*, *time* and *datetime* values, which are to be represented as simple strings in the respective standard format (no double quotes). The standard format for *datetime* values is *YYYY-MM-ddTHH:mm:ssZ* (ISO format, UTC), *date* shall be represented as *YYYY-MM-dd*, and *time* as *HH:mm:ss*.

data elements representing passwords can contain the clear text password in the *value* attribute, or the encrypted password in the text node. Where a clear text password is given as *value* attribute, it will be stored encrypted, otherwise the password will be stored as-is. Note that clear-text representation of passwords will be accepted by the interface, but never be exported.

<reference>

Represents a foreign key reference.

```
<s3xml>
  <resource name="xxx_yyy">
    <reference field="xy" resource="aaabbb"
      uuid="urn:uuid:e4bcb9fd-d890-4f2f-b221-1d75fff79e2d"/>
    </resource>
  </s3xml>
```

Parent elements: <resource>

Child elements: <resource>

Contents: Text

Attributes:

Name	Type	Description	mandatory?
field	string	the field name in the record	yes
resource	string	the name of the referenced database table	yes
uuid	string	the unique identifier of the referenced record (foreign key)*	(yes)**
tuid	string	a temporary identifier for a referenced record (foreign key)*	(yes)**

(*) Referenced records would always be exported in the same output file. If a referenced record is found in the same input file, then it will be automatically imported.

(**) Records will be identified within the input file by their *uuid*, or, if no *uuid* is specified, by their *tuid*.

If the referenced record is enclosed in the *reference* element, then *uuid* and *tuid* can be omitted:

```
<s3xml>
  <resource name="xxxyyy">
    <!-- content of the record goes here -->
    <reference field="xy" resource="aaabbb">
      <resource name="aaabbb">
        <!-- content of the referenced record goes here -->
      </resource>
    </reference>
  </resource>
</s3xml>
```

28. GLOSSARY

Amazon's EC2 - Amazon's Elastic Compute Cloud, a hosted cloud service

API - Application Programming Interface, an interface which software programs can use to communicate with each other

CRUD - Create, Read, Update, Delete

Debian - A free and open source community based Linux distribution

GIS - Geographic Information System/Geographic Information Services

Git - A distributed revision control system

GitHub - A free and open source suite of tools that help people and teams to work together on software projects; code hosting, bug tracking, mailing lists, etc..

GSoC - Google Summer of Code, a program sponsored by Google that encourages college students to participate in open source software projects.

IDE - Integrated Development Environment. An IDE is a software application that provides comprehensive facilities to computer programmers for software development. An IDE will normally consist of a source code editor as well as facilities to access other development tools such as compiler and/or interpreter, build automation tools, a debugger etc.

IFRC - International Federation of Red Cross and Red Crescent Societies

Instance - A single installation of the Sahana Eden software whether it be on a single server, USB drive or virtual machine.

ISCRAM - Information Systems for Crisis Response and Management

Module - A part of the software that creates functionality in Sahana Eden.

NGO - Non-Government Organizations

Pootle - An online translation management tool

RAD - Rapid Application Development

Repository - A source for software packages

RESTful - Conforming to the REST constraints, see REST

Representational state transfer (REST) - a style of software architecture for distributed hypermedia systems, see http://en.wikipedia.org/wiki/Representational_state_transfer

Resource - Modules define different resources are sets of all database records which describe a complex entity in the business process such as a person or an organization, or a request for items. A module's controller contains functions which provide an interface to its resources. See "Resource Model" appendix for a more detailed explanation.

Super-Entity - Allows sharing components across multiple resources: Instead of having several foreign keys for different primary resources, the shared component contains only one foreign key to the link table, the so-called *super-key*. See

<http://eden.sahanafoundation.org/wiki/S3XRC/ModelExtensions/SuperEntities>

UNDAC - United Nations Disaster Assessment Coordination

UN OCHA- United Nations Office for the Coordination of Humanitarian Affairs

UUID - Universally Unique Identifier

Web2Py - A free and open source web framework for agile development of secure database-driven web applications; written and programmable in Python.

29. CREDITS

This book was written during a three-day book sprint held during the Google Summer of Code Documentation Summit in Mountain View, California between the 18th and 20th of October 2011 by a team of Sahana Developers with assistance from book writing experts.



Authors: (from left to right) Pat Tressel, Fran Boon, Shikhar Kohli, Dominic Koenig, Belinda Lopez, Eli Lev, Michael Howden, Anne Goldenberg (not present)

Subjects: Disaster Management System

Summary :

Sahana Eden is an open source software platform for Disaster Management practioners. It allows tracking the needs of the affected populations and coordinating the responding agencies and their resources. This book is targeted at decision makers looking for solutions, users about to deploy the platform and developers who want to contribute to the project.

Cover Art: Laleh Torabi

Publisher: Lulu.com

Type of Document: collective handbook

Language: English

License: MIT