OSHIP-1.0.1a.x Basic Usage Guide
and OSHIP-1.0.1a.x Developers Tour

remember this is an alpha code implementation

SECURITY WARNING

OSHIP ships in development mode. There are many site specific security measures that need to be setup in your application before shipping a production healthcare application. The developers of these applications must be aware of and take precautions to use the vast security infrastructure available in this product.

Now on to the Show

First I would like to ask you to at least review the reference links in this document. While OSHIP itself is in 'alpha' condition. The concepts and tools presented in the links represent what is possible in a short period of time.

You should have already successfully installed and started your OSHIP server by now (See INSTALLATION.txt).
You will have seen only a generic Zope3 interface at this point.

Next you should shut down the server (Ctrl-C in the terminal window). You may need to manually remove the lock file; Data.fs.lock in the oship/var directory.

You will need to tell the archetype builder where all of the python eggs are for this installation. The bin/oship-ctl script does that to start the server. So we are going to make a copy of it.

$cp bin/oship-ctl src/oship/initialize
$cd src/oship
Open the initialize script in your editor. As you can see this script sets your sys.path to the eggs directories. At the very bottom of this script you will see:

```
import oship.startup

if __name__ == '__main__':
    oship.startup.zdaemon_controller()
```

You should change this to read like this:

```
import oship.at_bldr.py

if __name__ == '__main__':
    oship.startup.CreateAT()
```

May 30, 2008

OSHIP-1.0.1a.x Basic Usage Guide
and OSHIP-1.0.1a.x Developers Tour

remember this is an alpha code implementation

Save and close the initialize script.

Open the file at_bldr.py. In your editor.

OSHIP comes with 10 sample ADL files. They are located in the oship/src/oship/import_adl directory. You can replace/add more ADL files here if you wish or you can edit the 'adlDir variable in at_bldr.py (approx. line 48) to point to a new repository of your existing ADL files. There are a couple of commented out declarations that I use for testing. You can do the same. While you have at_bldr.py open, you **MUST** edit the dbDir variable to point to the correct location on your installation. Most people will just need to change the *'/home/tim/projects'* portion to point to where their oship/var directory is located.

TODO: Create an oship.cfg file and add adlDir and dbDir as parameters.

Once you have accomplished these setup items you will execute (from your sandbox/oship directory) $./initialize (see ADL14parse_errors.log for any reported errors) .
When it finishes you will need to change back to your top-level oship directory (e.g. $HOME/sandbox/oship) remove the oship/var/Data.fs.lock file ($rm var/Dat.fs.lock)

TODO: do this automatically.

Now restart your OSHIP server with ./bin/paster serve debug.ini

You should now be able to set your browser to http://localhost:8080 and see the same generic Zope3 page with the addition of three Site Management folders:

1. AR – the archetype repository
2. DEMOGRAPHICS – the demographics site
3. CLINICAL – the clinical record site (could be complete EHRs)

The DEMOGRAPHICS and CLINICAL folders will be empty. The AR folder should contain a listing of the ADL files that were correctly processed by the at_bldr.py execution. At this point those archetype objects contain mostly parser output in list format for each of the major sections of an archetype:

1. adl_version

OSHIP-1.0.1a.x Basic Usage Guide
and OSHIP-1.0.1a.x Developers Tour

remember this is an alpha code implementation

2. archetype_id
3. concept
4. parent_archetype_id
5. definition
6. ontology
7. invariants
8. revision_history

NOTE: As of the first import to SVN and checking the checkout/install and use instructions if you click on the AR folder it fails to open for some new reason I'll fix this very soon but you can see that it does list the number of items (10 by default).

There are no views yet developed for these archetypes so if you click them you will get an error page. See the Developers Tour section for more information on the project plan.

At this stage of development I am illustrating that healthcare application developers using OSHIP will have access to archetype objects in order to build inter-operable future-proof applications; based on the concepts described at: http://www.openehr.org/home.html whether they are simple disease registries or complete EMR's. The developer will accomplish this using only a suite of Python web application tools such as the z3c libraries. See the tutorial at: http://carduner.net/docs/z3c-tutorial/introduction.html and complete documentation at: http://docs.carduner.net/z3c.form/index.html

OSHIP-1.0.1a.x Developers Tour

I want to thank you for considering development of OSHIP.

This section of documentation will provide a look inside at what has been done to date, a

OSHIP-1.0.1a.x Basic Usage Guide
and OSHIP-1.0.1a.x Developers Tour

remember this is an alpha code implementation

list of TODO items and a look to the future of I envision OSHIP to become.

** A note to healthcare application developers:
If you have heard bad things about Zope2.x (and prior versions) and how difficult it is to use then you need to give yourself some time to get familiar with the Zope Component Architecture (ZCA) (aka. Zope3). About the only thing that Zope3 shares with Zope2 is the first four letters of the name. The ZCA is a complete re-write and comes in the form of a library (set of Python eggs). Instead of plugging in your application to the Zope framework (Zope2) you will be building applications re-using all the great libraries from the Zope Community. The ZCA and the implementation based concepts of openEHR http://www.openehr.org/about/origins.html provide a real opportunity to finally make a marked difference in healthcare application interoperability. While the ZCA comes with form development tools I feel that the z3c.* package tools are much richer in functionality and flexibility. Therefore I have included them with OSHIP.

OSHIP is a Python implementation of the inter-operable, future-proof openEHR specifications: http://www.openehr.org/releases/1.0.1/roadmap.html ;

OSHIP relies heavily on other open source Python components. Especially prominent is the Zope Component Architecture: http://wiki.zope.org/zope3/Zope3Wiki ; if you prefer hardcopy books, Phillipp von Weitershausen's popular "Web Component Development with Zope3" is in its 3rd printing ISBN-13: 978-3540764472 I highly recommend it.

You should also join the OSHIP mailing list and possibly the OSHIP development list at Sourceforge: http://sourceforge.net/mail/?group_id=152993   ;

With that said, I want you to know that I do not feel that I “own” OSHIP. From a somewhat legal stand point I have shared the Copyright with the openEHR Foundation. This insures that OSHIP will always be open and available to anyone to use and enhance. The primary goal of the OSHIP idea is to promote the use of the openEHR specifications to create inter-operable healthcare applications for use in any setting. If successful, this will improve the quality and accessibility of information in all healthcare settings. Whether your goals are philanthropic of you are looking to get involved in healthcare application development and consulting as a business; OSHIP can provide rewarding opportunities. My personal goal is to continue to evangelize the openEHR approach at various conferences and to hold workshops such as the first OSHIP workshop scheduled for late July in Brazil http://www.oshipworkshop.if.uff.br/ The hosts for this workshop

May 30, 2008

OSHIP-1.0.1a.x Basic Usage Guide
and OSHIP-1.0.1a.x Developers Tour

remember this is an alpha code implementation

have gathered an impressive list of attendees and I look forward to the "lessons learned" from this intensive 10 day event.

Now on to the Tour

The OSHIP packages are designed after the Reference Model (RM) and Archetype Model (AM) specifications hierarchy seen here:
http://www.openehr.org/releases/1.0.1/roadmap.html
while integrating the ZCA application server functionalities. The AM and RM trees are found under oship/src/oship/openehr. I have two reasons for this approach. First it is designed to allow for easy updating if and as the specifications change. Secondly, by modeling the class names as closely as possible, users of OSHIP (healthcare application developers) will have a lower learning curve in implementing the openEHR specifications than if we mapped everything directly to ZCA functionality. For example if the data types and data structures were not recreated in name then healthcare application developers would have to not only learn the openEHR specs and the ZCA functionality but the mapping between them. I am therefore very committed to this vision.

However, reality strikes. After creating Interfaces and Implementations of the classes I realized that there are serious circular import issues such as the ones represented in datatypes.interfaces.encapsulated where the interface for IDvMultimedia requires importing datatypes.encapsulated.DvMultimedia because the thumnail attribute is a type DvMultimedia. I solved this by creating a Thumbnail class but this isn't according to the specs. Another unresolved circular import is where IDvText requires a CodePhrase field type from the same openehr.rm.datatypes.text package where DvText is defined and therefore IDvText is implemented(). My temporary solution was to define the DvText.definingCode attribute as a Dict; instead of CodePhrase as it should be. There are other instances of this problem but instead of trying to correct them myself I prefer to have Python/Zope3 experts look at this and discuss the best practices for refactoring before I spend too much time on them. Again, with a goal to adherence to the specs in mind. You may notice in the openehr/rm/datatypes/temp_ci_resolution directory I had attempted to extract each class and each interface into separate files. This approach still did not resolve all of the circular import issues. I look forward to your ideas.

I also know that there are subclass issues to clean up as I become more familiar with the ZCA.

Once the import issues are resolved we can continue work on the archetype builder

May 30, 2008

OSHIP-1.0.1a.x Basic Usage Guide
and OSHIP-1.0.1a.x Developers Tour

remember this is an alpha code implementation

(at_bldr.py) so that full archetype instances can be persisted for reuse in applications. Next will be z3c.widgets so that developers can use these directly in their applications.

This is only a start to a real TODO list. I feel that these are exciting times for healthcare application development/deployment. I look forward to your participation.

You are likely also interested in these mailing lists:
http://sourceforge.net/mail/?group_id=152993
http://www.openehr.org/community/mailinglists.html

It will also be great to see OSHIP developers participate on the openEHR Architecture Review board: http://www.openehr.org/about/arb.html please see http://linuxmednews.com/1211040060/index_html for an announcement.

Sincerely,
Tim Cook <timothywayne.cook@gmail.com>

May 30, 2008