

Yet Another Infrastructure Management Framework (YAIMF)

I know, just what the world needs, YAIMF. For some this whole concept will be alien, for others a workable solution may already exist. The most concise reference I have seen to date is "[Bootstrapping an Infrastructure](#)" by Stephen Traugott. It effectively captured nearly all of the best practices many of us had evolved towards. However, the goals of STiBET were to enable deployment of these types of virtual machines, not just within a given environment, but to create a framework to allow system administrators to manage several distinct environments. The three key deliverables were to abstract out specific content (as far as possible) from the framework, to make the framework easily replicatable (a virtual machine in and of itself), and to make leveraging the content and mechanisms as easy as possible. To this end, the STiBET project was born.

A small team within Hewlett-Packard voluntarily met, doing an exhaustive literature search on the topic, researching each of the tool sets in this space, and forming a development plan of record. Actually development is almost a misnomer as it turns out, since one of the prime directives was to leverage open source or freely available tool sets, only resorting to actual coding when missing integration pieces were discovered. Some guiding principles were established, the core framework pieces were identified, and the project began in earnest. Much like the tools being used, the entire development process was steeped in the open source models of collaborative review and contribution.

Design decisions of record were documented along with how each piece was developed (to aid in later replacement or modification), explaining why a particular tool was chosen and the interface it provided. Classic design phase iterations were undertaken, with rapid build, test, scaffolding cycles, often resulting in a two step forward, one step backward implementation. At it's current incarnation, STiBET is entirely suited to setup for any given environment, and management of the systems can begin. Much consideration was given to the various levels of system administration experience, targeting tool sets and interfaces appropriate to the task and the expertise. Documentation abounds, even including training materials.

So enough background, what is STiBET really ...

Guiding Principles

- Optimize reliability, availability, serviceability, and scalability
- Focus on framework itself, not on content nor policy creation (except as examples)
- Work towards abstraction, making trade offs favoring leverage and portability

Core Deliverables

- Self-refresh capability that also satisfies the bootstrap functionality
- Abstract class membership, actions, and configurations away from tools
- Provide the basic services to manage an environment, and setup a developer API

STiBET

STandard is BETter, or STiBET is BETter ;)



Logo



Mascot

Looking inside STiBET

As expected, two key services needed to manage an environment are a repository for the objects applied, and a scalable way to deliver the relevant objects to the managed nodes. These two services are considered to be within the **STiBET** provider core cell space. **STiBET** accomplishes the repository function via its **Version Control** service. In typical software development fashion, this provider offers a way to manage revisions of any object, in an authenticated fashion so that changes may be tracked. Also, several release streams are supported, “stable”, “testing”, and “unstable” so that managed nodes may subscribe to the level of risk appropriate to their functioning. A wrapper interface for handling change management is supplied to interact with **Version Control** service, but as with all of the **STiBET** components, the native interface is also available and usable.

To handle the replication, **STiBET** provides the **Global Filesystem** service. This has both a server and client component, in that the servers can be deployed to mirror all the **Version Control** service content, and the clients use the same mechanism to replicate local copies of the necessary runtime components from the **Global Filesystem** service providers. An instance of the **Global Filesystem** provider is included in the **Version Control** provider, so that a single machine may be used to manage an environment. As required, however, more **Global Filesystem** providers can be added at any time to spread out the load (and the clients use any that are available).

Both of these **STiBET** providers are built using **STiBET** processes and methodologies, and are, in fact clients of your cell. Each is built using the iterative stages of the [Desktop Management Task Force Application Common Model](#). A simple but elegant gatekeeping function is included to coordinate the content visibility from the **Version Control** provider, **Global Filesystem** provider, and the **STiBET** managed client nodes.

So, how do I get **STiBET** bootstrapped into a new or existing environment ...

Core Services

- **Version Control**
 - repository for any object applied to environment (aka environment “source” tree)
 - includes support of multiple release streams
- **Global Filesystem**
 - scalable, replicated storage system of all **Version Control** components (ie. runtime access of **Version Control** “source”)
 - conscious of release streams

Current Technologies

- **Version Control** -- [CVS](#)
- **Global Filesystem** -- [RSYNC](#)
- service provider setup via stateful, re-entrant [CFENGINE](#) scripts

Bootstrapping STiBET

Okay, so you are now ready to give **STiBET** a try, how is the bootstrap accomplished? It really doesn't matter whether you are setting up a new **STiBET** cell or another managed node (the only difference in this case is the bootstrap host). Basically, a download of a simple shell script is all that is needed. The script, when executed will perform four basic steps. First,, bootstrap checks for necessary prerequisites. Only a very few are not likely to be already present. The bootstrap script will **not**, however, install these for you, since the installation of software is best left to the preferences of the administrators. Second, the script tries to ascertain a valid bootstrap server via of one four conventions – a DNS alias, the same DNS alias with a control domain appended, the host provided as a command line argument, or the last successfully used bootstrap server. Next a refresh of all the core **STiBET** components including itself is completed. Finally, if the newly arrived bootstrap is different, it restarts the entire process , else it links to the normal run-time version of itself. At this point, you have a **STiBET** client that is ready to be managed.

If this system is destined to provide the **Version Control** service for a new **STiBET** cell, then a simple cut-n-paste procedure is used. It involves the download of the remaining **STiBET** content, editing a couple of configuration files to match your environment, and importing this content into your **Version Control**. At this point, you have effectively cut the proverbial umbilical cord to your upstream bootstrap host and can manage as autonomously as you desire. As expected, the content on this cell server should be managed with regard to backups and recovery processes, albeit the backup could be a simple as copying the repository to another machine. This is because the cell server can likely be recreated in much less than an hour, probably even counting data recovery.

At this point you likely now have a fully functioning **STiBET** cell, that is itself managed by the **STiBET** processes. With just a little care, this cell can host multiple administrative domains, can be embedded into an environment with other **STiBET** cells, and can become the bootstrap server to spawn other **STiBET** cells. Congratulations, Dr. Frankenstein, your monster is alive!

So, now how do I manage nodes with **STiBET** ...

Bootstrap

- Check for prerequisites
- Find bootstrap server
- Refresh core run-time components, restart if necessary
- Link to normal run-time execution, exercising entire tool chain

Technologies used (aka prerequisites):

- shell script (with common commands)
- perl + some CPAN modules
- rsync
- cvs (only if using “unstable” branch for managed nodes)
- cfengine

Managing with STiBET

To create an environment of managed nodes (above and beyond the cell server you already have) with **STiBET**, **four basic steps must be accomplished:**

- 1) *First, and perhaps the most important task is planning. Without this, you will need to do much rework later and may paint yourself into a corner. Of course **STiBET** doesn't accentuate nor diminish the importance of this step. What you do need to focus on is how to create classes of operations. That is, categorize actions into common chunks. This can be based on operating platform, functionality, organization, or pretty much any namespace you wish to implement.*
- 2) *Next, you will create an actionable package to implement. This usually includes some type of configuration attributes and a finite set of what steps are necessary to accomplish the goal. These steps will be encoded into directives for the **STiBET-launcher** to parse and perform during **STiBET** runtime. The basic directive types available are: **exec**, **file**, and **include**. The “**exec**” directive type allows an arbitrary executable to be called. The “**file**” directive type allows the conditional replacement of the noted file. And the “**include**” directive type makes it possible to leverage whole sets of directives from another action class. Using tools like [CFENGINE](#) can make these actions sensitive to platform or operating system dependencies if necessary. Based on the underlying file management tool, [LUDE](#), the namespace of the action packages can even be reusable and stackable.*
- 3) *The directives for a given class are then grouped together into a manifest. This is really a simple XML configuration file that makes the directives easy to parse and maintain. Each class has a manifest, and the directives are parsed and executed in the order listed. With the judicious use of the include directive, effective super-classing and sub-classing can bring granular actions.*
- 4) *Finally, the association between managed nodes and the class membership is made in the “**classlist**”. This is another XML configuration file listing the hostname, and the classes it belongs to.*

*Okay, now what, in **STiBET** terms ...*

Classlist

- XML file associating hostname with an ordered list of class membership

Manifest

- XML file listing ordered directive actions for a given class

Directive

- any combination of “**exec**”, “**file**”, or “**include**” primitive types

Example Classes (but totally free-form)

- platform
(eg. Linux, HP-UX, ia64, etc.)
- functionality
(eg. Desktop, WebServer, etc.)
- organization
(eg. FirmwareDeveloper, ChipDesigner, etc.)

Cool STiBET Tricks (simple to complex):

- 1) to rebuild another instance of a managed node, simply install the prerequisites, modify the classlist by copying the desired reference host class associations to the new host's name
(eg. bring a second web server on-line)
- 2) have the ability to have an N+1 spare readily available for any managed node (except for data, all managed nodes are “appliances” now)
(eg. leave a bootstrapped, yet non-class associated node running, then rename this system to the affected node's name)
- 3) utilize clients configured with either the “testing” or “unstable” release streams as early warning test platforms
(eg. configured this way, the system administrator's host can be considered a “canary in the mine shaft”)
- 4) test interactions between classes
(eg. manually invoke **STiBET-launcher** with the `-h <hostname>` to run class-based actions associated with another host)
- 5) utilize the baseclass configuration file “env.cf” to capture all relevant environment configurations
(eg. makes it easy to encapsulate an environment's setup, and then have a clone-and-go approach for a second environment by bootstrapping from the first cell, simple edits to “env.cf” and “classlist” will take advantage of all defined actions for new environment)
- 6) create a new baseclass administrative domain, effectively splitting scope of administration commands (and/or mistakes)
(eg. manage cell servers distinctly from other servers and from typical clients)
(eg. manage two distinct environments from the same **Version Control** service, yet leveraging all the directives, manifests, packages, ...)
- 7) even with no running **Version Control** service, managed environments can continue functioning, albeit in the same snapshot stat with no updates possible, so DRP is moved back to regular office hour timeframes

Current Status (Developer Release 1):

- all components listed are functional, documented, and operational
- design decision and developer API documented,
- framework being used to manage several environment (10s – 100s of nodes), with previous conceptual model managing several large (1000s of nodes) environments
- in process of being reviewed for open-source distribution under GPL license

Futures

- enhanced gatekeeping
- move to cfengine version 2.x support
- documentation of framework disaster recover
- sky is the limit with regard to contributed services (ie. Flesh out the remaining service providers and consumers from the “Bootstrapping” paper)