# Goals of Linux Agent

* Platform agnosticism
  + Run on a variety of platforms *(Ubuntu, RHEL, SUSE, Meego, ESXi, Xen, etc.)*
* Desktop, Server, Embedded
* Dynamic and static linking options
* Maintainability
  + Documentation
  + Installation

# Linux Agent components

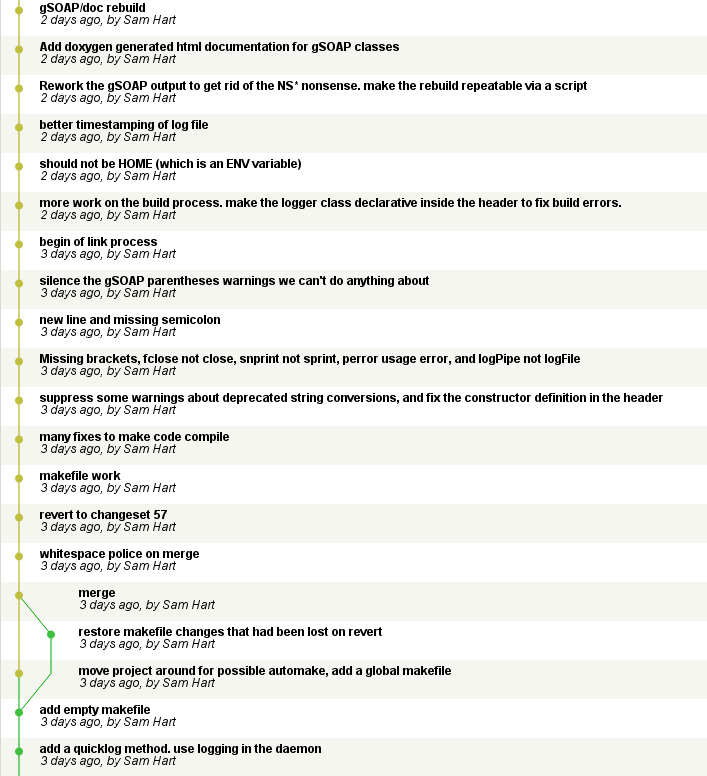
Current plan is to have Linux agent consist of two components:

* A “Steward” which interacts with CCMS, obtaining commands, updating status, etc.
* A “Dispatcher” which the Steward issues commands to, which abstracts the underlying platform specificities.

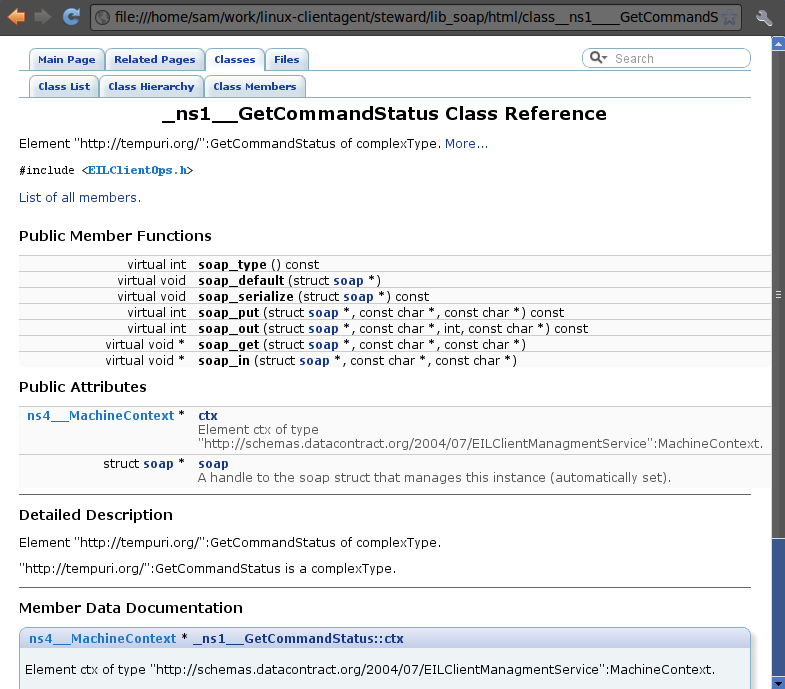
## “Steward” Linux Agent Design

* Previous goal had been to utilize existing C# code in Mono.
  + This lead to complications on platform support re: inhomogeneous Mono versioning across specific Linux distributions and its absence on others.
* Current goal is to utilize lower level gSOAP C/C++ API and write the “Steward” Linux agent in C/C++.
  + The Linux agent will be “thin”, operating only with CCMS and the Dispatcher.
  + Most of the work will be handled by the Dispatcher.
  + gSOAP will allow for maximum compatibility with as many diverse Linux platforms as possible.

## “Dispatcher” Design

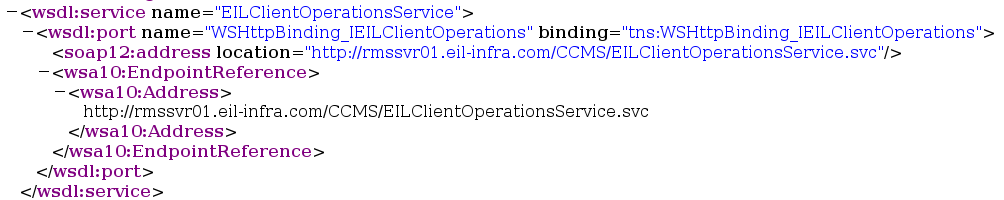
* The “Dispatcher” has not changed in design since the decision to switch from Mono to gSOAP.
* Dispatcher will be a central BASH/ASH script (BASH will be present on all Desktop/Server platforms, ASH on the embedded platforms).
  + Dispatcher will include “suite scripts” tailor made for platform specific differences dependent upon distribution.
  + The goal of the Dispatcher is to abstract any platform specific design choices away from the rest of the Linux agent.
  + The Dispatcher will also be responsible for automatically patching/upgrading the Linux agent (“Steward” and “Dispatcher”).

# What is done

* All current work can be found inside the lab network at a Mercurial repository <http://10.4.0.60/>
  + Mercurial is a Distributed Version Control System, it is used for source control and can also be used for auto-patching on certain clients (Ubuntu, RHEL, SUSE, possibly Meego) in the future. It is cross-platform, open-source, and available for Linux and Windows.
* **Dispatcher**:
  + Framework is mostly complete
    - Installs, upgrades, uninstalls, purges, etc.
  + Scripts are stubbed out
  + Unit testing framework in-place with existing functionality tested
  + Hooks for “Steward” once it is ready
  + Helper tools complete
* **Steward (C/C++ Linux Agent)**:
  + Runs as a secured daemon on Linux
  + gSOAP code has been generated and documented (using Doxygen)
  + System logger in-place, connects with Dispatcher for install/logging information
  + Stubs for gSOAP integration “glue” once it has “firmed”

# Problems/Work remaining

* gSOAP has exposed possible issues in CCMS generated WSDL.
  + WSDL generated contains un-resolvable domains. As an example:



* + - This *may* explain some of the issues we were having getting the C# code working in Mono
    - A work-around in Linux has been to add this un-resolvable domain to /etc/hosts, though I am not certain that is the best long-term solution.
  + The .NET framework masks formatting issues which gSOAP exposes with regard to certain specific command interfaces. This, in and of itself is only a problem in that it complicates the resulting gSOAP API.
    - As an example, “MachineContext”, which is required for each command request from CCMS, winds up being assigned a “Read Only” attribute underneath each command class inside the resulting gSOAP code. E.g.,:  
      \_ns1\_\_GetCommandToExecute->ctx *(Read Only)*
  + These can be overcome, but additional initial work must be done to figure out *why* it is happening and automate the repair of it in the future.
* Presently, the “Steward” can ping and query the possible interactions with CCMS. The work that remains is to resolve the aforementioned MachineContext issue so it can actually receive commands.
* Additional “glue” between the “Steward” and the “Dispatcher” once commands are being received must be completed.
* Additional “Dispatcher” suite scripts (currently only hard requirement is “Reboot”, surely there has to be more than just that?)
* Additional unit tests to be determined.