**SEVENTH FRAMEWORK PROGRAMME**
**Challenge 1**
**Information and Communication Technologies**

**Trusted Architecture for Securely Shared Services**

**Document Type:** Deliverable

**Title:** # Interface Management Report

**Work Package:** WP12

**Deliverable Nr:** D12.1.2 (PM24)

**Dissemination:** PU (public)

**Preparation Date:** December 29, 2009

**Version:** V2.0



## The TAS³ Consortium

| | Beneficiary Name | Country | Short | Role |
|---|---|---|---|---|
| 1 | KU Leuven | BE | KUL | Coordinator |
| 2 | Synergetics NV/SA | BE | SYN | Partner |
| 3 | University of Kent | UK | KENT | Partner |
| 4 | University of Karlsruhe | DE | KARL | Partner |
| 5 | Technische Universiteit Eindhoven | NL | TUE | Partner |
| 6 | CNR/ISTI | IT | CNR | Partner |
| 7 | University of Koblenz-Landau | DE | UNIKOL | Partner |
| 8 | Vrije Universiteit Brussel | BE | VUB | Partner |
| 9 | University of Zaragoza | ES | UNIZAR | Partner |
| 10 | University of Nottingham | UK | NOT | Partner |
| 11 | SAP Research | DE | SAP | S&T Coord. |
| 12 | EIfEL | FR | EIF | Partner |
| 13 | Intalio | UK | INT | Partner |
| 14 | Risaris | IR | RIS | Partner |
| 15 | Kenteq | NL | KETQ | Partner |
| 16 | Oracle | UK | ORACLE | Partner |
| 17 | Custodix | BE | CUS | Partner |
| 18 | Medisoft | NL | MEDI | Partner |
| 19 | Symlabs | PT | SYM | Partner |

## Contributors

| | Name | Organisation |
|---|---|---|
| 1 | Jeroen Hoppenbrouwers | KU Leuven |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |

# Contents

# 1 Executive Summary

This document reports on the results achieved after 24 months of integration work. In order to bootstrap the integration, a temporary re-focus on mutual development and security software training turned out to be necessary. This re-focus has lead to working *integration technology demonstrator* software, which utilises a significant number of all planned architectural modules. Starting in PM 25, the focus will shift back to integration: interface and configuration management, testing, issue reporting, and preliminary user feedback.

Rules and guidelines are in place to deploy a controlled process of component integration according to the TAS3 Quality Manual. Procedures have been developed to allow components to be efficiently verified and validated against the relevant specifications. A software system which partially automates the process and enforces the procedures is available.

## 1.1 Reading Guide

Two other WP12 documents need to be consulted in combination with the current document: D12.3 “End-to-End Testing Report” for the overall testing plan, and D12.4 “TAS$^3$ Integrated System” for the blueprints of the integration demonstrators.

In the current document, Chapters 2, 3, and 4 discuss the full TAS$^3$ Integration Process and associated tools from a management and developer perspective. D12.3 “End-to-End Testing Report” is a natural extension of Chapter 3. Chapter 5 is a support system description for system administrators and (support system) developers. Chapter 6 provides more detail about the available tools for integration support.

The interface management support system will be a combination of the Trac Wiki and the Pool, both described in Chapter 6.

# 2 Introduction

The Integration Work Package (WP12) has the following specific objectives:[1]

- Ensure that all developed software modules, and all work performed by WP1–10, maintain a close fit and integrate with the overall TAS³ project.
- Define, document, implement, and manage interfaces between the core technical modules (i.e., trust layer: WP3-7, application layer: WP8).
- Integrate the trust layer with the employability & eHealth application layers (WP8–9), and test the TAS³ system as a whole on the following:
  - Functionality
  - Performance and manageability
  - Usability and effectiveness
  - Adaptivity

These overall objectives have a sense of sequence: testing can only be done on working software, interfaces can only be managed if there is a software design, and a design can only come from a stable architecture and detailed requirements.

The first 1.5 project years (2008-2009) were characterised by finding out common ground between sometimes very different partners, and establishing a working relationship between the architects and the developing partners. As already described in D12.1.1 (released in December 2008 and revised in June 2009), WP12 had to temporarily re-focus on architecture and design support.

Since June 2009, actual development and lab-level experimental integration has taken place to provide evidence that most planned technologies can actually be integrated. WP12 has re-engineered the existing central server infrastructure to support this integration process, and provided the necessary processes and tools (including developer workshops) for the job and beyond. Having all partners actively use this central infrastructure was a key WP12 goal, which we are happy to report has been reached. The current document functions as a reference guide for this central infrastructure and provides technical documentation for the server farm.

From January 2010 onward, further development will produce formal demonstrators according to the specifications. D12.1 will then change appearance once again, to become a more formal report on the management (process, tools, metrics, results) of the component interface specification repository. All existing tools will be needed for this, and additional tools are expected to be built or acquired when needed. Close cooperation with WP1 “Requirements” is foreseen to assure that delivered software meets the overall and specific requirements which have been defined previously.

[1] See also TAS³ Description of Work, p. 82.

# 3 The Integration Process

For the integration process, a proven and well-honed model has been chosen from the OpenSource development community, the *Debian package life cycle*.[2] Debian has always been well-known for its focus on integration quality and process, and excels in providing normative policy documents and associated toolkits to enforce package management aspects which usually exceed the services offered by other distributions. The Debian package life cycle model divides the process into four steps, which we adapted to the TAS[3] environment:

1. **Upstream**. Developers produce individual components. They are handed over to a component maintainer close to or member of the development team, who prepares the component for submission to the integration repository ("The Pool") and becomes the contact point after submission.

2. **Unstable**. New components in Unstable have been sufficiently documented and have been accepted into the Pool. Fellow developers can fetch them and expose them to each other in lab situations. Issues and defects are logged into a central tracking database. Components are not guaranteed to remain unchanged (hence "unstable"). When components have spent sufficient time in Unstable and still do not exceed a maximum number of reported critical issues, they automatically promote to Testing.

3. **Testing**. Components in Testing have been resilient against developer exposure and are being validated for formal release by rigging them together on a test bed. If critical issues are discovered, the component is demoted back to Unstable and Testing downgrades back to the previous component revision. Project management monitors the status of components in Testing and decides on a subset to be promoted to Stable. Testing is stable enough for thorough testing, but there are no guarantees that nothing will change.

4. **Stable**. Well-defined component sets are collectively released under a collective tag, and offered for external (demonstrator) access. Stable releases are maintained for a predetermined time (by project management) and in principle receive only defect fixes, not feature upgrades. Several Stable releases may exist in parallel under different tags.

Exceptions to the promotion rules can be made at the discretion of project management. Especially clean defect fixes to components in Stable can be expected to quickly move through Unstable and Testing.

With the exception of the Stable category, components will be in a continuous revision cycle: the same component may have more than one version active, though just one at a time in Unstable and one in Testing. As subsequent revisions of the same component come from Upstream, they move through Unstable into Testing and accumulate quality indicators (actually badness indicators). Depending on their quality, components will make more progress.

[2] http://en.wikipedia.org/wiki/Debian#Package_life_cycle

Issues and defects are tracked for each release, where a newer release has to prove to fix previous issues. Up to the Testing position in the process, component maintainers may withdraw components and replace them by newer revisions without consulting peer developers.

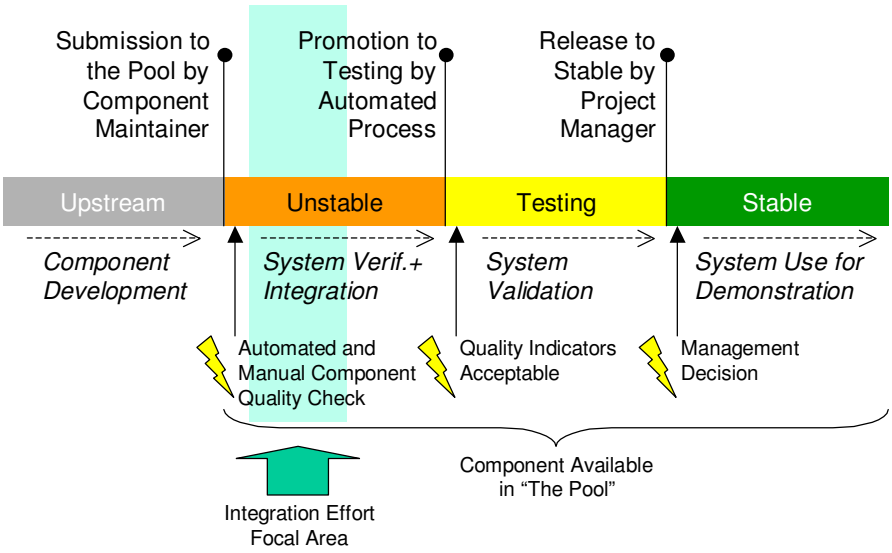


**Figure 1: TAS³ Integration Process**

The TAS³ development and integration process has an unusual focal area around 30% into the process. This reflects the *release early, release often* policy which aims at assuring as soon as possible that components will integrate. Trying to advance the component through the process is the target, not the means. A brand-new component is expected to have significant issues passing this focal area at first. Later revisions should be able to pass this area swiftly and "go to work".

Creating individual components is left to the development partners. Sufficient *development process freedom* must be allowed to honour the fact that TAS3 is a research project. Agile development processes are recommended, not enforced. Unit tests in particular are heavily recommended but code coverage is not enforced by the integration process.

Component maintainers need to *create installation and configuration software in addition to documentation*. An automated process will check out the component from the repository and attempt to assemble a meaningful system from it. This is less a typical build-from-source, and more an atypical install-configure-test-teardown cycle.

A component with good installation instructions that can be successfully deployed, but that has zero or wrong functionality, *is accepted into the Pool without further questions*. It may even leave Unstable for Testing if no issues are raised against it. However it will not pass into Stable as it will not validate.

Components that have been in the unstable phase for long enough without issues are promoted to *stable setups which change only at formally communicated moments*, so that demonstrators have ample opportunity to align their own systems. Stable setups are maintained by the project until Project Management decides to no longer support the setup.

## 3.1 The Four Integration Categories

The TAS[3] development and integration process is divided in four categories. Each component (software module, with a unique identifier and release number, following the pattern T3-PEP-RQ-3.0.1) starts its life cycle in the first category and may promote to further categories in time.

### 3.1.1 Upstream: Individual Component Development

In the strict sense, the Upstream category is *outside* the TAS[3] integration process, as it precedes the moment a component is submitted for integration. For clarity, Upstream is discussed here as well.

In order to get to components that perform properly in isolation (including locally rigged laboratory orchestrations with other components), developers may elect any platform or system or setup they wish. This approach is especially valuable for advanced research prototypes of yet untried solutions. It also readily enables loose coupling of existing modules that are not operationally maintained by their respective developers (in other words, downloaded copies of other software packages that are left unmaintained).

Developing components that work in isolation does in no way imply that the development can take place in isolation as well. For TAS[3] it is essential that very early in the development stage, the developers start communicating to their future "customers", which are fellow developers of adjacent components. There will be few if any cases where an existing interface specification is sufficiently mature to cover all aspects of inter-component communication, so thorough discussions of projected interfaces will be needed. Instead of focussing on the "core" of their component, developers in TAS[3] should focus mostly on the "edge." However it is unrealistic to assume that all neighbour components will be available for immediate interfacing during the development, so a level of independence is granted. As long as there is communication between developers, which WP12 will monitor.

As is common in research environments, developers are encouraged to swap as many components as they wish, especially if two components need each other to perform the desired function. But for the sake of efficiency and engineering freedom, no formal maintenance or service level agreements need to exist between developers. Neither is there any requirement to package software, to keep documentation current, or to use a source code control system to track software changes. We assume that many researchers already use some of these tools anyway, as they are certainly valuable even to the most unorganised developer.

We consider it part of any serious development effort, professional or research, to have proper unit tests available for the component. Stand-alone unit testing tools are commonly available, and many modern software development methods emphasize early unit testing as an essential part of development. We leave it to the developers to decide whether to first write the tests and then the code, or to use any other sequence, as long as eventually there is local test code available to exercise the component. The WP10 partners offered their assistance if partners need advice on how to implement unit tests and how to deploy testing frameworks.

Lastly, in top-down integration there is a need for components that are just empty shells which always return the same result. Even with no real functionality, developers can already install, interface, document, and pre-integrate the component. Focussing on these aspects before any "real" code is being written will allow the developers to get the basics right from the start, and will reveal early interface problems when other developers gain insight in what will be offered. WP12 encourages early releases of all components, which are feature-incomplete, but adhere to all requirements to make it through much of the integration process. It is an illusion to think that any component will integrate the first time, no matter how well the specifications have been written and implemented. Early releases and continuous integration will be a requirement for any success, a vision which is strongly shared with the infrastructure work package, WP8.

### 3.1.2 Unstable: Publishing Components for Integration and Verification

Integration is very much part of development. In TAS$^3$ we cannot expect components to get fully finished in isolated development labs. Integration therefore is not a matter of sending the module out and waiting for the issue list to be returned a month later. The developers need to virtually travel with the module and co-develop together with fellow developers of other modules. Submitting a module for integration *increases* developer's commitment and communication efforts, and also puts his documentation and packaging skills to the test.

As soon as a component leaves the research lab to be integrated, rules come into play. If other developers need to interface to the component, the bare minimum requirements it needs to fulfill are:

1. It needs to be formally submitted, with release notes, a suitable component name such as T3-PEP-RS, and a suitable version identification such as major.minor.patch.

2. It needs to be made available in a known, central place (the pool) for downloading by fellow project developers.

3. It needs to come with comprehensive documentation for fellow developers. A minimum content standard and format will be proposed in close cooperation with the developers.

4. It needs to be packaged in a suitable way beyond plain source code, so that fellow developers can install and upgrade the component with minimal effort, and limited automated sanity checks can be performed at pool submission time.

5. A formal component maintainer (person or very small group) needs to be appointed to function as the contact point for all component issues.

6. The component maintainer assures that third-party software on which the new component depends is also submitted to the Pool. By default the same component manager assumes responsibility for these third-party components, but (s)he may delegate these responsibilities.

7. The component maintainer commits to reasonable response to raised issues with all components for which (s)he has the responsibility. This includes third-party components.

Formal release numbers are required to track software progress, keep the software linked to the proper documentation sets and interfaces, and to define known dependencies on other component versions. The packaging requirement is both to allow fellow developers to quickly and easily exchange versions of the same component for integration, and to test the packaging itself for robustness. Software packaging may be as simple as a Java .jar file or as complex as a Debian package with full dependencies and other metadata included. Which packaging formats are acceptable to the project will need to be discussed with both developers and system administrators later on. Releasing packages does not mean that the source code should not be available to fellow project developers: at this stage, all software needs to be inspectable and shareable within the Consortium (unless specific agreements to the contrary have been made either in the Contract or in the Consortium Agreement).

Please note that this formal release system puts no limit on the actual number of releases. With co-development being the primary target of the integration phase, multiple releases per day are to be expected. This clearly requires a streamlined release and packaging process, reason why we stress the professional skills required of the developers. Pool submission is routine, not in any way associated with the formal delivery of a Dx.y deliverable to the Commission.

The unit tests that may come with the component are a valuable tool at this stage to assure fellow developers that the component has been properly installed and can be expected to react in a predictable fashion to their integration experiments. As with the packaging and implicitly with the documentation, the shipment of unit tests is a quality pre-check to allow the component to progress smoothly to Testing when no deficiencies are found by the fellow developers.

The TAS$^3$ project does not enforce developers at this stage to install their components on a central shared server, because we foresee that many integration efforts will be carried out on a local server which is already available to many partners. However we reject having the component installed *only* on the developer's local server and access via a network being granted to other developers. This invites to take installation and documentation shortcuts and disallows proper installation by *other* developers, which is one of the purposes of

the Unstable pool category. Passing the component around and having it installed by others on several servers is the better option.

Next to the research and development needs, the fellow developers integrating the component need to document their experiences in a centralised system. There is no ban on private communications between developers, but traces of actual integration activities and results need to become available as well. This allows project management to track the progress of individual components out of laboratory research into the project. On top it should enable the original developers of a component to have a well-structured overview of what still needs to be done before their component may be considered mature enough for Testing.

When a component has been behaving well in Unstable for a certain amount of time (to be decided) and has seen sufficient interaction with fellow developers'components, it may be promoted to Testing.

### 3.1.3 Testing: Centrally Deploying Components for Validation

The requirements for a component to be included in "Testing" are that:

1. Must have been in Unstable for the appropriate length of time (the exact duration depends on the "urgency" of the submission).

2. Must not have a greater number of "release-critical" bugs filed against it than the current version in Testing. Release-critical bugs are those bugs which are considered serious enough that they make the component unsuitable for release.

3. All of its dependencies must either be satisfiable by components already in Testing, or be satisfiable by the group of components which are going to be installed at the same time.

4. The operation of installing the component into Testing must not break any components currently in Testing.

The purpose of Testing is to prove that the component has sufficiently matured, both in technical functionality and in environmental adaptation, so that it could feasibly be shipped as a component for application outside the project. This explicitly is not going to happen, but we have to provide evidence to the EC that we actually produced this quality of components.

Dedicated, non-developer, system administrators will use the documentation to fetch, install, configure, unit test, and deploy the component in a well-defined, clean, central Testing system. If any of these steps fail, the component is routinely rejected and considered "not yet delivered" by project management. But such a "failure" is by no means final or a disqualification. A component may already score well on some points (such as packaging, stability, documentation, and interfacing) while it is not yet there on other points (such as features). Such a component will formally fail Testing, but provide a very good indication of where work needs to be done. There is no reason whatsoever to not promote components from Unstable to Testing just to see what the scores are. This will

test not only the component, but also the validation process, and it will produce good progress indicators for project management.

When a component successfully deploys, dedicated testers will expose the new component to a predefined testing environment, which as far as possible consists of other components produced by the project. Some components will be target candidates for the WP10 testing framework. Others will serve roles outside the security architecture and do not need to be WP10-tested, but likely will be WP9-usability tested. In all cases, both the technical quality up to and including deployment, and the choreographical quality in terms of interactions with other components will be checked when assessing the acceptability of the component.

The system administrators and testers will keep a record of known good (accepted) components (names and versions), and compile subset reports that show which orchestrations worked properly. In some circumstances, it may be acceptable to have a component that performs as desired in just a few orchestrations, as it may be a phased development. The bottom limit is that any component must perform in at least one orchestration with at least one other component to pass Testing.

### 3.1.4 Stable: Centrally Deploying Components for Demonstrators

When components have successfully passed Testing and have been formally passed to the EC as project deliverables, they become available (unchanged) to the Demonstration Platform. This is a separate system rigged exactly like the Testing system, and maintained by the same people. Components get promoted from Testing to Stable automatically if they have been sent to the EC. In fact, the EC reviewers may request access to the Demonstration Platform for verification of these components, subject to their technical abilities and availability of testing tools.

On top, the project Demonstrators may access these components from their (experimental or limited production) systems, to assess the applicability of the components in real-life situations. While the Demonstration Platform has no formal Quality of Service restrictions (there is no Service Level Agreement between the TAS$^3$ project and the Demonstrator Partners), a reasonable stability and performance needs to be provided for. Changes to the Demonstration Platform are subject to formal change windows, timely announcements, and preplanned formal demonstrations such as during conferences or project events.

It is very well possible to have more than one Stable environment. If so, clear identifiers for each environment will be selected, and a clear life cycle for each environment must be announced. Resource limitations may dictate a practical maximum number of parallel Stable test bed environments.

# 4 TAS[3] Integration Rules and Recommendations

This chapter lists the formal rules and recommendations for the development, integration, and acceptation process of the TAS[3] project. Note that these rules and recommendations are subject to change. Like other documents, this D12.1 document is under change control, so the relevant people will be kept up to date of changes. The latest revision always will be available at the TAS[3] project management site.

This chapter uses the keywords (MUST, SHOULD, etc.) of RFC2119.[3] All text is normative unless expressly identified as non-normative. Prose and specification have precedence over examples, which in absence of normative text, should be considered RECOMMENDATIONS. Examples as used in the documents are illustrative of the application of the relevant principles contained in the documents and are not statements of principles.

## 4.1 Requirements of Submissions to the Pool

The following requirements need to be met by a component before it is accepted into the Pool (in the Unstable category).

1. Components MUST be free of known *Blocker* defects before leaving the lab. There SHOULD be a tangible penalty for being sloppy in self-testing of produced or fixed code, and in assuring that there is peer testing of the code. This could be for example in the form of a beer credit system at a local pub.
2. Components SHOULD come with a comprehensive self-test (unit test) part which can be run against the component on the same platform as where the component can be installed.
3. Components MUST come with sufficient documentation to allow beta testers (developers outside the producing group) to fully install, configure, (unit)test, and deploy the component.
4. Components MUST have a clear planning (road map) with features to be expected and relevant time frames, so that beta testers can plan for these.
5. Components MUST have a clear responsible person (“Component Maintainer”), not necessarily the developer, who responds timely and sufficiently to issues raised by beta testers.
6. Components MUST be packaged in a suitable format for the whole component life cycle. The package guidelines are to be published separately from this document and include naming and versioning rules.
7. Components SHOULD come with as many automated installation scripts, configuration scripts, (and unit tests) as possible to allow for reproducible results.

[3] S. Bradner, ed.: "Key words for use in RFCs to Indicate Requirement Levels", Harvard University, 1997.

8. All known *Blocker* component defects MUST be fixed before the same component with new features is accepted into Unstable. When on a deadline, judicious few exceptions can be made (and documented).

## 4.2 Requirements of the Pool (system and process)

The system supporting the Pool needs to meet the following requirements.

1. One central defect and issue tracking system MUST be used for all reporting of all issues and defects of all components having been submitted to and accepted into the Pool.

2. An "Issue Editor" SHOULD be regularly reviewing the issue tracking system, to assure that issues and defects have been properly documented so the component maintainers can handle them.

3. All defects MUST have been confirmed fixed by a reviewer, not the developer, before they are cleared from the tracker. It is preferred if the original issue reporter confirms the fix but this is not required.

4. The system MUST use as many automated checks as possible upon component submission to assure that basic requirements are met. This includes execution of self-tests where feasible (by unpacking the component package and attempting a standard installation, test, and teardown).

5. For all components in the Pool, the system MUST offer a direct overview of component status (Unstable, Testing, Stable), origin, roadmap, open issues and defects, and availability including download method.

## 4.3 Requirements of Promotion from Unstable to Testing

For a component to promote from Unstable to Testing, the following requirements hold.

1. Quantitative quality metrics MUST be available for each component (numbers of issues and defects, severity level (critical, cosmetical, ...), time since last issue or defect report, open versus closed versus reopened issues, ...).

2. Guidelines for minimum/maximum limits to quantitative metrics MUST be established by project management.

3. Project commitment MUST be assured to allow developers to spend sufficient resources on meeting the guidelines.

# 5 Central Servers

Although many servers are run for, and maintained by, a dedicated developer or developer group, some central resources are kept running by WP12 for the project as a whole. This section describes the main characteristics of these central servers, the way we manage them, and the technical details of operations. The section is intended to be an up-to-date reference document that can be used offline, in case of emergencies. However, some of its content may be also available online as part of the TAS[3] Wiki, to ease editing by multiple persons.

## 5.1 Hardware

The main central server is a 2U Supermicro machine with 2 x Intel Xeon X5365, Clovertown 3.0 Ghz Quad Core, 8 x 2GB RAM, 5 SATA-II disks of 500 GB each, of which the first four make up a RAID5 unit with a 3Ware controller. It is hosted unmanaged (just space, power, and network) by D-Cube Resource and physically located in a data center near Amsterdam.

The host is named `alpha.tas3.eu` to indicate it is hardware (alpha-juliet name range).

In case of non-critical hardware failure, the RAID driver should log these events in the normal syslog, where it should be picked up by the filters and mailed to root.

### 5.1.1 Console Access

Console access to the machine is possible, but not available all the time. The system administrator from D-Cube Resource[4] can set up Java-based access via HTTPS for a limited time only. So be very careful with this host. Don't run anything on it that is not strictly necessary for its task – especially no user processes. Do not open it up to the network except for the current `sshd` on a nonstandard port.

## 5.2 Xen Hypervisor

In order to offer more than just one host, and be able to split root access rights over dedicated machines instead of sharing everything on one box, we deploy the Xen Hypervisor.[5] This is a well-known virtualisation system that provides near-native performance with easy manageability.

Xen works by running a modified Linux kernel on the hardware, that allows sub-kernels to exist which do not have access to the protected hardware except via controlled channels. Xen calls the hardware the *Xen Host* and the various virtual machines the *Xen Guests*. Alternative, common names are Domain Zero (Dom0) for the host and User Domains (DomU) for the guests.

[4] Jan Dries <jan.dries@dcube-resource.be>

[5] http://www.xen.org/

## 5.3 Operating System on the Hardware (Xen Host, Dom0)

Linux, the Ubuntu 8.04 “Hardy” distribution (Long Term Support until April 2013). The kernel has been patched for Xen using the normal Xen kernel package. In the host boot menu, the unpatched kernel always sits next to the Xen kernel, so in case of significant trouble it is easy to revert and fix the problem. The Xen kernel is boot default.

### 5.3.1 Patched Xen Kernel

A major issue is that the drivers for the 3Ware RAID controller contain a critical bug in the Xen kernel as delivered via the Ubuntu distribution. A new kernel *must* be patched before it is booted, else the system crashes hard on first disk access.

The patch process has been scripted by means of a hook in `/etc/kernel-img.conf` so that in principle a kernel upgrade should be fully automatic. Issues have been observed when the Ubuntu upgrade scripts miss a header package, but the actual kernel upgrade works.

All required material and the instructions for a manual upgrade and patch can be found at `alpha:/data/xen-3ware-patch`. The README file contains a simplified procedure which is as follows.

```
# apt-get install linux-headers-2.6.24-24-xen (USE CORRECT NUMBERS)
# cd /data/xen-3ware-patch/3w-9xxx
# vim Makefile
SRC := /usr/src/linux-headers-2.6.24-24-xen
# make
# cp 3w-9xxx.ko /lib/modules/2.6.24-24-xen/kernel/drivers/scsi
# update-initramfs -k 2.6.24-24-xen -c
```

This should leave you with a bootable Xen kernel. Note that new kernels that do not change version number still need to be patched.

In case the Xen kernel is not patched, the system hangs on the 3ware drivers, aborts with a kernel panic, and subsequently cannot be booted again from disk as the controller has been pushed outside its state space. The system will try to do a network boot instead, fails this, and looks dead. But another (second) reboot will allow a normal boot into Grub (press Esc quickly when Grub appears) followed by the selection of the non-Xen kernel.

It may be worth to install an automatic fallback procedure to a non-Xen kernel in case of a failed boot, but there is only so much time in a day.

Notice that in all cases, a Xen host kernel upgrade should be followed by a Xen guest kernel upgrade on all guests. This may be postponed a bit, but not for too long, for obvious reasons.

### 5.3.2 Console tty1

Xen fiddles with the Linux console device, and tends to replace `tty1` with `xvc0`. This has been manually remedied by editing the `/etc/event.d/tty1` file, and the associated kick of the Upstart event `tty1`.

### 5.3.3 ssh access

In order to keep unwanted people and bots out, Alpha runs the ssh daemon on port 22 for selected IP addresses only, and on a nonstandard port (5045) for the rest of the internet. Further restrictions are put in the `/etc/ssh/sshd_config` file to create a tight security policy: no passwords (neither tunneled nor challenge/response), explicitly allowed users, and no direct root login. The `/etc/sudoers` file is likewise tightened up.

### 5.3.4 Nullmailer

The mail delivery from the Alpha host to the internet is not done directly. We use the official TAS[3] email relay on another machine (kilo) for this, as the hosting company's relay needs TLS and SASL authentication. Such a relay needs significant configuration and to avoid unnecessary activity on the hardware host, all this has been delegated to a virtual machine. On Alpha, the nullmailer package provides for a very simple delivery agent. The drawback is that if the virtual relay host isn't running, Alpha cannot call for help. This situation will be caught by the Nagios monitor which runs outside the Alpha cluster.

### 5.3.5 Logcheck

The standard logcheck package is used to regularly scan all relevant log files and mail the exceptions to root. Additional rules to silence nuisance warnings are available in `/etc/logcheck/ignore.d.server/local`. Do not change configuration files that come with the logcheck distribution package.

### 5.3.6 Firewall

Alpha is not only the hardware host, but also provides all firewall services both for itself and for all its virtual guest machines. This centralises maintenance and avoids firewall mismanagement by administrators of virtual hosts, a situation that likely will occur as individual developers get root access to their remote server box. VMs still can have their own firewall to further restrict access.

The Alpha firewall is based on the standard iptables/netfilter system. It is set up by the `/etc/init.d/firewall` init script, which runs automatically at boot before services are started. The firewall can be started and stopped (i.e., completely removed, not made solid) by the regular Debian `invoke-rc.d firewall [start/stop/restart]` command. You can maintain it via the normal Caspar/SVN mechanism from your config working directory.

Documentation about exactly what is and is not allowed is included in the firewall script. In general, everything outbound is allowed, and only very explicit service requests inbound are allowed. In particular, where possible ssh access to

port 22 is not allowed, and replaced by access to a nonstandard port. This avoids much of the port probing and brute-force password cracking attempts we face daily (and use denyhosts to counter).

The FORWARD rule chain is configured to have the same kind of restrictions per Xen guest. Do not forget to enable access for any new Xen guest, else it will seem dead on the network. Usually a copy and modification of an existing guest in the firewall script will work fine.

The structure of the file is self-documenting. Please just follow the framework. If you want to make sure that you do not lock yourself out (a real risk), uncomment the line in `/etc/cron.d/restart-firewall` that puts up the originally installed firewall, and experiment from the configuration working directory (do not copy, i.e., do not make-install).

### 5.3.7 Network time

Alpha receives network time from the global NTP pool, and keeps all Xen guests updated automatically.

### 5.3.8 Backup

See the dedicated section 5.6 about Backup for more information.

### 5.3.9 Upgrades

The Alpha host follows the regular updates of the Ubuntu Hardy distribution. However, these updates are not automatically downloaded and applied. At least each month, and directly when critical security patches become available, the Alpha host needs to be completely upgraded as follows.

```
# apt-get update
# apt-get dist-upgrade
```

In most cases, there will be no Linux kernel in the upgrade, in which case there should not be any additional activity required. In case there is a new kernel, you *must* apply the kernel patch as described in section 5.3.1. Normally this is automated but you must know what happens and what to do when it does not immediately work due to a missing RAID driver.

## 5.4 Operating System on the Virtual Machines (Xen Guests, DomU)

The system on the guests is in principle the same as that of the host, though this is not strictly necessary. There is no reason to run different kernels on any machine, so these need to be kept in sync by adapting the Xen configuration files of the guests. All the rest of the guest OS is independent of the host, and needs to be kept up to date using the normal `apt-get dist-upgrade` command.

On the Xen Host (alpha), modifications to the standard `xen-create-image` process have been made, to quickly generate a proper system that fits with the rest of the

setup. These involve a few defaults in `/etc/xen-tools/xen-tools.conf`, a patch of the actual `/usr/bin/xen-create-image` script to use the newer tap protocol for disk images instead of file, and additional post-creation scripts for the `/usr/lib/xen-tools/hardy.d` distribution setup.

When on the Xen host a kernel is upgraded, also install the Linux kernel image (which brings in the modules, and these are the required part) on the Xen guests. Note that the kernel is not actually loaded from the guest disk image; it comes from the host disk. This avoids complex peeking into guest disk images at VM boot time.

```
# apt-get install linux-image-2.6.24-26-xen
```

Then on the Xen host update the used kernel in `alpha:/etc/xen/guest-name.cfg` to make sure that at the next VM boot, the right kernel image is picked up. It will never hurt to try this immediately. Missing modules cause logs in the syslog while the system may appear to be working correctly. Assure you review the logs for about 24 hours to catch most obvious problems (logcheck should do this for you).

### 5.4.1 How To Set Up a new Virtual Machine on Alpha Hardware

In order to create a new virtual machine, follow this plan.

The procedure to create a new Xen VM on Alpha is straightforward, as nearly everything has been scripted out. This delivers you a working, near-zero-maintenance Xen guest that runs absolutely nothing except `ssh` on port 22. However, as always, as soon as you start installing services and adding users, you summon the maintenance gods.

1. Create a DNS entry for the new host name, and allocate an IP address for it. Decide on the host name. VMs come from the set (kilo, lima, mike...) .tas3.eu. Decide on the service name (service.tas3.eu). Update the DNS at jouwdomain.be. It may take up to one hour for the update to trickle through, but often it is just a minute.
2. Decide what disk space the new VM should get. It is always possible to increase it later, but better safe than sorry. Especially since we create virtual disks using sparse files, i.e., they don't take up more real disk space than what you use (at the expense of slow performance when carving out new space).
3. As root, do:
   `# xen-create-image --verbose --hostname=mike.tas3.eu -ip=85.17.66.164 -size=40Gb`
   This takes a few minutes, mostly spent in creating the `mkfs.ext3` command and the `debootstrap` stuff. You can find out all defaults by looking at the various `xen-tools` configuration files including the material in `/usr/lib/xen-tools/hardy.d`.
   This process must take minutes. If not, it has failed (but does not tell you). The command will eventually ask for a root (console) password. Write this down!

4. The command will leave you with a Xen config file in `/etc/xen` and a Xen disk image in `/data/xen/domains`. These are ready to run. You may want to put a symlink in `/etc/xen/auto`.
5. Do not forget to enable network access via the Alpha firewall. By default, new VMs have a working network but Alpha blocks *everything*, in and out. Only open the Alpha firewall when you are sure your machine can handle the wild Internet. Do not install a secondary firewall on your new VM, so you can keep all security control outside the local root's hands. See separate Firewall How-To (**Fout! Verwijzingsbron niet gevonden.**).
6. After you start the machine and log in as root (see next section), run `apt-get dist-upgrade` to get the latest and greatest security updates (you need a running network for this, of course). You probably also want to tighten the `/etc/ssh/sshd_config` file. Upgrade all base packages at the new host:
   ```
   # apt-get update
   # apt-get dist-upgrade
   ```
   You may want the following extra packages: nullmailer, logcheck, logcheck-database, denyhosts, caspar, caspar-doc. Usually you also need some Linux kernel package for the modules (see 5.3.1 and 5.4; the actual kernel comes from the Xen Host file system).
7. Install and configure the Caspar/SVN management system. See separate How-To (5.5.1).
8. Configure nullmailer. Use smtp.tas.eu as the mail relay and rewrite nearly all outgoing envelope addresses by the following two files.

   ```
   /etc/nullmailer/adminaddr:
   mike-tas3-root@some.working.address

   /etc/nullmailer/remotes:
   smtp.tas3.eu
   ```

## 5.4.2 Starting a Xen Virtual Machine

You can manually start and stop Xen VMs by the `xm` command, as usual. Creating a VM equates powering up a machine. Do not mistake `xm create` (startup a machine from config files and disk images) for `xen-create-image` (building a machine from scratch and creating the config files, disk images, etc.).

```
# xm list
Name                              ID   Mem VCPUs      State   Time(s)
Domain-0                           0 13438     8     r----- 122584.6
kilo.tas3.eu                       1  1024     1     -b----  21160.0
lima.tas3.eu                       2  1024     1     -b----  14687.4
# xm create mike.tas3.eu.cfg
Using config file "./mike.tas3.eu.cfg".
Started domain mike.tas3.eu
# xm console mike.tas3.eu
    (press Enter like on a serial console)
Ubuntu 8.04 mike.tas3.eu xvc0
mike.tas3.eu login:
```

It is recommended to let VMs shutdown themselves "from the inside" using the

normal shutdown command on the VM, but you can force a shutdown "from the outside" using `xm shutdown`, `xm reboot`, or even `xm destroy` to pull the virtual power plug.

When everything works normally, there is no need to access a VM via the virtual console. Normal `ssh` access and `sudo` upgrades should be routine.

You can make the VM autostart when Alpha (Xen host) starts by adding a symbolic link to the config file in `/etc/xen/auto`.

Normally, when Alpha goes down in a controlled way, it will try to save the running VMs and when Alpha restarts it restores the VMs from the saved files. This takes an enormous amount of time. If you want to prevent it, just stop the VMs manually. However this means the VMs will notice that they reboot; the save/restore option just pauzes them and the machines will not notice except for a clock jump.

## 5.5 Configuration Management

Especially when making changes to multiple systems by multiple people, tiny discrepancies in configuration files can make for very hard to find differences in behaviour. We deploy a configuration management system based on simple tools, the Subversion[6] version control system and Caspar,[7] a collection of Makefile snippets and associated files. This system allows to track and revert all configuration changes made since the virtual machine was imaged from scratch, at the expense of a tiny bit of manual overhead that is gained back very soon.

In a nutshell, a central repository is kept in which all config files of all hosts are stored, and on each host each administrator uses a local copy of the relevant part of the repository to work. A Makefile system is used to push the config files from the local copy to the active place on the system using sudo.

### 5.5.1 How to install and use a Caspar/SVN configuration management system

Two hosts are important. One is the *repository host* which in principle must be different from the *config host*. For practical reasons the repository host should be as independent of the config host as possible. Currently we use a non-project server as repository host for the project servers.

On the config host, two directories are important. One is the *configuration directory*, often in the `/etc` tree, where the actual, live configuration files are placed. The other is the *working directory* where you maintain the working copy which is linked to the repository. In principle, you never access the configuration directory directly. You nearly always use a `Makefile` for this, and therefore a variant of the `make` command.

[6] `http://subversion.tigris.org/`
[7] `http://mdcc.cx/caspar/`

1. Decide on the place where to create the working directory. My default choice is `~/var/hostname.tas3.eu`. I suggest you do this as well. Likely you want to create `~/var` for this.
2. Make sure the repository is created on the repository host. This is something only Jeroen can currently do. He will also ask for your ssh public key.
3. Check out the repository in your working directory:
```
$ cd ~/var
$ svn checkout svn+ssh://hopsvn@hoppie.nl/hostname.tas3.eu
```
4. Only if you are the very first person to use this config repository, create the following file in it and put the given contents:
```
$ cat > ~/var/hostname.tas3.eu/include/install.mk
csp_UHOST = dummy
csp_PUSH = $(csp_sudocp_FUNC)
csp_sudocp_FUNC = sudo $(csp_CP) $(csp_CPFLAGS) $(1) $(3)
include caspar/mk/caspar.mk
```
5. Make sure you have Caspar installed:
```
$ sudo apt-get install caspar
```
6. From here on, a working Caspar environment and Subversion repository is assumed.
7. If you want to modify a live config file (usually in /etc), *copy it to the working directory first*, and check it in as from the distribution. For example:
```
$ cd ~/var/hostname.tas3.eu
$ mkdir etc
$ mkdir etc/ssh
$ cp /etc/ssh/sshd_config etc/ssh
$ svn add etc
$ svn commit -m Distro
```
8. Now edit the config file in the working directory. If you want to test the changes, copy the file over the original using `sudo cp` (or use Caspar to automate this). When testing is complete and you're done, commit the file with a meaningful comment to Subversion.
9. You can use Caspar to copy a working file to the configuration directory:
```
$ make sshd_config-install
```
Caspar has many interesting options to also make your life easier when you need to restart daemons etc., but examples are easier to follow than rules. Here is a Makefile for nearly every type of task:

```
csp_DIR = /etc/postfix
csp_LOAD = generic-load aliases-load postfix
include ../../include/install.mk

generic: generic-install generic-load
generic-load:
<tab>sudo postmap $(csp_DIR)/generic

aliases: aliases-install aliases-load

aliases-load:
<tab>sudo newaliases

postfix:
<tab>sudo postfix reload
```

This file allows you to just type `make` and have everything copied in, then run all Postfix rehash commands and finally reload the complete Postfix config. It is just an example for Postfix, but you can as well create one for Apache2, Nagios, or any other subsystem.

Further reading about this combination of Caspar and Subversion is available at `http://www.hoppie.nl/pub/node/79`.

## 5.6 Backups

We use a comprehensive backup system, that provides incremental daily backups to a remote host, using secure mechanisms. It is monitored for correct operation by our normal Nagios monitor (5.8).

### 5.6.1 Backup Policy

Backups are made to prevent irreplaceable data loss in case of catastrophic disk failure of the machine. They are neither quick-reinstall disk images, nor "oops" archives. In some cases, we maintain several daily snapshots at the backup storage end. For individual file version management, we use CVS and Subversion repositories (that are backed up as repository).

The backup clients (the machines you want backed up) need to individually push their irreplaceable data to the backup server. This includes stopping, dumping, and restarting all their local databases where relevant. Files which come from packaged software are *not* backed up. Re-installing this is easier from packages; the installed package list itself is backed up. While the backup job must run as root, there is no remote root access involved at all, neither incoming nor outgoing.

We use `rsync` for data backup, so only changed files will travel over the network. Connections are made via a secure link using SSH with private keys (but no password due to cron activation), and are heavily shielded against abuse via SSH forced commands (useful when the backup client gets compromised).

Backups are monitored by logging the full script output on the local client, reporting success or failure to local syslog, and reporting success or failure to the TAS$^3$ Nagios monitor. In case a report is not received in time (i.e., 26 hours after the last report), the Nagios system raises a critical alert.

### 5.6.2 Backup Install and Configuration Manual

The quick list of tasks to perform is: copy a few files from a .zip file to `/opt/backup`, generate a ssh key pair, review the config files (they have sane defaults), add the public key to the backup storage host, add the public key to the Nagios host, run a test, and set up a cron job. A routine installation takes about 10-15 minutes, excluding actual backup network time.

Typically, local root access on the backup client is required for all steps. To add the key to the backup storage host and the Nagios host, TAS$^3$ system admin help is needed (currently Jeroen).

On the new backup client, install the backup script and configuration files, and create a buffer directory:

```
$ sudo su -
# cd /opt
# wget http://www.hoppie.nl/src/backup.zip
# unzip backup.zip
# rm backup.zip
# mkdir /var/lib/backup
# chmod go-rwx /var/lib/backup
```

On the new backup client, create a ssh key pair just for the backup job. Replace FQDN with your fully qualified domain name, such as `alpha.tas3.eu`. Do not give the key a password (press Return twice).

```
# cd /root
# mkdir .ssh
# chmod go-rwx .ssh
# cd .ssh
# ssh-keygen -t rsa -b 2048 -C "backup FQDN" -f backup-FQDN
```

While here, add a default config entry that directs ssh later to use nonstandard port 5045 on the Nagios monitor host:

```
# cat >> config
host monitor.tas3.eu
port 5045
^D
```

Now get the `backup-FQDN.pub` file (not the other file) that was created to both the backup storage host and the Nagios monitor host. You probably need sysadmin assistance for this.

While both keys get installed, review the configuration files in `/opt/backup`. Everything is likely okay as it is, but you may obviously tweak stuff if you want to. Leave the shell script `/opt/backup/backup` as it is; this is sometimes upgraded.

When sysadmin reports back that your key has been installed, do a full backup test run. You must do this manually (not from cron) as you need to manually confirm once that the host key fingerprints are correct (by typing `yes` when asked to do so).Currently the host key fingerprints are:

backup1.tas3.eu: 2b:b3:24:03:b4:01:4f:13:06:98:ae:c6:b7:34:d3:bd

monitor.tas3.eu: 38:d2:be:30:04:b1:4e:a1:8b:99:e5:e5:8f:44:50:3e

In order to avoid complexities with private keys from your ssh agent that may take precedence over the keys you just generated, it is recommended to make sure you have used `sudo su -` to become root, and not just `sudo`.

```
# cd /opt/backup
# ./backup
... lots of stuff ...
Done.
```

If this completes without any further warning or error message, congratulations, everything works. If you simply run it again, and again, it should complete without asking you anything.

Now add the backup job to cron and you're finished:

```
# cd /etc/cron.d
# cat > backup
21 4 * * *     root     /opt/backup/backup > /var/log/backup.log
^D
```

You should adapt the backup job start time (04:21 hours in the example) to what suits you best. Typically it should be before office hours. Where the log of the backup job goes is also your own decision, the example usually works fine. Each day, the log gets rewritten, so it does not grow forever.

If your machine uses some configuration management setup such as Caspar and Subversion, do not forget to update it with the new files in `/opt/backup`. and `/etc/cron.d`.

Done! Get yourself a cup of local brew and enjoy the peace of mind.

For the TAS[3] system administrators, the following instructions may come in handy, though typically additional entries can be made by simple copy and modify operations.

Backup storage host: `/home/ttztk1/.ssh/authorized_keys`.

```
command="rsync --server -vlogDtprRz --delete . var/FQDN/latest"
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAA
AQEA038Z/NiW3Q6BiDjr9gw6R80nLNC0ANPA3FmmR97s0kecwZZ6lKKJ2LczV38FwDI
LJ3Ft58ZuEnDjnYrrvRHzVm/FldDv1+Rvffs51GJYypdyffF+OTM5Z2I4E28PD0EouW
G2bhh8bWhd/CqCrnoIPMG0t1rNZFHKSj7YG+U/3HCzceO+ngjs+CSBI3vfWMkmRSn8h
sMrFGkdDnLAyDlieh6xB5iNA+kGH5FuR79qMyH7j9YaC5SbzPhitbN4Zsm92aQ7oBR0
FlKom/8kC5lVjiwBzguIPFxc/OGxloIDNxu493FYbBTkktqGQ6OOgvO5fskRDisKErj
NOwUb0gesvw== backup FQDN
```

Daily snapshot: crontab of user `ttztk1`.

```
33  4 * * *   bin/snapshot var/FQDN/latest
```

Nagios monitor host: `/home/nagstat/.ssh/authorized_keys`.

```
command="bin/nagios_status FQDN Backup $SSH_ORIGINAL_COMMAND" ssh-
rsa AAAAB3NzaC1yc2EAAAABIwAAAQ
AsJssz+eZfm4gCEjoCqwBBsHkEMxlahUteDHKdUt7MYx+SvfFr2wFcKlLh4L1DNONsJ
2besq0PSRdE70vRYkYP8SNVjq05s5wlsH6sP5KfaQ5Rc16/kXyO7iXma4uzj2PwqNGz
VucCPzMRLQ6uyV9RKxDiQaElJWoT901j6YetXQ42PE+/etQnWOt8GXqochQ0esykOO2
bkXhCmg/c7Pejhf4NKbXbX3JNfIwkUa0h0kDxJL9bo8BJ0So9dkkBiow8NXT7kERLyj
wUTKfwuMdVa2XWaC9hjJ5MQdpjIZdpWW2Dl/JaAOhy+jEcA0nD7voFeiTgde6BDON1l
DNSd0DCw== backup FQDN
```

Nagios: `/etc/nagios2/conf.d/FQDN.cfg` (via Caspar and Subversion).

```
define service {
  use                  backup_job
  host_name            FQDN
  servicegroups        internal_services
}
```

## 5.7 Mail Relay

With ISPs getting more aware of the security risks of open mail relays, it has become standard operation practice to put restrictions on SMTP relays. For the TAS[3] cluster in Amsterdam, this means that we need to deploy TLS and SASL authentication to deliver mail at the first relay hop. Although this is not terribly complex, it is a burden for developers that want to quickly set up mail delivery.

To alleviate this burden, TAS3 offers a *project mail relay*. All hosts that are known to the project system administrators are allowed to just drop mail here. It does not matter whether these hosts are in Amsterdam or elsewhere. This allows hosts to deploy a very simple mail transport agent such as the Nullmailer package. The complete configuration for Nullmailer goes into two files:

```
$ cat /etc/nullmailer/adminaddr
hostname-tas3-root@somwhere.com
$ cat /etc/nullmailer/remotes
smtp.tas3.eu
```

The first file contains the email address that is used to forward all mail intended for *local users*. This is handy if you want to avoid piling up mail to root on your machine. Use at your own discretion, or else leave the file out. Mail not aimed at local users will go out normally no matter what you do here. Take care that `~/.forward` files are *not* processed by Nullmailer!

The second file contains the actual address of the TAS[3] SMTP relay and is mandatory.

We run the TAS[3] SMTP relay on the virtual host `kilo.tas3.eu` which is not used for development purposes. It may rarely be offline, in which case Nullmailer will dutifully queue up your mail and try again later. The TAS[3] SMTP host itself deploys Postfix as mail relay.

Again, be aware that the SMTP relay needs to know that mail from your host is allowed. It will not work without telling system administrators that you need relay services.

## 5.8 Server and Service Monitoring (Nagios)

In the TAS[3] Project we use the Nagios[8] system to monitor the various servers (hosts) and services running on these hosts. The principle of a monitor setup requires that the monitor does not run on the resources being monitored. Currently Nagios is set up on an external machine, physically separate and

[8] `http://www.nagios.org/`

dislocated from the main server rack. The system can be reached by `https://monitor.tas3.eu/` using the same credentials as for the Portal.

| Host | Service | Status | Last Check | Duration | Attempt | Status Information |
|---|---|---|---|---|---|---|
| alpha.tas3.eu | Backup | OK | 2009-11-25 05:41:02 | 31d 4h 31m 7s | 1/1 | Completed in 1 seconds |
| | SSH:6045 | OK | 2009-11-25 13:28:28 | 35d 23h 39m 28s | 1/4 | SSH OK - OpenSSH_4.7p1 Debian-8ubuntu1.2 (protocol 2.0) |
| backup1.tas3.eu | SSH | OK | 2009-11-25 13:28:23 | 4d 19h 10m 35s | 1/4 | SSH OK - OpenSSH_4.3 (protocol 2.0) |
| gateway | PING | OK | 2009-11-25 13:24:10 | 0d 15h 34m 44s | 1/4 | PING OK - Packet loss = 0%, RTA = 1.98 ms |
| kilo.tas3.eu | Backup | OK | 2009-11-25 05:44:02 | 31d 4h 30m 44s | 1/1 | Completed in 1 seconds |
| | HTTP | OK | 2009-11-25 13:23:45 | 27d 1h 35m 1s | 1/4 | OK - HTTP/1.1 302 Found - 0.014 second response time |
| | HTTPS | OK | 2009-11-25 13:25:34 | 35d 23h 39m 28s | 1/4 | HTTP OK HTTP/1.1 200 OK - 16807 bytes in 0.041 seconds |
| | SSH | OK | 2009-11-25 13:27:19 | 9d 22h 56m 29s | 1/4 | SSH OK - OpenSSH_4.7p1 Debian-8ubuntu1.2 (protocol 2.0) |
| lima.tas3.eu | Backup | OK | 2009-11-25 05:36:03 | 2d 22h 0m 18s | 1/1 | Completed in 1 seconds |
| | HTTP:8080 | CRITICAL | 2009-11-25 13:23:56 | 95d 21h 58m 35s | 4/4 | Connection refused |
| | HTTPS:8443 | CRITICAL | 2009-11-25 13:26:24 | 82d 2h 37m 19s | 4/4 | Connection refused |
| | SSH | OK | 2009-11-25 13:27:35 | 36d 23h 27m 49s | 1/4 | SSH OK - OpenSSH_4.7p1 Debian-8ubuntu1.2 (protocol 2.0) |
| | idpdemo HTTP | OK | 2009-11-25 13:24:15 | 46d 4h 33m 51s | 1/4 | HTTP OK HTTP/1.1 200 OK - 402 bytes in 0.004 seconds |
| | idpdemo HTTPS | OK | 2009-11-25 13:26:24 | 35d 23h 39m 28s | 1/4 | HTTP OK HTTP/1.1 200 OK - 402 bytes in 0.038 seconds |
| | spdemo HTTP | OK | 2009-11-25 13:27:50 | 36d 23h 27m 49s | 1/4 | HTTP OK HTTP/1.1 200 OK - 402 bytes in 0.005 seconds |
| | spdemo HTTPS | OK | 2009-11-25 13:24:24 | 35d 23h 39m 28s | 1/4 | HTTP OK HTTP/1.1 200 OK - 402 bytes in 0.038 seconds |
| localhost | Backup | OK | 2009-11-25 07:32:57 | 6d 22h 3m 28s | 1/1 | Completed in 55 seconds |
| | Current Load | OK | 2009-11-25 13:26:24 | 34d 3h 51m 19s | 1/4 | OK - load average: 0.00, 0.02, 0.00 |
| | Current Users | OK | 2009-11-25 13:28:02 | 60d 0h 51m 34s | 1/4 | USERS OK - 1 users currently logged in |
| | Disk Space | OK | 2009-11-25 13:24:39 | 35d 23h 39m 28s | 1/4 | DISK OK - free space: / 2702 MB (28% inode=93%): /lib/init/rw |
| | Total Processes | OK | 2009-11-25 13:26:27 | 59d 23h 45m 20s | 1/4 | PROCS OK: 47 processes |
| mike.tas3.eu | Backup | OK | 2009-11-25 05:48:03 | 31d 4h 26m 23s | 1/1 | Completed in 1 seconds |
| | HTTP | OK | 2009-11-25 13:28:15 | 36d 23h 24m 51s | 1/4 | HTTP OK HTTP/1.1 200 OK - 1346 bytes in 0.005 seconds |
| | HTTPS | OK | 2009-11-25 13:24:51 | 35d 23h 39m 28s | 1/4 | OK - HTTP/1.1 302 Found - 0.037 second response time |
| | SSH | OK | 2009-11-25 13:26:46 | 36d 23h 24m 51s | 1/4 | SSH OK - OpenSSH_4.7p1 Debian-8ubuntu1.2 (protocol 2.0) |
| tassservices.uni-koblenz.de | HTTP | OK | 2009-11-25 13:28:31 | 6d 1h 51m 34s | 1/4 | HTTP OK HTTP/1.1 200 OK - 8369 bytes in 0.034 seconds |
| www.tas3.eu | HTTP | OK | 2009-11-25 13:25:15 | 0d 6h 18m 39s | 1/4 | HTTP OK HTTP/1.1 200 OK - 1808 bytes in 0.058 seconds |

**Figure 2: Nagios Service Overview**

Basic Nagios configuration consists of *resources* (contacts, contact groups, hosts, host groups, etc.) that are combined to produce efficient specifications of "objects" to be monitored. Some resource specifications are not directly specifying an object to monitor, but just provide a *template* to provide defaults for real monitor objects. Templates do not need to remain "virtual objects" but usually they are. "Virtual" objects have the `register=0` parameter; "real" objects leave this parameter out (default `register=1`).

The distinction between resource definitions and resource *group* definitions is mostly a display decision. All resources are shown in a flat list, while the group lists scatter themselves over the screen. Groups therefore can be quickly assessed for responsibility and for severity. There is little advantage in creating groups by type if this does not split either responsibility or severity.

Whether to see a virtual Xen machine (guest) as a host or as a service is a matter of interpretation. Usually the guest itself runs services. Since we do not use host and service *dependencies* but only host parent-child relationships, it becomes logical to see Xen guests as children of Xen hosts, and to not make any difference in the services running on hosts and guests. Another possible rule is that the same host or service should not appear in more than one hostgroup or servicegroup.

### 5.8.1 Host Definitions

A host definition is used to define a physical server, workstation, device, etc. that resides on the network. We also include logical servers (Xen guests) in this group, as they function exactly like physical servers from Nagios' point of view. The relationships between hosts ("parents") take both the network topology and the Xen host-guests relationships into account, so that a Xen host that is down will mask out all notifications from the dependent Xen guests. Nagios host dependencies are not (yet) used.

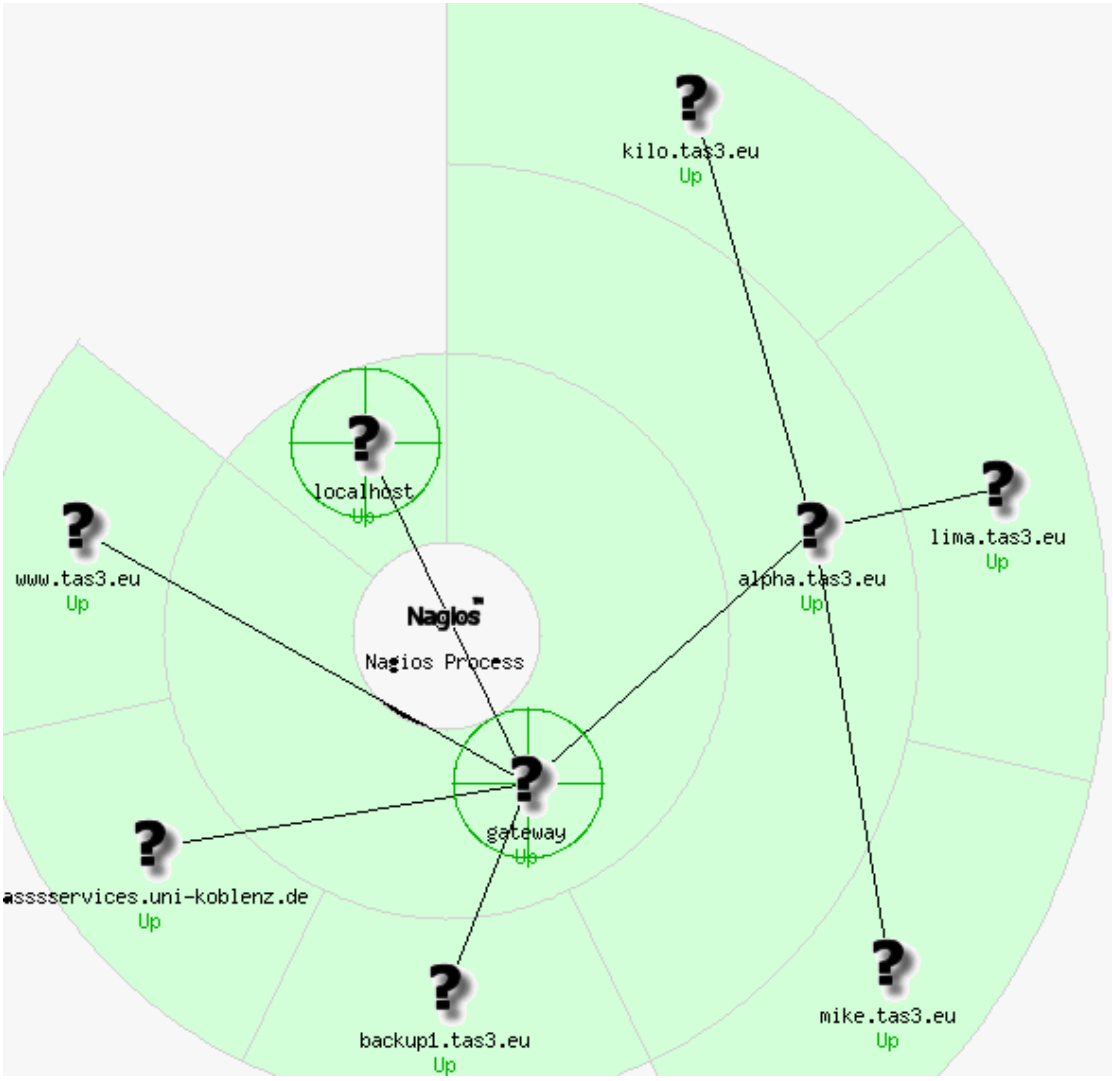


**Figure 3: Nagios Relationship Map between Hosts**

In TAS[3], we consider all the following object types a *host*:

- Physical servers that provide services.
- Virtual servers that provide services.
- Network devices that do not provide separate services (such as routers).

For each of these object types, we have a host template that itself is not an object.

### 5.8.2 Host Group Definitions

Nagios offers a way to display all hosts in groups. Given the structure of the TAS[3] project, we create these host groups based on *hosting provider*. This indicates responsibility in some way, though a hosting provider is not necessarily the responsible party for everything associated with a host. However, this grouping may offer quick diagnosis of network and other provider-related issues. Currently, the following host groups are defined:

- D-Cube Resource
- Nottingham Uni
- VirtualConcepts
- EIfEL
- Koblenz University

More groups will be added when additional hosting providers become engaged in the project.

Hosts are assigned to host groups; we do not list group members in the hostgroup definition.

### 5.8.3 Service Definitions

Services run on hosts. Basically anything can be a service, ranging from free disk space via HTTPS certificate expiration dates to actual SOAP services. Various dedicated service checkers are deployed by Nagios to check for the status of each defined service.

The Nagios developers and community provide for many plugin status checkers, yet for TAS[3] we will need to develop a few more to properly check the security features. Good SOAP checkers are available which may be a proper base for native developments.

Services that run on a host are defined in the single file which describes the host. This allows for a clear single spot to look for all definitions.

### 5.8.4 Service Group Definitions

Service groups are a display feature, and we use it to group services by impact.

- Public services (such as www.tas3.eu).
- Demonstrator services.
- Project-wide internal services (such as portal.tas3.eu).
- Development services.

This list does not imply a hierarchy or pecking order of impact. Whether public services are more or less important than development services is a matter of taste. However, these categories do group services of comparable impact together, possibly facilitating decisions if needed.

Services are assigned to service groups; we do not list group members in the servicegroup definition.

### 5.8.5 Nagios Notifications and Reports

As soon as some relevant event occurs, Nagios sends email messages to the relevant people. Practical groups of people have been created that prevent needless spamming around. For example, all developers get a message as soon as a developer service goes down, because this may influence local software as well. But only project management people and one system administrator get notified of outages of the TAS³ public web server.

| Time | Alert Type | Host | Service | State | State Type | Information |
|---|---|---|---|---|---|---|
| 2009-11-25 07:10:28 | Service Alert | www.tas3.eu | HTTP | OK | HARD | HTTP OK HTTP/1.1 200 OK - 1808 bytes in 0.410 seconds |
| 2009-11-25 07:10:28 | Host Alert | www.tas3.eu | N/A | UP | HARD | PING OK - Packet loss = 0%, RTA = 13.02 ms |
| 2009-11-25 06:41:24 | Service Alert | www.tas3.eu | HTTP | CRITICAL | HARD | CRITICAL - Socket timeout after 10 seconds |
| 2009-11-25 06:41:24 | Host Alert | www.tas3.eu | N/A | DOWN | HARD | CRITICAL - Plugin timed out after 10 seconds |
| 2009-11-25 06:41:14 | Host Alert | www.tas3.eu | N/A | DOWN | SOFT | CRITICAL - Plugin timed out after 10 seconds |
| 2009-11-25 06:41:04 | Host Alert | www.tas3.eu | N/A | DOWN | SOFT | CRITICAL - Plugin timed out after 10 seconds |
| 2009-11-25 06:40:54 | Host Alert | www.tas3.eu | N/A | DOWN | SOFT | CRITICAL - Plugin timed out after 10 seconds |
| 2009-11-25 06:40:44 | Host Alert | www.tas3.eu | N/A | DOWN | SOFT | CRITICAL - Plugin timed out after 10 seconds |
| 2009-11-24 21:54:24 | Service Alert | gateway | PING | OK | SOFT | PING OK - Packet loss = 0%, RTA = 0.32 ms |
| 2009-11-24 21:53:24 | Service Alert | gateway | PING | WARNING | SOFT | PING WARNING - Packet loss = 0%, RTA = 149.57 ms |
| 2009-11-24 00:18:24 | Service Alert | gateway | PING | OK | SOFT | PING OK - Packet loss = 0%, RTA = 0.36 ms |
| 2009-11-24 00:17:24 | Service Alert | gateway | PING | WARNING | SOFT | PING WARNING - Packet loss = 0%, RTA = 100.19 ms |
| 2009-11-23 16:07:14 | Service Alert | gateway | PING | OK | SOFT | PING OK - Packet loss = 0%, RTA = 0.34 ms |
| 2009-11-23 16:06:24 | Service Alert | gateway | PING | WARNING | SOFT | PING WARNING - Packet loss = 0%, RTA = 126.76 ms |
| 2009-11-23 11:46:14 | Service Alert | gateway | PING | OK | SOFT | PING OK - Packet loss = 0%, RTA = 0.32 ms |
| 2009-11-23 11:45:24 | Service Alert | gateway | PING | WARNING | SOFT | PING WARNING - Packet loss = 0%, RTA = 125.95 ms |
| 2009-11-23 06:19:14 | Host Alert | lima.tas3.eu | N/A | UP | SOFT | PING OK - Packet loss = 0%, RTA = 54.51 ms |

**Figure 4: Nagios Notifications**

Nagios keeps a database of notifications, and can compile historical overviews and statistical reports from these. Although not so much required in the development phase, these statistics may become important when compiling key performance indicators and other figures for demonstrator trust networks.

## 5.9 TAS³ Certificate Authority

The amount of technically required X.509 certificates for TAS³ is sufficient to found our own Certificate Authority. Since the technical experience inside the project is high, we can save the expense of purchasing a Root Certificate issued by one of the world's well-known commercial authorities. Installing the TAS³ Root Certificate will be routine for all partners, both developers and demonstrators. The exceptions are publicly accessible web sites, such as http://www.tas3.eu/, and the Portal web site https://portal.tas3.eu/ that is used also by non-technical people who do not generally understand X.509 and are surprised by "insecure web sites" in a security project.

Several files are intended for semi-public distribution and can be downloaded directly from `https://portal.tas3.eu/x509/` (the TAS³ CA distribution site). This resource is protected by an externally purchased certificate, so a reasonable guarantee that people fetch the real TAS³ root CA certificate is available. The MIME types of the files have been selected to allow most web browsers to

immediately import and process the offered files. Sites which use TAS[3] CA-signed certificates should alert their users to the CA distribution site; an example of a site that does this is `http://training.tas3.eu/` (the HTTPS equivalent does not do this).

### 5.9.1 Standard Operating Procedures

Although the TAS[3] CA technically operates as any state-of-the-art X.509 CA, we do not adhere to the very strict security rules usually associated with a CA. A security breach on the CA will not influence the research and demonstration parts of TAS[3] at this moment in time.[9] For reasons of convenience and operation, the TAS[3] CA "lives" on one of our networked hosts instead of on a heavily guarded non-networked machine. However, few people in the project have access to this host, and it is fully shielded by regular access restrictions such as public key ssh (only), firewall, and other standard security mechanisms deployed on bastion hosts. We focus on the organisational and operational aspects of the TAS[3] CA in this document.

The TAS[3] CA uses OpenSSL[10] software to create certificates. The software package is installed on the Alpha host and currently accessible in the `~jeroen` account only. A dedicated `~tas3ca` account may be created (and files moved) if this gives benefits in the future. Root access is explicitly not needed, and therefore the default Ubuntu location of the OpenSSL package including `/etc/ssl/` is disregarded. If the same host would ever need a signed certificate from the CA, these files could still be useful to generate the certificate signing request (CSR) as per How-To 5.9.2.

Everything required to operate the CA, including the configuration file(s), is kept in `~jeroen/tas3ca`. Currently there are no special provisions to keep this content truly secret, such as by encryption; the system backup will dutifully copy it elsewhere in plaintext. The private CA key is protected by a passphrase.

The main interface to the TAS[3] CA is a `Makefile`. It can sign new certificates (via a CSR) and maintain certificate revocation lists (CRL) which are needed in TAS[3] for demonstration reasons. For the same demonstration reason, issuing of short-lived certificates that expire in minutes will be routine.

For the TAS[3] CA, the following structure was set up according to common OpenSSL guidelines[11].

[9] This will change as soon as any form of live data appears in applications.
[10] `http://www.openssl.org/`
[11] `http://sial.org/howto/openssl/ca/`

```
alpha:
/home/jeroen/tas3ca
/home/jeroen/tas3ca/serial
/home/jeroen/tas3ca/openssl.cnf
/home/jeroen/tas3ca/tas3ca-cert.pem
/home/jeroen/tas3ca/newcerts
/home/jeroen/tas3ca/index
/home/jeroen/tas3ca/crl
/home/jeroen/tas3ca/private
/home/jeroen/tas3ca/private/tas3ca-key.pem
/home/jeroen/tas3ca/Makefile
```

The actual structure was generated starting from just an adapted `openssl.cnf` file and associated `Makefile` from the OpenSSL guideline mentioned above. The adaptations mainly concern the change of the default `ca` into `tas3ca` and putting the working tree in `~jeroen/tas3ca`, plus the associated TAS[3] name and address data (the `stateOrProvinceName` must be filled in, else you get a misleading error message). Also, since we use OpenSSL 0.9.8g, the RSA key size was forced to 2048 bits as recommended. Lastly, we want a password on the private TAS3CA key, and therefore removed the `-nodes` argument from the `Makefile`.

```
jeroen@alpha:~/tas3ca$ make init
# NOTE use "-newkey rsa:2048" if running OpenSSL 0.9.8a or higher
Generating a 2048 bit RSA private key
...+++
.....................+++
writing new private key to './private/tas3ca-key.pem'
Enter PEM pass phrase: *******
Verifying - Enter PEM pass phrase: *******
-----
```

The result of this operation is a complete tree including the public and private TAS3CA certificates/keys. As this is in principle a one-time operation, further knowledge of the setup of the CA is not required.

To issue X.509 certificates, in other words to honour certificate signing requests (CSR), please see the CSR How-To (5.9.3).

## 5.9.2 How to make a Certificate Signing Request (CSR)

The TAS[3] Certificate Authority has been created (5.9) to routinely issue signed certificates for internal TAS[3] SSL/TLS applications. If you really need a certificate signed by a well-known CA that has a root certificate available in mostly every web browser, you need to purchase a certificate from one of the commercial providers such as VeriSign.

Normally, X.509 certificates are bound to a FQDN (machine). In such a case, it pays off to set up one system-wide configuration file so that subsequent certificate signing requests can be easily performed using the same data. In typical Ubuntu/Debian distributions, the files are in `/etc/ssl/`. You can of course copy the files to another directory, as long as it is secure, but then you must make sure that `openssl` can find this new file. For the TAS[3] project central servers, we use data as follows, but your own data may of course deviate where appropriate.

```
$ egrep '_default.*=' /etc/ssl/openssl.cnf
countryName_default = BE
stateOrProvinceName_default = Vlaams-Brabant
localityName_default = Leuven
0.organizationName_default = tas3.eu
```

Especially the `commonName` usually needs to match your machine's FQDN.

Good background reading can be found at sial.org.[12]

In order to generate your secret host key, perform:

```
$ openssl genrsa -out host.key 1024
$ ls -l
-rw------- 1 user group  887 2009-08-20 15:53 host.key
```

with 'host' typically replaced with your host name or FQDN. You do not need to be root to do this, but eventually you *may* want to copy the key and certificates to areas for which you need root access. Also, in `/etc/ssl/` you usually find a bogus host key and certificate from 'Snakeoil' that are technically correct, but of course not good for anything except basic testing. You need to generate a host key just once, and maintain good security on it (make it not publicly readable).

Now you create a Certificate Signing Request (CSR) from your host key, using information from the system-wide `openssl.conf` file as default. In this example, only default information was used (repeated return key presses):

```
$ openssl req -new -nodes -key host.key -out host.csr
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [BE]:
State or Province Name (full name) [Vlaams-Brabant]:
Locality Name (eg, city) [Leuven]:
Organization Name (eg, company) [tas3.eu]:
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) [kilo.tas3.eu]:
Email Address [your@email.address]:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
$ ls -l
-rw-rw-r-- 1 user group  692 2009-08-20 16:00 host.csr
-rw------- 1 user group  887 2009-08-20 15:53 host.key
```

The important bit is that by convention, **the Common Name must match the**

[12] http://sial.org/howto/openssl/csr/

**FQDN for which you want a certificate**. And usually you do not want a password, else you would always be asked to give it when your services start up.

You now have your Certificate Signing Request file, `host.csr`, which you need to transfer to the TAS³ CA. Please contact `jeroen.hoppenbrouwers@esat.kuleuven.be` for information on how to do this securely. He will return a server certificate to you, usually in the form of a `host.cert` file.

### 5.9.3 How to honour a Certificate Signing Request

This How-To is only of interest to the TAS³ Certificate Authority (currently Jeroen).

You receive a CSR from a TAS³ partner, whose identity should be confirmed by any practical means such as a phone call. Save the certificate request file(s) in the `tas3ca` directory, usually renaming it to `f.q.d.n.csr`. Then execute:

```
$ make sign
Using configuration from openssl.cnf
Enter pass phrase for /home/jeroen/tas3ca/private/tas3ca-key.pem:
*******
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 1 (0x1)
        Validity
            Not Before: Aug 20 14:21:26 2009 GMT
            Not After : Aug 20 14:21:26 2010 GMT
.....
Certificate is to be certified until Aug 20 14:21:26 2010 GMT (365
days)
Write out database with 1 new entries
Data Base Updated
$ ls -l
-rw-rw-r-- 1 jeroen jeroen 4225 2009-08-20 16:21 kilo.tas3.eu.cert
```

All `*.csr` file(s) have now been replaced by corresponding `*.cert` files. On top, copies of these certificates are in the `newcerts` directory but renamed to the CA serial number. Since all the info inside the certificate is static and the file name does not add anything, a simple `grep` will produce all certificates that match a host, so there is no further filing needed. After getting the new certificate(s) to the requester, the `*.cert` file(s) may be deleted.

If, when signing a new request for a host a previous certificate was signed for, OpenSSL fails with an error similar to the following, either set the `unique_subject = no` option in `openssl.cnf`, or first revoke the old certificate.

```
failed to update database
TXT_DB error number 2
```

To check for equality of certificates, beyond a binary diff, do:

```
$ openssl x509 -noout -fingerprint < mail.example.org.cert
MD5 Fingerprint=1C:74:0F:3B:43:7C:F6:82:9F:1B:FF:C0:DA:37:E4:77
$ openssl x509 -noout -fingerprint < newcerts/01.pem
MD5 Fingerprint=1C:74:0F:3B:43:7C:F6:82:9F:1B:FF:C0:DA:37:E4:77
```

To revoke a certificate, do:

```
$ make revoke cert=newcerts/06.pem
```

and publish the resulting Certificate Revokation List at https://portal.tas3.eu/x509/tas3ca-crl.pem (which must also be done at monthly intervals to keep the file from expiring):

```
$ make gencrl
```

This last command even copies the file over to the target web site.

# 6 Integration Tools and Systems

Internal to the project, WP12 has produced several key resources to enable, monitor, and audit the integration process and project progress in general. Most of these resources have been deployed on the Alpha server cluster hosted in Amsterdam and they are 100% actively managed by TAS³ WP12 staff. Exceptions are:

- The backup server, which is hosted and managed by Nottingham University.
- The mailing list server, which is hosted and managed by KU Leuven.
- The Nagios monitor server, which is privately owned and managed by a key WP12 staff member (a monitor must be physically dislocated from the resource it monitors).

In this chapter, we will briefly discuss the resources available to all project beneficiaries.

## 6.1 The TAS³ Project Portal

The main internal project resource is commonly referred to as "the Portal". The Portal itself has very few actual resources, but it functions as a one-stop-shop for access to all available resources. The most visible part of the Portal is the TAS³ Portal Home page which aims to be a very compressed access point.

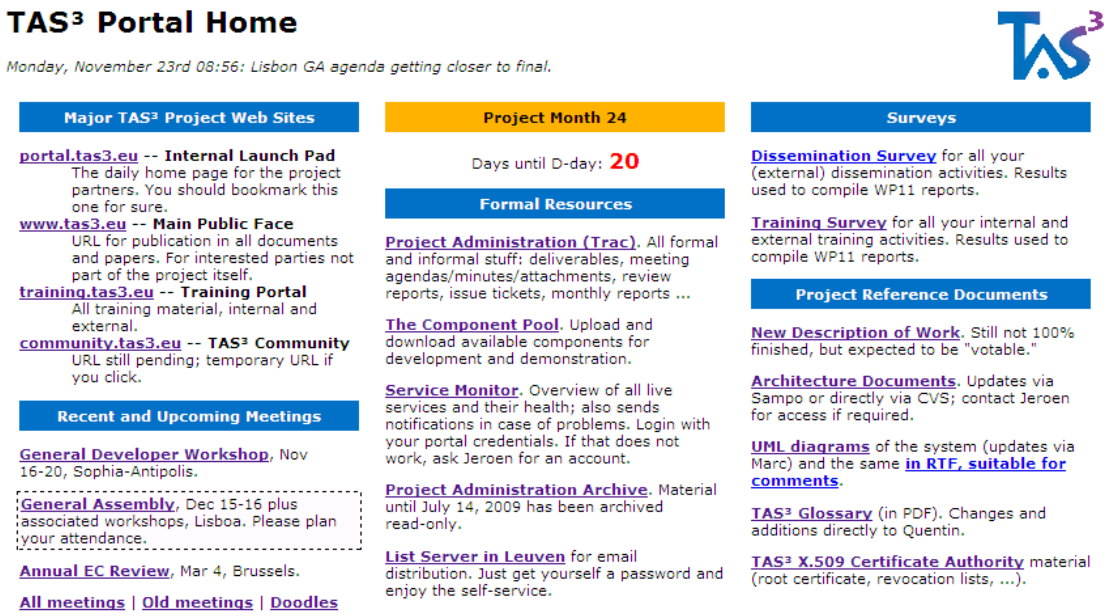


**Figure 5: TAS³ Portal Home Page**

Because the Portal was not meant to be an information resource for people unfamiliar with the TAS³ project, a high information density per square screen centimeter was considered more important than a clear division in target group

sections or other consumer-oriented features. The page scales itself in multiple columns depending on the size of the display used to view it.

A small maintenance effort is spent on keeping the Portal Home Page relevant when project structure changes, or when significant events happen or resources are changed. A list of meeting planners is kept on the same page (but below the part depicted in Figure 5) and special sections appear and disappear when needed. An internal news letter is appended below the meeting planner section.

Project management recommends all participants to use the Portal Home Page as the only bookmarked project reference, so that everybody visits the home page at least a few times per week.

On the same machine as the Portal, several small subsystems have been constructed by WP12 to facilitate day to day operations:

- A full static copy of an older project management and documentation tool that has been decommissioned in July 2009.
- An automatic (hourly) checkout copy of several CVS and Subversion repositories, especially the generated PDF dump directories.
- The TAS[3] Certificate Authority Distribution Site, see Section 5.9. This is an automatically maintained file drop, linked to the generator scripts for the CA.

The TAS[3] Project Wiki and the TAS[3] Component Pool are technically hosted on the same machine as the Portal, and in day to day language often are called “the Portal” as well. However, their specific function and software warrant separate descriptions.

A shared features of most Portal subsystems is a layout library and single sign-on. It is planned to replace the current single sign-on with a ZXID-based SAML2 SSO solution which is part of the current TAS[3]

## 6.2 The TAS[3] Wiki and Issue Tracker

To administrate all issues and provide a functional documentation distribution site, as well as to have a practical means to publish organisation material such as meeting agendas, minutes, and beneficiary addresses and contact information, a combination between an issue tracker and a Wiki was set up. The project decided on Trac[13] as the software, mainly because of its small dependencies (nearly nothing, especially no database server, and no large Java deployment platform). Secondary reasons for Trac were its minimalist philosphy, easily adaptable workflows, and suitable authentication and authorisation models.

### 6.2.1 Trac Installation Notes

Trac is part of the TAS[3] Portal. The installation of Trac is standard, with a few choices that warrant some notes.

[13] http://trac.edgewall.org/

The Ubuntu Hardy distribution comes with a Trac package, but this is the 0.10 version and we want to have at least 0.11 for several practical reasons. So the package was installed manually and not as part of the package manager.

With the way Trac uses the Python installation system, installation is relatively straightforward but not standard. Mostly the instructions on the Trac wiki were followed[14] without the Subversion integration (nobody uses our central Subversion server) and without Bitten. Trac was installed as proposed in `/var/lib/trac` (with a lot of chmod to www-data) and connected to Apache2 as published, root URL `/trac`.

The Trac Authentication Store was changed to use the `trac.htpasswd` file which is shared with the Portal for convenience (poor man's SSO). The `trac.ini` file was updated with several configuration items that cannot be done via the GUI. But since the GUI touches this file often, the file was not added to the system config Subversion. Instead, it is simply backed up with all other files in the Trac directory. We use SQLite which can be backed up straight from the file system.

Two Python plugins in egg format were added: AccountManager and graphvizplugin. The documentation for these is easily found via any search engine. The eggs were built in the `~jeroen/python/trac-plugins` directory.

A custom Trac ticket workflow was built (see 6.2.4) in the central Trac config file as usual.

### 6.2.2 The Wiki Home Page

The Wiki has been set up largely with project administration sectioning, so that the relevant documents can be found using a straightforward selection tree. The root of the Wiki tree (the front page) has a one-line main menu which gives direct access to Reports, Meetings, Deliverables, Work Packages, People, Components, Use Cases, Reviews, Contracts, and Frequently Asked Questions. Further details have been left out, as only people familiar with the project will use this Wiki.

The rest of the Wiki front page is a personalised query of the ticket database, so that at every Wiki access, a user's tasks are made visible. We divided the ticket queries into: My Task List, I Am Waiting For..., and I Am Following... These sections are created by ticket queries for non-closed tickets on owner, reporter, and CC: fields, respectively.

Discussions on specific topics are also done via tickets, as Trac offers nice ways to organise replies to tickets as some kind of Forum. We created a dedicated Request For Comments (RFC) section of tickets to this discussion feature.

A complete list of all open tickets, organised by priority, completes the Wiki home page.

Although there is some minimal overlap between the Portal Home Page and the Wiki Home Page, the minimal menu on the Wiki Home Page is more handy than

[14] `http://trac.edgewall.org/wiki/0.11/TracOnUbuntu`

authoritative, and the lists of tickets clearly indicate that this is not the Portal. A colour difference takes away all other confusion.

### 6.2.3 Components in the Wiki

Next to the Components Pool (Section 6.3) itself, where actual component packages can be uploaded and downloaded, the Wiki has a page dedicated to each component as well.

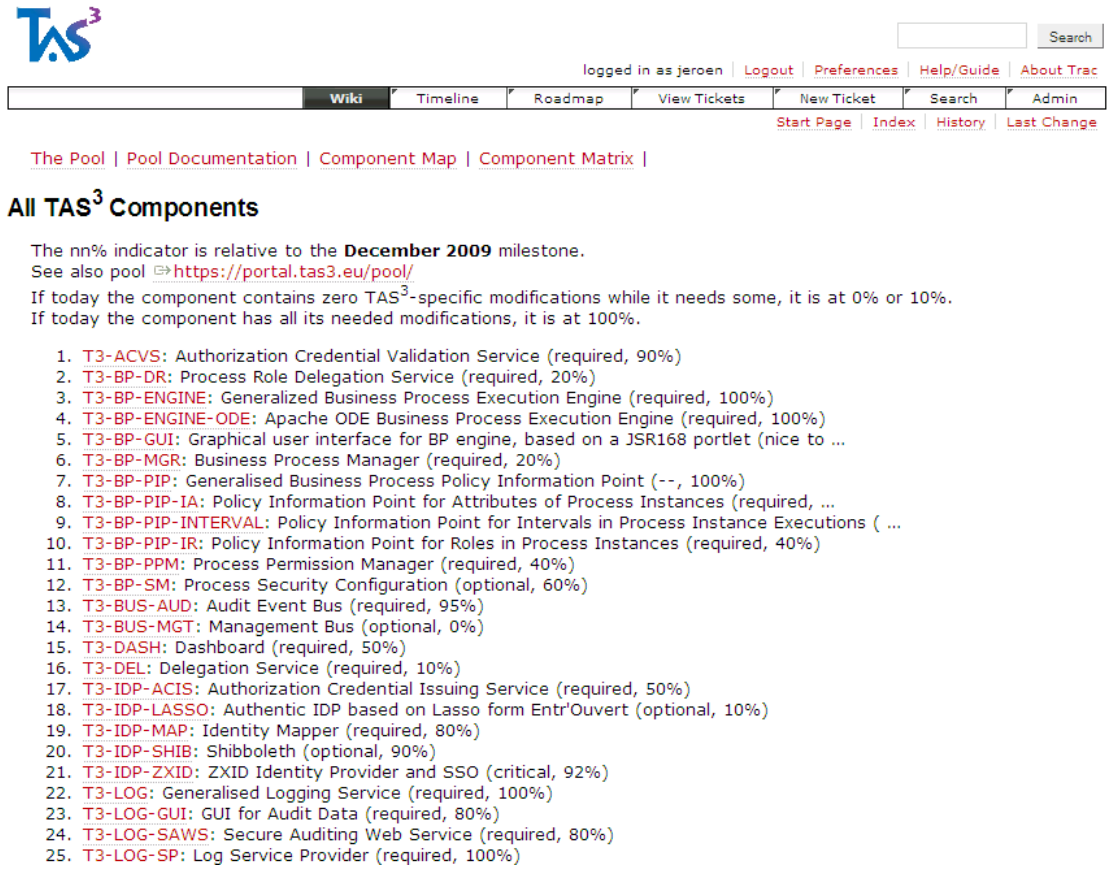


**Figure 6: The Component List in the Wiki**

The wiki provides a formal “home page” for each component at a standard spot, which can be updated by everybody to reflect component status. All component pages (Wiki, Pool, Map. Matrix) have been cross-linked and in some cases even embedded in each other. The Wiki Component List of Figure 6 is updated dynamically (by the Trac software) when an individual component page is updated.

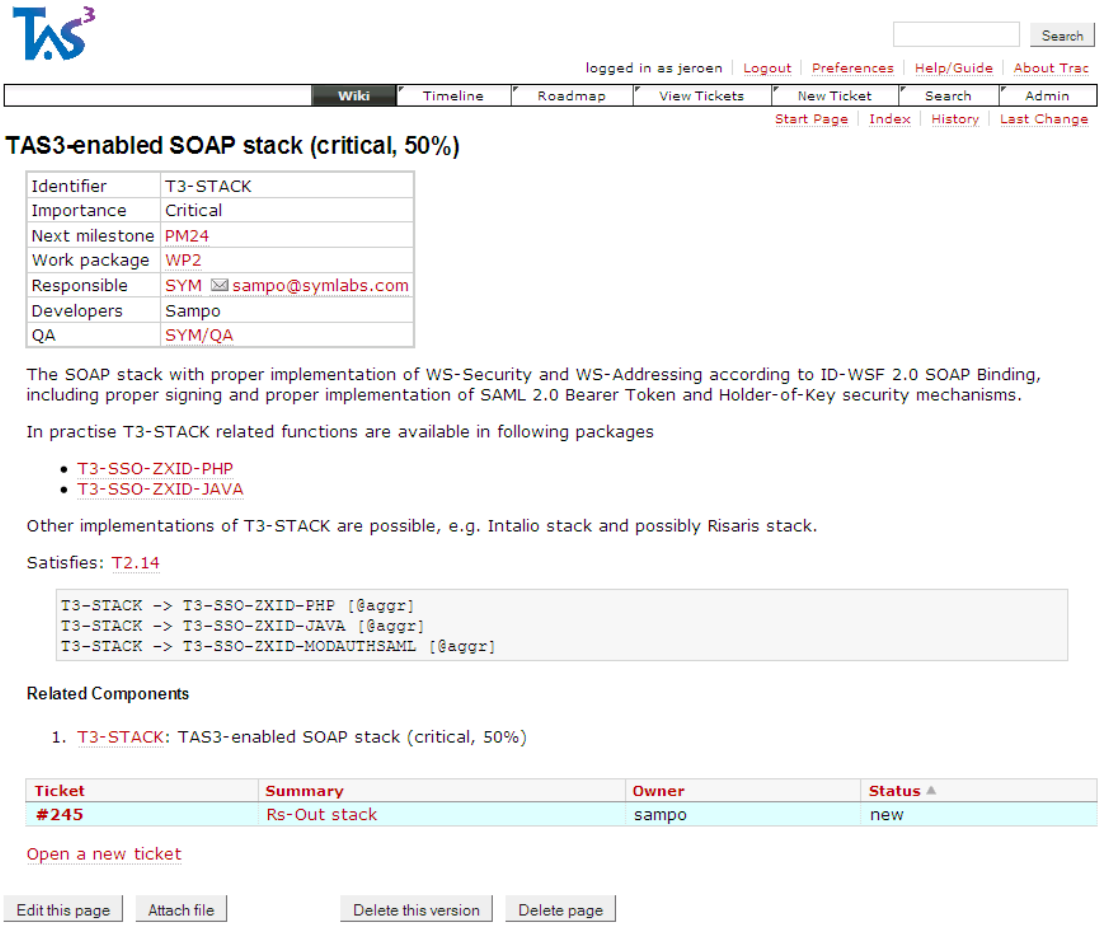


**Figure 7: Individual Component Wiki Page**

For each component, the individual page has a standard template with key information which is also used by the Matrix and Map generators of the Pool. The Map additionally uses component relationships, which also come from the Wiki page. Each component also has a dedicated ticket section, enabling quick overviews per component of development status.

## 6.2.4 Tickets and Ticket Queries

Trac has a standard ticket database which we adapted for TAS$^3$ usage. Very formal release-based milestones and other global constraints have been relaxed to reflect the distributed development approach of the project. The ticket work flow (Figure 8) has been adapted from standard Trac to allow both tasks, defect tickets, and enhancement request tickets to be handled within the same structure and database.

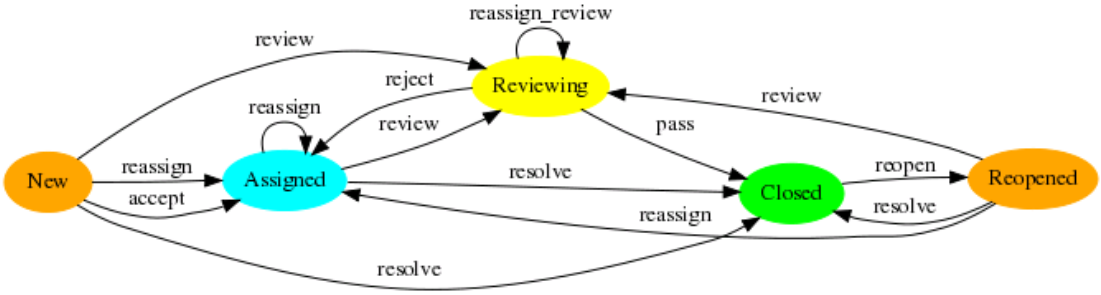


**Figure 8: TAS³ Ticket Workflow**

Standard priorities have been simplified so that there is less confusion about the urgency of a ticket, though fixing priorities are not centrally handled. Currently the Technical Board is regularly presented with the list of Blocking and Critical open issues and asked for response.

**Complete Task List**

**Priority: blocker** (5 matches)

| Ticket | Summary | Owner | Type | Status | Milestone | Component |
|---|---|---|---|---|---|---|
| #186 | Audit Bus + Trust Services | christian | RFC | assigned | | T3-BUS-AUD |
| ~~#12~~ | Investigate and secure compromised host backup1.tas3.eu | thomas | task | closed | | Central Servers |
| ~~#72~~ | Add all deliverables to the component dropdown list | jeroen | task | closed | | Portal |
| ~~#74~~ | Write up programme for Karlsruhe, Dublin, and Nice workshops | jeroen | task | closed | | Software Design |
| #185 | Policies for Nottingham Demo | george | task | new | | Architecture |

**Priority: critical** (24 matches)

| Ticket | Summary | Owner | Type | Status | Milestone | Component |
|---|---|---|---|---|---|---|
| #3 | Document the full setup of alpha.tas3.eu | jeroen | task | assigned | | Central Servers |
| #62 | Requirements - SAWS | david | RFC | assigned | PM24 | Architecture |
| #64 | Workshop Planning | thomas | RFC | assigned | | D11.8 |
| #76 | Finalize components from Risaris | jeroen | task | assigned | | Component Pool |
| #197 | Assure interface availability of all your components | jutta | task | assigned | PM24 | D12.1 |
| #198 | Assure interface availability of all your components | david | task | assigned | PM24 | D12.1 |
| #200 | Assure interface availability of all your components | sean | task | assigned | PM24 | D12.1 |
| ~~#30~~ | Add more people to the Wiki | jeroen | task | closed | | Portal |
| ~~#50~~ | Alpha needs a reboot but has no console | jeroen | task | closed | | Central Servers |
| ~~#53~~ | Define programme of Budapest Arch&Int workshop | jeroen | task | closed | | Component Pool |
| ~~#68~~ | waiting for lunch | jeroen | task | closed | | Architecture |
| ~~#70~~ | Increase component package upload size to 50Mb | jeroen | task | closed | | Component Pool |

**Figure 9: Typical Tickets**

A complete ticket can be updated by anybody, as all activities are logged for historical reference. We do not use tickets as a formal way of work distribution, and ticket monitoring is on a best effort basis. Ticket owners and followers get e-mail notification as soon as a ticket has been updated in any way.

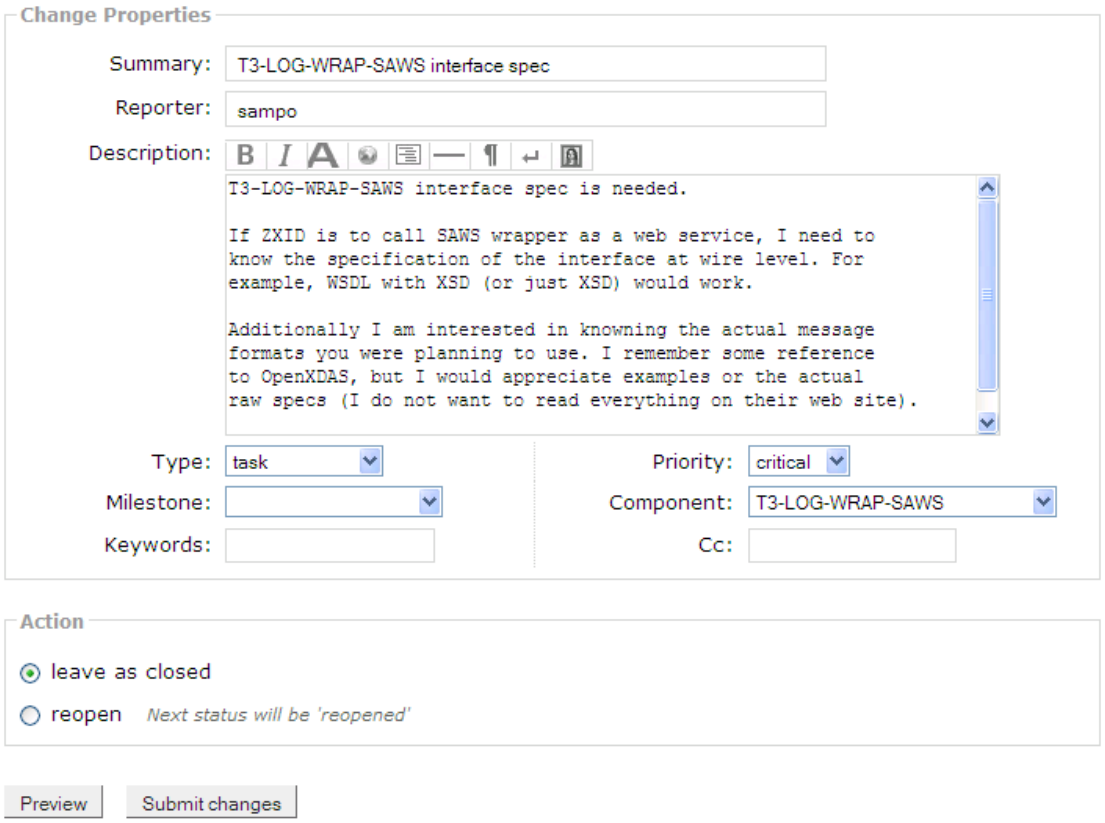


**Figure 10: A Typical Ticket**

When tickets are routinely handed out for activities associated to a designated milestone, such as a planned release of a report or a software component, Trac offers the possibility to plot "percent completed" charts for all milestones. As such an approach assumes that *all* the work for a milestone has been converted into tickets, the project currently has limited benefit from these milestone charts (called Roadmap in Trac terminology).

### 6.2.5 Full View on Wiki Activities

Since all Wiki pages and issue tickets are fully managed in a database and fully logged, historical tracking of all Wiki objects is made possible. A nice overview of what is happening can be quickly gotten from the Timeline page, which is a valuable project management tool to spot the activities.

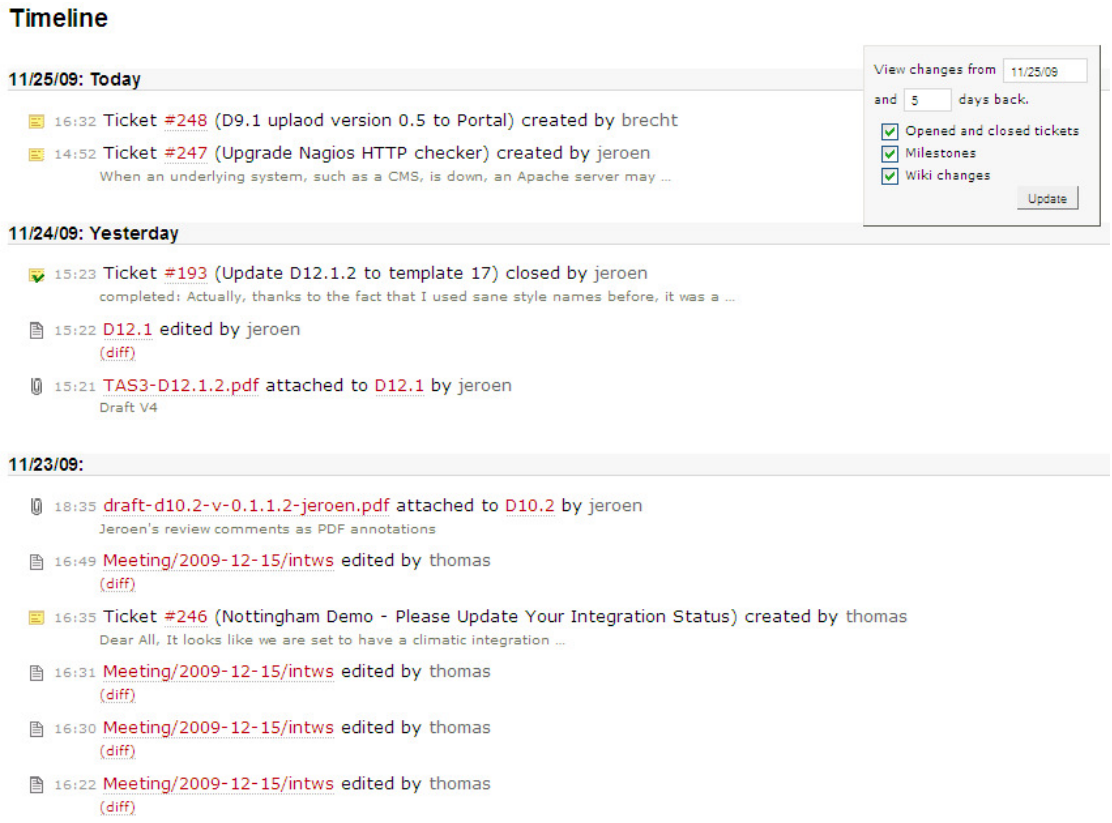


**Figure 11: Wiki Timeline**

## 6.3 The TAS$^3$ Component Pool

To support of the TAS3 Integration Process (Chapter 3), a custom subsystemof the Portal has been developed. It allows upstream developers to submit a component package, via a designated Package Maintainer, to the central store of components ready for integration testing by other developers. By linking formal component identifiers to versioned software packages, a clear overview of available packages and their suitability for integration can be created.

The Pool has been built by a combination of PHP5[15], Linux file system features, the Trac wiki, and the GraphViz[16] graph plotting package. It is maintained using the regular Subversion[17]version control system and part of the TAS$^3$ ticket workflow, so that developers can file issues against it. In December 2009, it supports all required aspects of the Integration Process up to, but not including creating a Stable Release. When the project advances to this level, the Pool will be adjusted accordingly.

### 6.3.1 Manifest File

Central to the Pool concept is the component package. Each package is a self-contained set of files plus a Manifest Document, which describes the package in a

[15] http://www.php.net/
[16] http://www.graphviz.org/
[17] http://subversion.tigris.org/

machine-readable way. The Pool software can read these Manifest files directly inside compressed package files, thereby avoiding to keep metadata outside the packages.

The Manifest file is a flat text file (UTF-8 without byte-order mark if you want diacriticals) which is based on Debian packaging standards. It contains lines with text, most of which follow a key:value pattern. An example:

```
TAS3 component package, version 0.1.
Package: T3-PDP-M
Source: http://bla.de.bla/
Version: 1.2.0a
Maintainer: University of Kent <d.w.chadwick@kent.ac.uk>
License: BSD
Depends:
Replaces: T3-PDP-M (<< 1.2.0)
Architecture: i386
Description: Stub implementation of the Master PDP
 This package contains the stub which is useful to test if PDP callers
 can successfully connect. It always returns "allow", but in the correct
 XACML format.
X-MyOwnKey: This is just for myself.
X-CVSID: $Id: Manifest.T3-SSO-ZXID-PHP,v 1.2 2009-09-16 14:59:11 sampo$
```

The first line is required, verbatim as shown here. Future versions of the Manifest file may change this, the whole line is used as manifest version ID. The order of the lines is not important, but it is recommended to keep the given ordering. Empty lines and lines starting with a hash mark # are ignored.

**Package** (required) is the package name. It must follow the TAS3 convention, T3-XXX-YYY.

**Source** (optional) is the URI of the sources used to build the package. If you packaged Apache, just refer to where you got the sources. This may be a CVSROOT or Subversion URI as well.

**Version** (required) is your package version. The exact construction of the string is up to you, but you do yourself a favour if it orders alphabetically. No component package with the same package name and version string can exist in the Pool.

**Maintainer** (required) is the contact point for the component. Bug reports filed to the component will be addressed to the maintainer.

**License** (required) lists the license(s) under which this package is distributed. The TAS3 project has decided on a list of allowable licenses: BSD, Apache, LGPL, MPL, GPL. It is allowed to release a package under a multiple license, indicate this by slashes: MPL/LGPL/GPL. If you want to specify a version number, postfix it to the license code: GPL3.0. For external components only, the license that comes with the component is allowed to be less free than for native components. If the external component has a different license, use this format: PROP license name. For example, PROP Intalio | BPMS Community Edition License.

**Depends** (optional, but will become required): indicates which other T3 packages are required locally for the current package to function. These dependencies

should be limited to what a typical developer or tester will need locally. If your component needs Apache or Tomcat to run, these are dependencies. If your package is a PEP which needs a PDP somewhere on the network, this is not a dependency. However, you can use the Suggests line to indicate which other package(s) you typically expect at the far end of the network link. See the separate subsection on Dependency Syntax for the line syntax.

**Suggests** (optional): typical packages that are used together with the current package. See Depends for more information.

**Architecture** (optional): if the package is CPU architecture-dependent (such as when it contains compiled C binaries), indicate which architectures it supports. If left out, "all" is assumed (with most Perl and Java systems this is the case, and many C source code packages have also been written CPU-independently). Currently there is no syntax or vocabulary enforced for this line.

**Description** (required) is a one-line description of your package, for listings. Do not end it with a period (dot) (it will be removed anyway). Since many terminal windows are 80 chars wide, try to keep the description shorter than about 60 chars.

If you follow Description: with one or more lines that start with at least one space, these lines will be attached to the Description. All line feeds will be retained. This allows for more descriptive comments. The Pool may use either the one-line description, or just the long description, or both, depending on the use case. If you really must include an empty line in the long description, use a single period . to create it (a really empty line would be ignored).

**X-...** (optional): keys starting with "X-" are subject to syntax checking, but their contents is not used by the package mechanism. You can use these for your own purposes. Some of these may eventually lose their "X-" and become official keys.

### 6.3.2 Manifest Dependency Syntax

For standardisation reasons, we pick up the complete existing syntaxis used by the Debian package managers. This may be a bit over the top, but has the advantage of a proven system that is known to work well.

Both the *Depends* and the *Suggests* fields have a uniform syntax. They are a list of package names separated by commas. These package names may also be lists of alternative package names, separated by vertical bar symbols `|' (pipe symbols).

The fields may restrict their applicability to particular versions of each named package. These versions are listed in parentheses after each individual package name, and they should contain a relation from the list below followed by the version number. The relations allowed are: <<, <=, =, >= and >> for strictly earlier, earlier or equal, exactly equal, later or equal and strictly later, respectively. For example,

```
Depends: T3-FOO (>= 1.2), T3-PEP-BAR (= 1.3.4), T3-PDP
Recommends: T3-IDP (>> 4.0.7), T3-QUUX, T3-QUUX-FOO (<= 7.6)
```

### 6.3.3 Pool Interfaces

When submitting packages to the Pool, two interfaces can be used: a regular HTTP/HTML interface (also suitable for machine submissions as the protocol is stable and published), and a CVS or SCP-based lower level interface which is intended to be integrated in software build and release scripts. The HTML interface is shown in Figure 12. The system supports both blocking filters (the required entries in the Manifest file, plus any processing error) and warning filters.

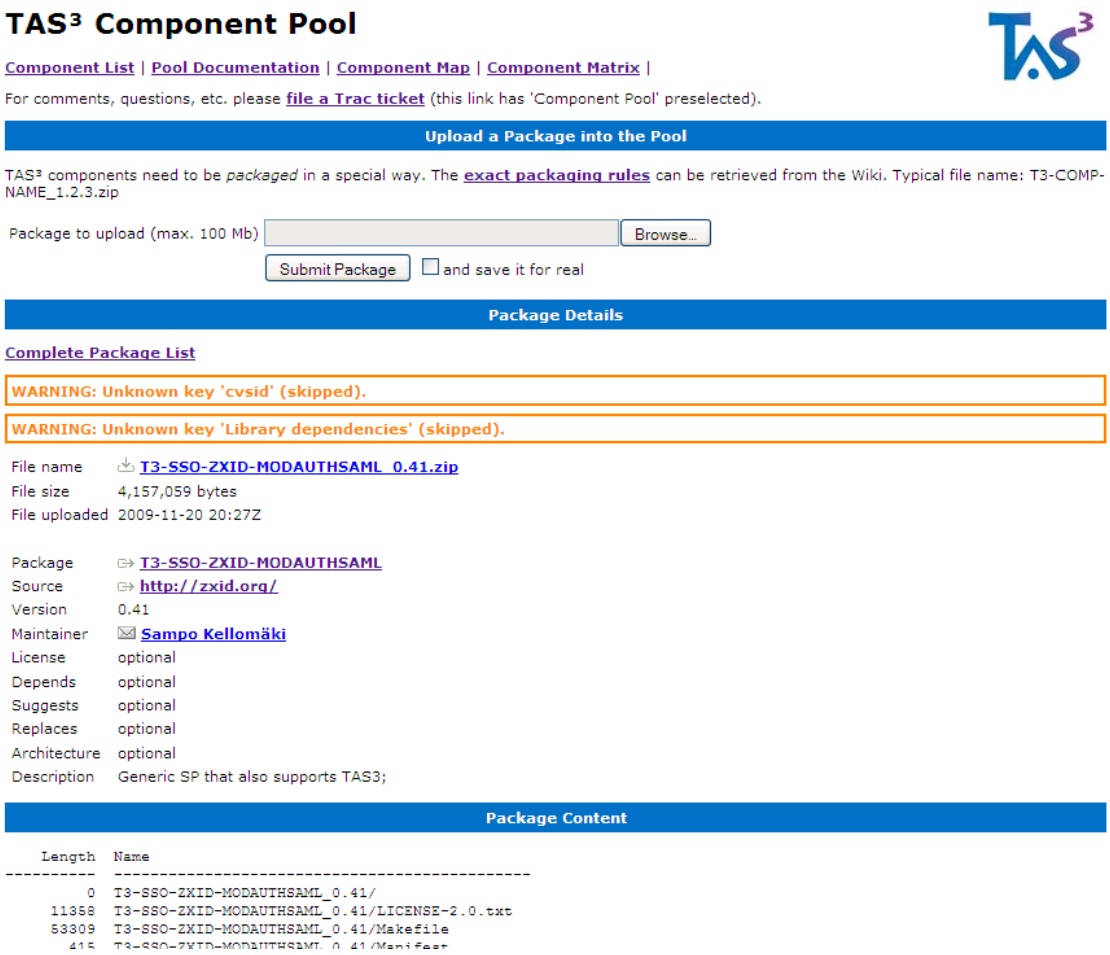


**Figure 12: Individual Pool Submission Details**

A variant on the HTML interface does not show one single component, but a summary list of components that match a filter. The default filter creates the Pool home page, which is a full overview of all components available in the Pool.

## TAS³ Component Matrix

Component List | Component Pool | Component Map |

The %complete is relative to **the component's next milestone**.

**Matrix generated 2009-11-25 16:04 UTC (once per hour)**

Download as CSV

| Component | Importance | Compl | MileSt | Pool | WP | Maintainer | Developers |
|---|---|---|---|---|---|---|---|
| T3-ACVS | Required | 90% | PM24 | 0.1 | WP2 | Stijn | Stijn, George |
| T3-BP-DR | required | 20% | PM30 | -- | WP3 | jens.mueller@ipd.uka.de | Jens, Thorsten |
| T3-BP-ENGINE | -- | 100% | -- | -- | WP3 | -- | -- |
| T3-BP-ENGINE-ODE | Required | 100% | PM24 | 1.0 | WP3 | Thorsten Haberecht | Thorsten, Jens |
| T3-BP-GUI | Required | % | PM24 | 0.1 | WP8 | Michael Kutscher | Michael Kutscher |
| T3-BP-MGR | Required | 20% | PM24 | 0.1 | WP3 | Jutta | |
| T3-BP-PIP | -- | 100% | -- | -- | WP3 | -- | -- |
| T3-BP-PIP-IA | required | % | PM30 | 0.1 | WP3 | jens.mueller@kit.edu | Jens, Thorsten |
| T3-BP-PIP-INTERVAL | Optional | % | PM24 | 0.1 | WP3 | jens.mueller@kit.edu | Jens, Inna |
| T3-BP-PIP-IR | Required | 40% | PM24 | 0.1 | WP3 | jens.mueller@ipd.uka.de | Jens, Thorsten |
| T3-BP-PPM | Required | 40% | PM24 | 0.1 | WP3 | jens.mueller@ipd.uka.de | Jens, Thorsten |
| T3-BP-SM | Optional | 60% | PM24 | 0.1 | WP3 | muelle@ipd.uka.de | Jutta, Anna |

**Figure 13: Pool Component Matrix**

Figure 13 shows another Pool interface, which is a live mashup of the Wiki component pages and the Pool submissions. This overview offers project management a concise list of component status, progress, and availability as real downloadable package. It is regularly used and can also be downloaded as CSV file for import in spreadsheet software.

The last interface to the Pool is the Component Map, such as in Figure 14. This graphical layout is generated straight out of the Wiki pages, and therefore needs no extra maintenance. Please be aware that the relationships shown are not the data flow or any architectural relationship: they are just software dependencies and not normative or standardized.

It is clear that at this stage in the development, a great number of components is being built. It can be expected that combinations of components into larger units will turn out to be effective. The goal of the Pool is to have meaningful units available in the form of packages, not every individual source module or library. Changes in component names and dropping of existing components as they are absorbed by larger components is to be expected.

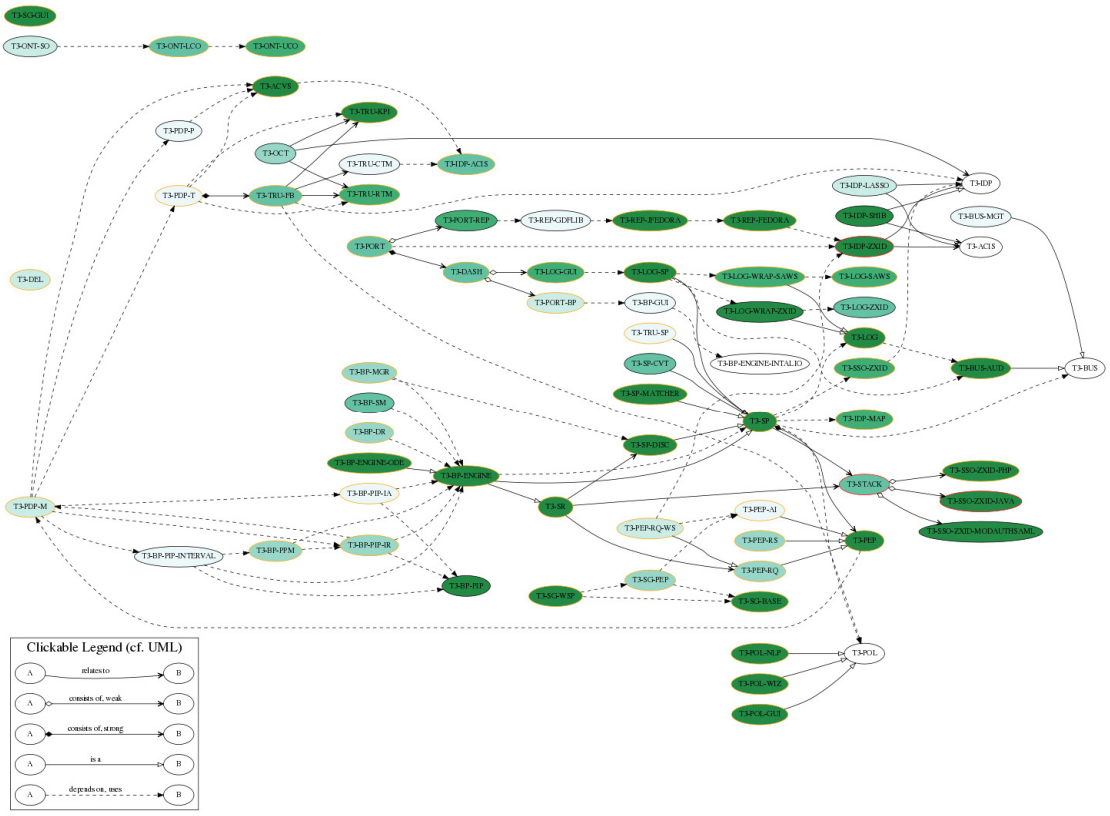


**Figure 14: Pool Component Map**

## 6.4 The VM Circus

Next to the Pool, WP12 maintains the Circus, which is a commonly managed set of virtual (Xen) servers that host subsets of components for demonstrations.

In the TAS[3] release process, a Stable release is made at a given moment by earmarking the set of components that at the release moment is in Testing. The Pool software allows to collect these components into a convenient bag. If required, we can set up a virtual machine and install the components as a group, providing a stable (non-changing) environment which typically is used for demonstrations or end-to-end testing.

The nature of many TAS[3] components makes that they are not by themselves performing any relevant function, and hence cannot be deployed. They are either libraries that need to be included in other products, or tools that do not have a human interface beyond a command line. This kind of component does not benefit from a VM testing environment.

## 6.5 The TAS[3] Mailing List Server

KU Leuven has designated their organisational mailing list server as the TAS[3] project list server. On the server we maintain two dozen lists varying from very busy technical discussion lists to formal announcements. The used software is

standard LISTSERV and the project is a resource consumer only, we perform no management or maintenance except the occasional user change.

## 6.6 Circle of Trust Manager and IDP

Organisationally these two components are not part of the Portal yet. For further details, please see the ZXID[18] documentation. When we integrate the ZXID single sign-on technology deployed in the TAS[3] applications with the Portal (planned for January-February 2010) and add the Yubikey[19] authentication token support, these components will become part of the Portal and will be described appropriately.

## 6.7 Document Templates

Although not really a tool, several document templates both for Word and for LaTeX have been produced via WP12 and have been used in all external and most internal TAS[3] project communications.

[18] http://www.zxid.org/

[19] http://www.yubico.com/products/yubikey/

**Amendment History**

| Ver | Date | Author | Description/Comments |
|---|---|---|---|
| 1 | 2009-08-10 | JH | First draft for PM24 release. |
| 2 | 2009-08-20 | JH | Lots of technical details and How-Tos added. |
| 3 | 2009-10-01 | JH | More added. The document now reflects the state of the system, except for the Pool which needs to go in next. |
| 4 | 2009-11-24 | JH | The new TAS3 document template is added. |
| 5 | 2009-12-07 | JH | Full addition of all project resources by WP12. |
| 6 | 2009-12-29 | JH | Final version. |