perfSONAR Hand over process

Authors	Luis Marta, Loukik Kudarimoti
Date	04-12-06
Current Version	1.6

Document Change Log

As SA3-WI15 Document			
Version number	**Date**	**Description of change**	**People**
1	02-11-06	First draft issued	Luís Marta
1.1	09-11-06	Changed template and content based on Szymon's suggestions seen in metadata configuration file. Added more content to many sections	Loukik Kudarimoti
1.2	10-11-06	Added Section on Micro releases	Loukik Kudarimoti
1.3	14-11-06	Added section on Bundle Releases, Bug Fixing and Summary of Documents.	Luís Marta
1.4	15-11-06	Added sections for pS-BASE and updating. Made some changes to other sections	Loukik Kudarimoti
1.5	30-11-06	Changes according to Nicholas Simar comments, and Development Process is now section 2.	Luís Marta
1.6	04-12-06	Added sections on removal of branches (bundle and micro) based on Jeff Boote's suggestions. Added templates	Loukik Kudarimoti

Table of Contents

1. Introduction

This document describes the Hand over process for perfSONAR releases. Hand over is a process which ensures that the services that are contained in the perfSONAR releases meet the expected high levels of quality. This ensures that the users have a good experience in using the software and also helps the perfSONAR support team in providing support to a well tested product. Hand over process (release management in general) also defines the time intervals and protocols for testing, documentation, roll out, etc.

Hand over in a nut shell:

- Specify a list of documents to be provided by the developers along with the software
- Specify timelines for documents and software, based on dates provided by release strategy
- Quality Checks
  - Test “Handed over” software
  - Check the quality of documentation provided
  - Call upon the developer to fix bugs/problems
- Bundling
  - Hand over process is required if different products need to be bundled together to look like one product
  - Products which are bundled together have a similar installation procedure and user experience
  - Bundling also helps in improving the user experience while installing the software

2. Development Process

When a developer starts working on a new version of a service, he/she must already know what new features are expected to result from his/her effort.

The timing of this development must also be very well known and based according to the release strategy which defines the release date for next version of software and also the list of features to be includes

During development, all of which happens on the CURRENT branch, the developer should start writing documentation about the service he/she is coding. During this phase, the developers will be required to deliver the following two documents, preferably at the beginning of the development stage:

- Functional Specification of service (features/functionality available in the service such as getKey, getData, register, de-register, query, etc)
- Interface specification (syntax and semantics to be used for all applicable features listed above- the inputs and outputs of each service with XML Schema, examples of XML messages and the rationale behind these messages)

These documents define the features and also the syntax and semantics of each service and they are very important for the Testing team to start designing and even coding the testing scripts, so that functional testing of the service might start as soon as possible.

When the development phase draws to a close or when the code is nearing stability, the developer would need to start preparing for a 'micro-release'. The audience for this micro-release could be the perfSONAR Bundle release team or the end users (usually early adopters of service). The micro-release process has been explained in section (2a) above. It includes the creation of micro-STABLE and micro-RELEASE branches for the development.

The developer/development team would create a micro-RELEASE branch for their development according to specifications in section (2a) when they think that their development is ready to be released and included within the bundle. This phase is called the 'hand-over' phase. During this phase, the developers must also deliver the following documents, relative to the new version of the service, to the Release team:

- Specification of Ant targets (installation instructions - in case of perfSONAR services in Perl)
- Sample configuration files
- Sample Metadata configuration files

As soon as the Release team has all the information about the specification of Ant targets for all the services, it must be incorporated with the installation scripts of the final release.

Not all services will finish development and have its own micro-RELEASE branch at the same time. So, the developers may continue to develop on the CURRENT branch, while functional testing is done by the Testing team on the micro-RELEASE branch containing the handed-over stable development.

Due to resource constraints, it is possible that not all services will be tested by the dedicated testing team. In such cases the development team will have to 'wear the hat' of a testing team and test their development. It is also possible to exchange this testing effort with some other development team so that both the teams will be testing each others development and this might improve the quality of testing. In all these cases, the development teams will be notified well in advance that the dedicated testing teams won't be able to do any testing for their developments.

3. Micro-Releases

From release 1.1 onwards, it has been agreed that each service will go through its own release before (and sometimes independent of) the perfSONAR bundle/package release management. For the purpose of clarity in communications, we term these as Micro-releases.

A micro-release is usually managed by the developer/development team in charge of the service development. If the products resulting out of such micro-releases want to make it into the perfSONAR bundle, it is necessary for the micro-release process to take the following requirements into consideration.

a. Branches in Micro-Release Management

The convention with most release managements is to have at least 3 branches. We propose to follow the same approach for all micro-release managements as well. However, the only mandatory branch from the release management team's perspective is the Release Branch. The Current and Stable branches are seen as a best practice that each development team may or may not follow.

Current – This is where current development is taking place. Latest developments and latest code, albeit untested, probably unstable and unreleased, are available here. This is usually the HEAD or TRUNK

Stable – Once a code reaches stability and is being prepared for a release, it is placed in the stable branch. There is usually an assigned CODE FREEZE date by which all the code planned to be contained in the next release is expected to be present in the stable branch. It is the responsibility of each development team to define and enforce such CODE FREEZE dates for their developments and micro-releases,

With respect to micro-release management, a Stable branch is usually needed when there are more than a couple of developers in the development team in charge of a service. A Stable branch per service development is suggested as best practice by the perfSONAR release management team. However, following this principle and enforcing it is left to the team in charge of the development and micro release of the service

The suggested naming convention for Stable Branch is

```
<unique-service-name>-<service-type>-STABLE
```

An example:

RRDType-MA-STABLE

Care has to be taken to make sure that the service name is non-repetitive. For example, if two organisations are developing RRD MA, the addition of characteristic and/or the organisation name to the <unique-service-name> is suggested.

Release – This is the only mandatory branch in Micro-Release Management. This branch contains the code which is being 'handed over' to the release management team. It is first created by the development team of the service being handed over.

Once the code is handed over, the perfSONAR bundle release management team will take control of it, make it read only and control all commits to the branch. The code will then go through a testing phase before the final bundle release. If bugs are found during testing phase, one of the following two approaches will be taken:

1) pS bundle release team will give up control of the release branch for that service, ask the developers to fix the bug and start the hand over process all over again.

2) pS bundle release team will notify the development team and will allow commits from the development team which are aimed at fixing the identified bug.

Most small bugs found during testing will get the (2) treatment. Serious bugs and performance problems will receive the (1) treatment above.

The obligatory naming convention for the 'per Service Release Branch' is

```
<unique-service-name>-<service-type>-Release-<date-in-ddmmyyyy-format>
```

An example:

RRDType-MA-Release-05062006

Care has to be taken to make sure that the service name is non-repetitive. For example, if two organisations are developing RRD MA, the addition of characteristic and/or the organisation name to the <unique-service-name> is suggested. Further, it is best if the unique service name is the same as the unique service name of the stable branch and any previous release branches.

b. Release Candidates in Micro-Release Management

The Release Candidates in this Micro-Release Management are formed from the Release Branch being handed over from the development team to the pS bundle release team. This Release Branch concept for Micro-Release management has been explained in the preceding section.

c. Removal of Micro-Release branches

Micro-release branches are used for bundle releases. A micro-release branch is useful (to the bundle release management and support team only) if the bundle release that it is a part of is supported. For more information on bundle releases and removal of bundle release branches, see section below

If the bundle release management team decides to remove some micro-release branches, the development teams will definitely be consulted before doing so. At this stage, it is the responsibility of the development team to take appropriate actions such as merge micro-release branch with the trunk/head or any other branch, etc.

4. Bundle Releases

While each perfSONAR service has its own micro-release on a per-service level, we must have a perfSONAR bundle/package release management, where different products constitute a suite and look like one product.
The different services which are bundled together must have a similar installation procedure and user experience.

a. Release Candidates in Bundle Releases

When all the services that have been chosen to take part on the new release are on their separated micro-RELEASE branches, and all the listed documents have been delivered to the Release team, a RELEASE branch is created for that Bundle Release, joining the most recent versions of the different micro-RELEASE branches of the services.

If the development of a service and its documentation are not ready on the date defined to create the bundle-RELEASE branch, now is the moment for the Release team to decide if the wait is worthwhile, or if the service will be dropped from this bundle release.

All modifications to RELEASE branch require approval of the Release team. Only serious bug fixes or security issues will warrant changes in the RELEASE branch at this time.

Release candidates should be created from this branch, and tested by the community of users and by the Testing team. Not only the services, but also the installation procedure should be tested at this time. As the Testing team had already started functional testing on some of the services, it is expected that the possibility of finding bugs and other issues on those services is somehow reduced.

According to the bugs and issues found by testing, it may be necessary to launch several Release Candidates, until one can be turned into the final release.

b. Branches for Bundle Releases

When a Bundle Release is to be created by the Release Team, and the release process starts, the release team must decide what services go into the release. For one release, there will be one micro-RELEASE branch per service, and for the next bundle release, there will be a new micro-RELEASE branch per service. Then, as stated in the previous section, the Release Team will create a RELEASE branch for that Bundle Release, joining the most recent versions of the different micro-RELEASE branches of the services.

c. Removal of bundle release branches

The release management and support teams will only support a particular number of releases. As of now we don't have a blanket rule saying that only a particular number of bundle releases will be supported.

When the release management team and the support teams decide to provide no further support to a particular version of the bundle (based on release strategies, criteria for support, etc), the users will be advised of this decision and if possible, they will be advised to upgrade to newer versions. The branches that contain code for such bundle releases will be marked for removal. All micro-release branches which are only part of such bundle versions (but none other which are supported) will also be marked for removal. Development teams will of course be contacted to inform them of this decision so that they have the opportunity to salvage something from such branches or retain such branches for any particular reason.

d. PerfSONAR – BASE

The content in this section only applies to software (services) that have been written in Java and which use the perfSONAR base classes which have also been written in Java. There are currently two approaches for handling perfSONAR-BASE within the services.

Up to a maximum of one version of base per service included in the bundle

In this approach, which is going to be used for the time being, any service included in the bundle can have any version of the base. The bundle can contain multiple versions of the same base in the bundle (and hence different versions of the same class) but up to a maximum one version for each service

The advantages of this approach are:

- If a service has advanced features because of changes to the base but they are still inter-operable with other services (such as the Lookup Service), they can be included in the bundle. The outstanding question is whether this case will ever come up.
- It requires lesser preparation or it could result in the reduction of effort needed to fix the perfSONAR-BASE version contained in the bundle for all services to work with.
- If a developer has less time to make his/her service compatible with pS-BASE version, its still possible to include their service in the bundle

The disadvantages of this approach are:

- Two versions of the same class will end up present in the bundle. In fact, all services make use of two jar files : perfsonar-generic.jar and perfsonar-rrdma.jar for example. This means that there will be many perfsonar-generic.jar files or they might need to be renamed to perfsonar-generic-rrdma.jar for example
- Any bug/feature request in the perfSONAR-BASE would result in the support team having to call upon all the development teams

Another Approach: One version of base for all services included in the bundle

This approach requires that all services included within the bundle be able to work with just one version of perfSONAR BASE classes. In order to implement this approach, advance planning is needed to determine the features that would be supported by the base and also the date by which the pS-base will be ready. This pS-base will end up being the one included in the upcoming bundle release. Enough time will need to be allocated to the developers to ensure that if their service is planned to be included within the bundle, the service should work with the specified bundle.

The advantages of this approach are:

- Services are considerably easier to support and update in case a bug appears or a new feature is requested
- The pace of development of different services can be better controlled
- Two versions of the same class will never end up being included in the bundle

The disadvantages of this approach are:

- It requires considerable planning and specifications
- Services cannot be included if they don't work with the specified version of the base (even if they interact with other services as well as any other service in the bundle)

This is an approach that we are not going to follow at this time but we will consider it for future releases.

5. Bug Fixing

During the Hand over process, the developers will be required to give their micro-RELEASE branch name to the Release team. If the Testing team discovers small bugs during testing, the Release team will ask the developers to fix them in the same micro-Release branch. If the Testing team finds serious bugs, security issues, performance problems, then the Release team won't accept the micro-RELEASE branch, and will ask the developers for a new one.

Accordingly, if a bug, or other issue of similar nature, is found in the first version of the micro-RELEASE, the Release team will either ask the developers to fix it within that version, or if it the issues are already fixed within a newer version of the service (on a more recent micro-release), then the Release team won't ask for a bug fixing but rather simply ask the users to upgrade to the latest release.

If a bug is found in a particular version of a micro-release, an investigation will need to be carried out to find out if the discovered bug affects any newer or older versions of micro-release. In such cases, the policy would be to certainly fix the bugs in the newest versions of micro-release and the user who is affected by the bug should be asked to upgrade to the newest version. Older versions of micro-releases could be fixed for the same bug on a case-by-case basis and potentially based on approval (for example, if there is a large deployment community who are unable to upgrade to the newest version for a recognized reason).

6. Updating deployed software

In order to make sure that the latest features, bug fixes and enhancements are applied to the users' deployments, it is necessary to have a software update process which is both intuitive and effortless. This can be achieved in one or combination of the following approaches

a. Manual process

In this process, the user can be asked to download a new piece of software which would interact with the user in order to find out more information about the deployed services. This software would then determine if any of the deployed services need to be updated and if so, it would take any required action.

b. Automatic process

In this process, the deployed software would either periodically and automatically check with a configured URL as to whether any updates are available for the deployed

services or the deployed software would have an updater which needs to be triggered by the user to discover if there are any updates. Once discovered, it would download the updates and apply them to the deployed services.

In both possibilities above, there might be some actions expected from the user such as metadata re-configuration, etc depending on the updates. It is recognized that such actions will need to be made as effortless as possible and also automatic if possible.

The release management team is currently undecided on the correct approach for this process. The dates by which these features will be available are undecided as well.

7. Summary of Documents for Hand-over process

During development of the service, and preferably on the first stages of it, the developers must deliver the following documents:

- Functional Specification of service (syntax)
- Interface specification (semantics - the inputs and outputs of each service with XML Schema and examples of XML messages)

As soon as the development phase ends for a service, these are the documents expected by the Release team:

- Specification of Ant targets (installation instructions - in case of perfSONAR services in Perl)
- Sample configuration files
- Sample Metadata configuration files

In the Appendix for this document, the developers can find the templates for these documents.

Appendix – Templates of Documents

- Template for Functional Specification of a Service
  - http://wiki.perfsonar.net/jra1-wiki/index.php/Image:Service-Functional-Specification-Template.pdf

- Template for Interface Specification
  - http://wiki.perfsonar.net/jra1-wiki/index.php/Image:Service-Interface-spec-template.pdf

- Template for specifying installation actions
  - http://wiki.perfsonar.net/jra1-wiki/index.php/Image:Service-Installation-targets-spec-template.pdf

- Template for sample configuration files
  - http://wiki.perfsonar.net/jra1-wiki/index.php/Image:Sample-config-file-template.pdf

- Template for metadata configuration files
  - http://wiki.perfsonar.net/jra1-wiki/index.php/Image:Sample-metadata-file-template.pdf