

Pause6 Implementation

Mark overmeer* Sam Vilain†

September 11, 2006

Abstract

This document describes implementation details for Pause6, such as file formats and use cases. This results in XML Schema's, which define the implementation requirements.

Contents

1	Archive layout	2
1.1	pause6-constitution	2
1.2	pause6-references	3
1.3	pause6-index	3
1.4	Project releases	4
1.5	Archive references	5
2	File formats	6
2.1	General project files	6
2.1.1	The meta-data file	6
2.1.2	The release-log file	7
2.1.3	The status files	8
2.2	Constitution specific files	9
3	Use cases	10
3.1	Manage a project	10
3.2	Submitting a releases	10
3.3	FTP-server	11
3.4	Daemon global configuration	12
3.5	User local configuration	13
3.6	Optimizing storage	13

*markov@cpan.org (The Netherlands) <http://solutions.overmeer.net>

†samv@cpan.org, Catalyst IT (New Zealand) <http://catalyst.net.nz/>

A Schema: Pause6 basic types	15
A.1 Schema wrapper	15
A.2 Releases	15
A.3 Release states	17
A.4 Release states	17
A.5 Release meta-data	20
A.6 Identities	20
A.7 Signatures	22
A.8 Checksums	23
A.9 Other	24
B Schema: Pause6 file formats	25
B.1 Schema wrapper	25
B.2 File release-log.xml	25
B.3 File archive-log.xml	26
B.4 File archive-list.xml	27
B.5 Files status-current.xml and status-next.xml	27
B.6 File constitution.xml	28
B.7 File repository.xml	29
C Schema: Pause6 messages	30
C.1 Schema wrapper	30
C.2 Accept signature	31
C.3 Get release status	32
References	32
XSD components index	33

Introduction

The CPAN6/Pause6 project is described in a few different papers. This paper describes file formats and use cases specific to the Pause6 implementation. This information is useless without the terminology definitions from paper [1] (“CPAN6 and Pause6 Design”), and the base implementation of CPAN6 in paper [2] (“CPAN6 Implementation”).

1 Archive layout

1.1 pause6-constitution

The project named “pause6-constitution” contains the configuration of the archive. The constitution is published as a release (a directory with some files in it) signed by a sufficiently large number of board members. It must be visible to all archive accessors.

The constitution contains the following file:

- `constitution.xml`: is the main configuration file
- `***name-space-layout.xsd`

1.2 pause6-references

The board members control this project “pause6-references”, which is publicly available information about the configuration with less restrictions than the constitution. Usually one board member can change this settings, where the constitution requires all-but-one signature.

- `archive-list.xml`: list of archive abbreviations
- `deployer-list.xml`: list of deployers

archive-list

The archive list is a single file which maps archive alias names to project names within the archive. These archive reference type of projects are simply directories with some data in it, especially the location and the public encryption key. Those project names will usually start with `archive-`, although that is just by convention.

Archive references have a trust component in them: that part of the reference data is stored in the `archive-list`, and has to thereby is signed by the board before it is published to the users.

The archive reference project information can get updates. The board may decide that the owners of the referred archives are permitted to update their referencing information, for instance can update the public key themselves.

deployer-list

Deployers are defined as identities, which on their turn are also maintained as simple projects. The preferred project names for these deployers start with `deployer-`.

The board may decide to open-up that name-space to everyone, so that people can add themselves as `deployer` without interference. A reviewer process is required to check the correctness of the configuration in such case. The reviewer can update the `deployer-list`.

1.3 pause6-index

The “pause6-index” project contains the name-space administration of the archive. The index is a directory, like all Pause6 items are kept like project directories. However, in this case the directory will usually not get signatures and versions and therefore stays hidden to normal users.

The board can decide to publish this index on regular intervals as project, or hide it such that people can only query the archive via the regular interface.

The index will contain:

- `release-list.xml`: listing of releases
- `archive-log.xml`: global overview of modifications

release-list

A single file lists all releases. Per release, it contains the humanly understandable project name (in utf8, blanks and other nasty characters permitted), a version indicator (utf8, syntax unspecified), project type, status flag, and a location. The constitution can restrict project-names and version numbers with regular expressions.

In general, it is not possible to use project names and version numbers in actual file-names, because these strings may contain characters which are not permitted in the platform dependent filename syntax. Therefore, releases will get a (random) number assigned during upload, and the path to the release administration (the location) will be derived from that number.

The location may also contain `<archive>:<project>`, in which case the browser will need to check the `archive-list` for additional information and retry the query to the new location.

The project indicator can not contain an explicit version number; these redirections are used for the merging of archives: either to solve naming conflicts or to do a lazy `-non-download, abstract-merge`. In either case, the version identification does not change.

archive-log

This file gives an overall activity overview on the releases. When the archive contains many releases, inspecting their individual moves will get expensive. This log will show which releases changed their state. The archive-log can be used to inform deployers and derived archives which releases they need to inspect to become up-to-date.

The archive-log will grow with one line per release change. Happily, the number of state changes per release has a relatively modest maximum. However, in case this file grows too large it may get condensed, leaving only the last state change per release.

1.4 Project releases

In the next chapter we will detail more on the expected content of projects, and how to work with them. On the administrative side of projects, we see the following files (again: together in one directory per release)

- `meta-data.xml`: contains the meta-data of the release, which will not change over time;
- `release-log.xml`: detailed trace on all actions involving this release;
- `status-current.xml`: extract of the log, listing what has sufficient signatures. (The current state of the release)
- `status-next.xml`: extract of the log, listing what may not have sufficient signatures yet. (The next state of the release)
- `repository.xml`: contains the actual location (or locations) of each of the non-inlined release components.

The reasoning behind this becomes more clear with the examples shown in paper [2].

meta-data

The meta-data file contains meta-data which is provided by the publisher on the moment the release was uploaded: project name, version number, publisher's identity, list of filenames, description, licence, the current list of project authors, and some project maintenance rules (a kind of project private constitution).

release-log

The release-log contains all changes and all changeable meta-data: signatures of authors, explicit project state changes (for instance deprecation), release expiration.

In Perl5 distribution terms: the `Makefile.PL` and `META.yml` files are used to specify some meta-data information. Pause6 compliant user tools can extract this information to provide Pause6 with sufficient data in a standardized format.

status-current

The status file is a combination of all static information from the meta-data file and an extract of the dynamic release-log information. This information is completed with sufficient the signatures of authors and CPAN6 commissioners, deployers, and so on on that data section of the file.

Normal users will work with this status-current file: before they download actual files, they need to inspect this meta-data.

status-next

This file is for the authors of the project: it contains the same static meta information but a slightly different dynamic release-log extract: showing the next state of the release. The number attached signatures is not yet sufficient to tell the users.

1.5 Archive references

In a large mesh of archives, it is hard to collect contact information. Therefore, information of remote archives can be kept as "projects" inside an archive.

In most cases, these archives will have an alias defined in the constitutional archive-list. In any case the reverse is true: each alias declared in the archive list requires an archive reference project.

The archive directory contains the following files:

- `definition.xml`: describing the remote archive
- public keys and other info required to trust the initial contact to the archive.

The board may decide that other external people own (and maintain) these references, or can keep this in their own hands.

definition

This file lists the url of the archive, information about the archive's type, and contact protocol information. Amongst others, the name of the file which contains the public key is listed. That key can be used if the public key infrastructure cannot be reached (i.e. no network available or limited system capabilities).

2 File formats

2.1 General project files

The concept of 'projects' is mainly for the users, but not that much for the Pause6 internals. Projects (release names) are used for name-space allocation, and the version string is used to define (optional and artificial) ideas about a sequence: which release was last?

The Pause6 archiver uses a unique code to store a release, because the (project) name a version string are free-format utf8 and therefore can conflict with the file-system capabilities. For the same reason, filenames as used by the project will get mutilated by the Pause6 administration.

Each of the files is readable in informal YAML or formal validatable XML. See the XML specification is section ??.

2.1.1 The meta-data file

The meta data is kept per release: over time, any aspect of a project can change: authors, description, rules, the name, and so on. Changes in next releases of a project should not destroy information from the past.

The meta-data has to be provided by the client application. In some enviroments –like perl5–, there is already an infrastructure present to collect these facts: Pause6 requires a uniform representation of that data. During upload, the client-side tool will convert the meta-data into a Pause6 structure like:

```
1 project: Mail::Box
2 version: 2.065
3 authors: pause-id:markov pause-id:sam
4 licence: GPLv3
5 files:
6     filename: Mail::Box-v2.065.tar.gz
7 archives:
8     -
9     name: pause-id
10    type: pause6
11    address: http://pause.cpan.org
```

2.1.2 The release-log file

Each release directory contains a release-log file, which contains an incremental trace on the changes. The format is (globally):

```
1  -
2  type: request
3  at: Mon May 1 08:52:23 CEST 2006
4  by: pause-id:markov
5  state: upload
6  -
7  type: trace
8  at: Mon May 1 08:52:23 CEST 2006
9  log: publisher validated, trust connection 10
10 -
11 type: trace
12 at: Mon May 1 08:52:23 CEST 2006
13 log: upload file
14 by: pause-id:markov
15 filename: Mail::Box-v2.065.tar.gz
16 size: 23123
17 checksum: sha1 SDFIWUOWHCJKHGWIUHWKIC2384729f3scewjhk
18 -
19 type: trace
20 at: Mon May 1 08:52:28 CEST 2006
21 change: store file
22 checksum: sha1 SDFIWUOWHCJKHGWIUHWKIC2384729f3scewjhk
23 filename: SDF/WUO/SDFIWUOWHCJ
24 -
25 type: signature
26 at: Mon May 1 08:55:00 CEST 2006
27 by: pause-id:markov
28 signer: pause-id:markov
29 type: sha1
30 authograph: sha1 DCIOWUFW#$$*FWFHKJ@#YKJ
31 -
32 type: confirm
33 at: Mon May 1 08:55:00 CEST 2006
34 change:
35     state: published
36 -
37 type: confirm
38 at: Mon May 1 08:57:00 CEST 2006
39 change:
40     state: released
```

2.1.3 The status files

The `status-current` and `status-next` files are extractions of the `release-log`.

Whenever needed, these status files can be regenerated from the log. The 'next' collects the latest information, but does not require sufficient signatures: it is used to sign release changes. The 'current' collects the latest information with sufficient signatures. If the board decides to change the constitution, that may have implications for the status files: they need to be regenerated.

```
1  release:
2    name: Mail::Box
3    version: v2.065
4    publisher: pause-id:markov
5    authors: pause-id:markov pause-id:sam
6    licence: GPLv3
7    state: released
8    files:
9      -
10     filename: Mail::Box-v2.065.tar.gz
11     checksum: sha1 GoJWlg9avwW4MzSXbtRZrZPygAw
12     size: 592389
13  archives:
14    -
15     name: pause-id
16     type: pause6
17     location: http://pause.cpan.org
18    -
19     name: pgp
20     type: pgp
21  signatures:
22    -
23     signer: pause-id:markov
24     type: pgp
25     autograph: "@#DFKJO@UWUFH@YC"
26    -
27     archiver:
28       role: commissioner
29       type: pause6
30       address: http://cpan.org/cpan5/
31       trust: connection 10, publisher 90
32     signer: gpg:cpan5@cpan.org
33     type: sha1
34     autograph: sha1 DCIOWUFW#$$*FWFHKJ@#YKJC
35    -
36     archiver:
37       role: deployer
38       type: pause6
```

```
39     address: http://perl.example.com/cpan
40     trust: connection 100, commissioner 80
41     signer: gpg:mark@example.com
42     autograph: ASJFIOWURWOHCWjh
43
```

The authors and publisher all sign the `release-next` file. Each person signs the part up to the dashes, and sends that personal signature to the commissioner, which will add it to the `release-log` if the signature is correct. Then a new `release-next` is generated.

When enough signatures were received for the next release state, the commissioner will replace the `release-current` with the next information. Still, more signatures may come in: from other authors who thereby add trust to the release. It is thinkable that signatures are received for project states which already have passed: a warning will be issued in that case, and the signature ignored.

When the release has moved to the next state on the commissioner, that change will be reflected in the global `archive-log`. That log-file is used to update the deployers. Deployers only require a signature from the commissioner to trust the data they receive: they make check the publisher or authors as well, but that data may be expired.

When the release is copied from archive to archive, the commissioner and deployer parts get repeated. That way, the user can trace how a release came to his system. The trace is required to establish the trust of the release.

If possible, the user will check the signature of the release from his most trusted listed author. Otherwise, the publisher, commissioner, or deployer. The last option is needed, for instance, when everything is delivered off-line on CD/DVD. For instance, during system installation or in a secure environment or when installing a new Linux distribution from scratch. The calculated trust on authenticity must come above an user adaptable threshold or ask for human intervention.

2.2 Constitution specific files

The Constitution describes rules of the archive, as decreed by archive's board. Notably, it does not contain any release or owner information.

This is an example how "`xml2yaml constitution.xml`" could look like:

```
1  archive:
2    alias: cpan-traditional
3    address: http://perl.overmeer.net/cpan
4  board:
5    member: pause-id:people/SAMV
6    member: cacert:john@lnet.nl
7  permissions:
8    class: author-defaults
9    required-votes:
10   minimum: all
```

3 Use cases

In this section, a few use cases are described. The order is rather random, expressing various ideas which got a place in the design.

The list of use-cases will certainly be extended in future releases of this paper.

3.1 Manage a project

If someone wants to claim a name-space, he simply uploads a first release. Of course, the publisher must authenticate itself to a level which satisfies the board. For instance, the publisher must authenticate as part of a certain authority or be listed on a manually picked list of acceptable authenticities.

Not only the identity is required to start a project: it must also comply to name-space rules, for instance in the structure of the name and used characters. Besides, it may require one or more signatures of the board to start-off.

If you fulfilled the requirements, you can upload your first release. That release will produce the initial meta-data as required for searching, configuration and such.

The current set of authors is kept in the meta-data of the release. The information provided for the latest ‘released’ release contains the current set of authors. Until the first release, the publisher the the author.

When no versioning scheme is used, the release of the project which became ‘released’ last (and still is in that state) is providing that information. The concepts of authors and release ordering are configurable.

When a project wants to change name or to split-up, those new names will be considered new projects. Simply submit a final release of under the original name which depends on its follow-ups. Dependencies are defined in the release meta-data.

3.2 Submitting a releases

When a publisher sends a new release to the commissioner, it will check that the user provided prove of identity is sincere (enough) and acceptable for the archive. When the project name of the release was not seen before, a new project is seeded, with the publisher as author. Otherwise, the permission rules relating to the last release for this project are checked to see whether the publisher is allowed to upload a new version.

The commissioner transports the uploaded data files to the archive related repository within its store. Those files are checked to have the correct checksum before they are written to the store. The size, the checksum, and the original filename of all the files in the release are kept in the meta-data of the release, and will be signed.

When all files have been received, the publisher will download the meta-data file as constructed by the commissioner. It will sign this data with his private key, sending his signature on the data to the commissioner. Then the release state will become ‘published’.

On the moment that the release is published, all authors of the project will be able to see (and download) the release meta-data. They may even get an e-mail to inform them that a signature on the release is needed.

3.3 FTP-server

Can we offer additional services to existing ftp-servers? Quite easily: they already have their upload and distribution network in place. The additional service that CPAN6 can offer are:

- checking that the data on the ftp-servers is the same on all servers
- offer package search facilities to users (no detailed search)
- automating the search for mirror servers to the users,
- provide the user with install tools which also do the download,
- administration of installed software.

To set this up is quite simple: take one of the ftp-servers and flag that one as “the store of the commissioner”. Scan the directories of that server every hour (or less) for new files.

The hardest thing is to translate the newly found files into releases. In many cases, ftp-releases are just single file, which makes this a none-operation. Sometimes, the archive adds checks files. Some releases are more complex. Then, a version number needs to be derived (otherwise the files time-stamp is used).

It may be possible to extract more meta-data from the item found on the ftp-server, for instance if it is an rpm-package. It may be possible to do some extra checks on the item, for instance because it contains MD5 checksum (like in rpm-packages). In general, the person who is creating the CPAN6 wrapper will need to spend some time collecting meta-data which is otherwise provided by the publisher.

Once the meta-data is found, and checksum are generated from the files, the release can be added and directly move into the ‘released’ state. The process which collected the meta-data will probably sign the data it has collected, so the source can be traced. All discovered releases in the archive will have the discovering process as author.

The deployer’s have a much simpler task. As usual, they ask the commissioner for detected releases. They do not have to use a scribe to copy the data, because that is already done by the existing ftp-mirror infrastructure. The only thing they have to do is to check whether their archive contains the same files: check the checksums on the files found.

The default deployer functionality is very close to that of the rpmfind websites: they help locating released material within a set, within the archive. As limitation, deployers will (in the first implementation of Pause6 at least) not know about archive content so not provide for searches inside the released material.

3.4 Daemon global configuration

The examples in figures 1 and 2 show the top-level system global archive and one of the contained archives. Each box is a directory with some data-files, together in a repository on an accessible store. The system administrators found the top-level

“global”	address: http://archives.example.com
Constitution	project: pause6-constitution authors: board = sysadmins daemon configuration rules.
Archive-ref	project: user authors: sysadmins, publisher: user:root connects to local username table.
Archive-ref	project: tutorials authors: user:root, publisher: pgp:john published conference material.
Archive-ref	project: pgp connects to the pgp/gpg infrastructure
etc...	
Identity	project: pause6-daemon authors: sysadmins commissioner, deployer, scribe public key.

Figure 1: Example: global archive

archive, so are by default the administrators of the top-level archive. They may decide that (a selected set of) users can start their own archive (with or without signature of the board) or that they have to initiate that themselves.

“tutorials”	address: http://archives.example.com/tutorials
Constitution	project: pause6-constitution authors: board = pgp:john archive configuration rules.
Publication	project: yapc-eu-cpan6.odt publisher: pgp:markov OpenOffice presentation for YAPC::EU.
etc...	

Figure 2: Example: archive “tutorial”

An archive reference has a (project-)name, and can thereby be downloaded and installed in the user’s environment. The content will lead the user to the url of the referred archive, which may be anything. By default –and common practice– the xyz archive-ref in archive `http://abc` will lead you to `http://abc/xyz`.

The constitution of the global archive limits the configuration possibilities of the lower level archives, as those constitutions will reduce the permissions that the projects can set. For instance, the sysadmins can forbid any archive to accept publications from

non-authors. For this reason, treat archive references as symbolic links.

3.5 User local configuration

“local”	address: file:/home/markov/.cpan6/
Constitution	project: pause6-constitution authors: board = user:markov personal configuration rules.
Archive-ref	project: global connects to global archive.
Archive-ref	project: tutorials reference to global:tutorials.
Archive-ref	project: pauseid reference to http://pauseid.cpan.org
etc...	
Identity	author: pauseid:markov one of my multiple identities.

Figure 3: Example: user archive

3.6 Optimizing storage

The amount of files and storage can be improved by taking into account that various projects use the same files: parallel installed releases from one project, and releases from different projects may have overlapping code. For instance, many projects will distribute the same license file.

As more complex example, the ImageMagick Perl module requires the ImageMagick C libraries to be installed. The Python version of the interface requires the same external library. To be certain that you have the right version of all libraries, and header-files, you have to be very careful about project dependencies.

The solution of many Windows applications is to ship an application with all its requirements as one big integrated block. All dependency configuration is avoided this way. When a user runs multiple related applications on the system at the same time, multiple copies of exactly the same binary may have to be loaded into memory. This spoils disk-space, memory, and therewith system performance.

The GIT version control system introduces a nice concept. Based on the idea that you can determine a simple unique key per file (using MD5), you can easily check whether the file you need is already on the system. The concept of a *product release* did change from a *collection of versioned files* to simply a *collection of files*: that two following project releases both contain a file with the same name but with a different content is more a ‘coincident’ than a relation. Of course, applications running outside this VCS may interpret it as more than that.

Seeing a release simply as a set of related files which are selected by they checksum, as GIT introduced, has a few considerations:

- multiple archives may build on the same set of files, to represent different *views*. For instance, one archive could show a sub-set of the files which can be installed, an other view selects only the documentation files, a third includes the core files of the project plus extra files to build an rpm, a fourth view only shows that rpm. Users can decide which archive they want to address, for instance to copy to local disk. A server may decide to only deploy the pure documentation archive (to create a search facility). To create a (Linux or FreeBSD) distribution CD, one may take a clone of the rpm archive.
- you do not need to apply a sequence of patches to go from a file(name) in one release to the same file(name) in the next one, so both the VCS as the client side can be dumb in this respect. The data can be stored on any general ftp-server or directory structure.
- the release index (part of the releases meta-data, provided at upload) defines that is most important: which set of files will work together. That is more important than tracing the changes between two releases. The precise changes are not important for end-users, in general.
- some (graphical) tool can help you maintaining a project by displaying the related views in different archives. For instance, the tool can detect that the rpm is out-dated because one of its building-bricks has a new release.

Application

The tric of unifying equivalent files from various sources based on their checksum can be applied on many places. On the stores, the files will need to be kept some form of encoded, because CPAN6 permits utf8 filenames and version numbers without restrictions: many file-systems, operating-systems, and transport protocols can not handle those names. Therefore, the release index needs to connect the abstract transport filename with an original filename. Only on the user's system, mutilations may be needed to get the data to work. In the ideal world, the user's system understands utf8 filenames.

Stores may consider to use the checksum of the file as abstract filename. Client-applications may be smart enough to keep track on the checksums of downloaded and installed files to avoid duplicate downloading. Client applications may install a file which is already available elsewhere on the system by creating a hard-link.

MD5 checksums are quite weak: it is possible to produce a different file with the same MD5 within minutes. Accidental collisions are close to impossible, but they are insufficient to proof trust. Therefore, a better checksum is advised, either additional in the index or as filename. Which type of checksum is used is part in the trust computation.

Perl5

The Perl community distributes code as tar-balls (files packed into one tar archive which then is compressed using gzip) This has the advantage that all files are loaded at once, but postpones the above optimization of file-folding to after the downloading.

Besides, how to handle utf8 filenames within the tar archive (conversions not supported)?

It may be useful to change the current tar-ball practice for these reasons.

A Schema: Pause6 basic types

The schema defined in this section defines basic types used for Pause6. They extend the basic types of CPAN6.

A.1 Schema wrapper

This schema uses the Dublin Core (<http://dublincore.org>) definitions for document meta-data.

```
1 <schema
2   xmlns="http://www.w3.org/2001/XMLSchema"
3   elementFormDefault="qualified"
4
5   targetNamespace="http://cpan6.net/200609/pause6-basic"
6   schemaLocation="https://xml.cpan6.net/schema/200609/pause6-basic.xsd"
7   version="1.0"
8
9   xmlns:p6="http://cpan6.net/200609/pause6-basic"
10  xmlns:c6="http://cpan6.net/200609/cpan6-basic"
11  xmlns:dc="http://purl.org/dc/terms/"
12 >
13
14 <import
15   namespace="http://cpan6.net/200609/cpan6-basic"
16   location="https://xml.cpan6.net/schema/200609/cpan6-basic.xsd" />
17
18 <import
19   namespace="http://purl.org/dc/terms/"
20   location="http://dublincore.org/schemas/xmls/qdc/2006/01/06/dcterms.xsd" />
```

A.2 Releases

release

Pause6 adds security to CPAN6 releases. The publisher is kept per release. The creator is only used to honor people who contributed to the package, but has not additional value.

Each release can specify a new set of authors and permissions. Both authors and permissions are only effective when the release is accepted. So, whether the publisher is permitted to upload, change authors or permissions is defined by the previous release. The acceptable changes also depend on the rules in the constitution.

This data is used to initiate an upload. When the settings violate the current rules, they will not be accepted and the upload not started.

```
1 <complexType name="release">
2   <complexContent>
3     <extension base="c6:release">
4       <sequence>
5         <element name="creator" type="p6:identity"
6           minOccurs="0" maxOccurs="unbounded" />
7         <element name="publisher" type="p6:identity" />
8         <element name="author" type="p6:identity"
9           minOccurs="0" maxOccurs="unbounded" />
10        <element name="permit" type="p6:permissions" />
11        <element name="plan" type="p6:release-plan"
12          minOccurs="0" />
13      </sequence>
14      <attributeGroup ref="c6:release-label" />
15    </extension>
16  </complexContent>
17 </complexType>
```

Pause6 defines some additional release types:

- x-cpan6/pause6-constitution,
- x-cpan6/pause6-index,
- x-cpan6/pause6-archive-ref,
- x-cpan6/pause6-identity, x-cpan6/pause6-license, and
- x-cpan6/pause6-process. The latter defines a daemon or scribe.

release-component

Security for files is implemented by protected complex checksum for each of the published files. When more than one checksum is provided, they will use different formats. Only the ‘best’ format which is supported by the client will need to be checked.

```
1 <complexType name="release-component">
2   <complexContent>
3     <extension base="c6:release-component">
4       <sequence>
5         <element name="checksum" type="checksum"
6           minOccurs="0" maxOccurs="unbounded" />
7       </sequence>
8     </extension>
9   </complexContent>
10 </complexType>
```

release-state-change

Each state change is issued by someone or the cpan6 daemon; in any case, the

identity needs to be included. Then, zero or more signatures are required to get this state change accepted.

```
1 <complexType name="release-state-change">
2   <complexContent>
3     <extension base="p6:initiated">
4       <sequence>
5         <element name="state" type="p6:release-state" />
6       </sequence>
7     </extension>
8   </complexContent>
9 </complexType>
```

A.3 Release states

release-state

A release will move from one state to the other. When a release is in uploading state, it will not be available to deployers. When in published or embargo state, it will only be available to authors. The embargo means that the project has already sufficient signatures to go into the released state, but will be kept back by the daemon until the embargo is lifted (see item A.4)

Releases which are expired may still be available, although a clean-up may happen during download of the release (which will result in an error). The deprecated state of a release means that there are known problems with the release. The client may warn users to upgrade.

The installed state is used in the end-user archive. Usually, the project was in released state when it got installed. The user will be warned when the scribe (downloader) detects that release gets flagged as expired, rejected, or deprecated on the source archive.

```
1 <simpleType name="release-state">
2   <restriction base="token">
3     <enumeration value="uploading" />
4     <enumeration value="published" />
5     <enumeration value="embargo" />
6     <enumeration value="released" />
7     <enumeration value="deprecated" />
8     <enumeration value="expired" />
9     <enumeration value="rejected" />
10    <enumeration value="installed" />
11  </restriction>
12 </simpleType>
```

A.4 Release states

release-plan

To change the release plan, you need the release-plan-change rights.

```

1 <complexType name="release-plan">
2   <sequence>
3     <element name="embargo" type="p6:embargo-time"
4       minOccurs="0" />
5     <element name="blocked" type="p6:blocked"
6       minOccurs="0" />
7     <element name="expires" type="p6:expiration"
8       minOccurs="0" />
9     <element name="deprecated" type="p6:deprecated"
10      minOccurs="0" />
11    <element name="has-followup" type="p6:has-followup"
12      minOccurs="0" maxOccurs="unbounded" />
13  </sequence>
14 </complexType>

```

embargo-time

Releases can be uploaded under embargo, which will block the transition from published until released state until the condition is reached. Meanwhile, the release will be stuck in the embargo state, it will not be distributed to the deployers, and only available to authors.

When the release was already released before the embargo was set, this probably is done to correct a mistake. In this case, it is important not to alert the users who already got the data. The release will still move into embargo state, but users who have it installed or downloaded must not be warned by the client software.

```

1 <complexType name="embargo-time">
2   <complexContent>
3     <extension base="p6:initiated">
4       <element name="after" type="date" />
5     </extension>
6   </complexContent>
7 </complexType>

```

blocked

Whatever happens to the state of the release in the source archive, that will not have any effect to the local archive. For instance, you do not want an installed release to expire, or you do not want the external release to be used in any case.

```

1 <complexType name="blocked">
2   <complexContent>
3     <extension base="p6:initiated" />
4   </complexContent>
5 </complexType>

```

expiration

Move the release to the expired state after this moment. When keep-last is set, the release will not expire if it does not have a follow-up.

```

1  <complexType name="expiration">
2    <complexContent>
3      <extension base="p6:initiated">
4        <sequence>
5          <element name="after" type="date" />
6          <element name="keep-last" type="boolean"
7            default="true" />
8        </sequence>
9      </extension>
10   </complexContent>
11 </complexType>

```

deprecate

The 'deprecated' state is a special form of 'released': the release can still be the last with respect to permissions, but will not be used on clients except when explicitly asked for. Clients will also report the deprecation of installed software releases to users.

```

1  <complexType name="deprecate">
2    <complexContent>
3      <extension base="p6:initiated">
4        <sequence>
5          <element name="after" type="date" />
6        </sequence>
7      </extension>
8    </complexContent>
9  </complexType>

```

has-followup

When a release (independent on its current state) is followed by a newer release (defined by the release order algorithm, not by the moment of release), it may automatically phase-out.

When the `minimal-newer` number of releases for this project in 'released' state are sorted after this release, then the changes will get scheduled (probably irreversibly).

When the condition is met, the specified `deprecate-after` duration is waited until the release is flagged to be 'deprecated'. Then, the `expire-after` duration is waited, before the release state changes into 'expired'. Finally, another `reject-after` time-span is waited before the release is prepared to be removed.

At least one of the fields must be specified, otherwise the whole plan is lost (removed). The intermediate states may take longer then specified. It may also be possible that the state changes cannot be implemented on-time, for instance when the whole archive administration is distributed on a disk. In that case, the one-shot deployer must handle these state changes and merge them dynamically into the static archive index.

```

1 <complexType name="has-followup">
2   <complexContent>
3     <extension base="p6:initiated">
4       <sequence>
5         <element name="minimal-newer" type="xs:int"
6           default="1" />
7         <element name="deprecate-after" type="xs:duration"
8           default="0D" />
9         <element name="expire-after" type="xs:duration"
10          default="0D" />
11        <element name="reject-after" type="xs:duration"
12          default="0D" />
13      </sequence>
14    </extension>
15  </complexContent>
16 </complexType>

```

A.5 Release meta-data

meta-data

Commissioners probably require some of these fields to be present, especially a title and a description. However, This data cannot be processed automatically and is not related to Pause6 or CPAN6's behavior.

As examples, an `creator` is added here if he or she has no provable identity. The `publisher` may represent a company or such in the traditional meaning of the word, responsible for the publication of the material, but not the actual uploading the data into the archive. So be careful what to put here.

```

1 <!-- see http://dublincore.org/documents/dces-terms/ -->
2 <complexType name="meta-data">
3   <sequence>
4     <group ref="dc:elementAndRefinementGroup" />
5   </sequence>
6 </complexType>

```

A.6 Identities

identity

An identity is used to reference publishers, authors, users, and processes. It is representing (one of the) current accreditentials that the person or process has, which can change over time. Therefore, it is a versioned “project” address: a “release”.

```

1 <simpleType name="identity">
2   <extension base="c6:release-address" />
3 </simpleType>

```

admin-rights

The list of permissions will certainly grow. A list of these rights is used for various purposes: to express capabilities, limits, and vote requirements.

```
1 <simpleType name="admin-rights">
2   <restriction base="token">
3     <enumeration value="constitution-change" />
4     <enumeration value="archive-list-change" />
5     <enumeration value="project-delete" />
6     <enumeration value="project-start" />
7     <enumeration value="release-publish" />
8     <enumeration value="release-embargo" />
9     <enumeration value="release-accept" /> <!--released-->
10    <enumeration value="release-reject" />
11    <enumeration value="release-plan-change" />
12  </restriction>
13 </simpleType>
```

permission-class

Used to indicate the community to which a set of permission applies. The `authors-defaults` and `authors-maximum` are set by the board to limit the freedom of the project authors. The `authors` and `non-authors` are used by project authors, as meta-data item inside a release.

```
1 <simpleType name="permission-class">
2   <restriction base="token">
3     <enumeration value="board-members" />
4     <enumeration value="authors" />
5     <enumeration value="author-defaults" />
6     <enumeration value="author-maximum" />
7     <enumeration value="non-authors" />
8   </restriction>
9 </simpleType>
```

permission-set

Groups sets of rights for one target community. All permission flags should only be used once within this container. Unspecified permission flags will have a default.

```
1 <complexType name="permission-set">
2   <sequence>
3     <element name="permissions" type="p6:required-signatures"
4       maxOccurs="unbounded" />
5   </sequence>
6   <attribute name="class" type="p6:permission-class"
7     use="required" />
8 </complexType>
```

A.7 Signatures

signature

One signature package, which can be checked. When the signer (attribute by) information (is a versioned release) is not available, or its version got deprecated, or the infrastructure to check this identity is not available, then the signature should be ignored. In other cases, an failing check should block use of the signed release.

```
1 <complexType name="signature">
2   <complexContent>
3     <extension base="p6:initiated">
4       <sequence>
5         <element name="type" type="p6:signature-type" />
6         <element name="encoding" type="c6:encoding-type" />
7         <element name="autograph" type="string" />
8       </sequence>
9     </extension>
10  </complexContent>
11 </complexType>
```

signature-type

```
1 <simpleType name="signature-type">
2   <restriction base="token">
3     <enumeration value="PGP" />
4     <enumeration value="SSL2" />
5     <enumeration value="SSL3" />
6   </restriction>
7 </simpleType>
```

required-signatures

The minimum number of signatures required for a set of permissions.

```
1 <complexType name="required-signatures">
2   <sequence>
3     <list name="permit" type="p6:admin-rights" />
4   </sequence>
5   <attribute name="minimum" type="p6:signatures-count"
6     use="required" />
7 </complexType>
```

signatures-count

The named situations are especially useful for the board, although it can be used everywhere. Symbolic names have the advantage when the number of authors or board changes: the count probably does not need to change in that case.

```

1 <simpleType name="signatures-count">
2   <union>
3     <simpleType>
4       <restriction base="int">
5         <minInclusive value="0" />
6       </restriction>
7     </simpleType>
8     <simpleType>
9       <restriction base="token">
10        <enumeration value="all" />
11        <enumeration value="all-but-one" />
12        <enumeration value="majority" />
13        <enumeration value="never" />
14        <enumeration value="none" /> <!-- 0 -->
15      </restriction>
16    </simpleType>
17  </union>
18 </simpleType>

```

A.8 Checksums

checksum-type

The checksums of the files are used to check the originality of the contained data. For this reason, it must be hard to create a different file with the same checksum. MD5 has been cracked in recent years: in little time (less than an hour) a new file with the same sum can be produced. Also SHA1 is reaching its end-of-life, although just outside reach of desktop processing power.

Its on the client to judge the quality of the various checksum algorithms. New algorithms will be added here when time progresses. One file may be checksummed by different algorithms, which usually adds trust.

The `inlined` is an exception. When the files is small with respect to the size of the checksum key, that file content will be the limiting factor in the security. Besides, it is expensive to fetch that data on a remote store. Therefore, in this case the commissioner may (will) decide to inline the data. Symlinks will always get inlined.

```

1 <simpleType name="checksum-type">
2   <restriction base="token">
3     <enumeration value="inlined" />
4     <enumeration value="MD5" />
5     <enumeration value="SHA1" />
6     <enumeration value="SHA128" />
7     <enumeration value="SHA256" />
8   </restriction>
9 </simpleType>

```

checksum

```
1 <complexType name="checksum">
2   <sequence>
3     <element name="type" type="p6:checksum-type" />
4     <element name="encoding" type="c6:encoding-type" />
5     <element name="code" type="string" />
6   </sequence>
7 </complexType>
```

A.9 Other

trust

Trust is a calculated value, based on the way data is distributed and signed. A higher value means a higher trust. The more processes and steps are involved in the process, the lower the trust will be.

The trust value is dynamic: when a release is signed with a very trusted public key (for instance exchanged during some meeting in person), and that key expires, all releases which were downloaded using that key will be lowered in trust immediately. This may cause warning messages to system administrators which use these releases.

```
1 <simpleType name="trust">
2   <restriction base="float">
3     <minInclusive value="0.0" />
4     <maxInclusive value="100.0" />
5   </restriction>
6 </simpleType>
```

initiated

Used as base-class to anything which needs to be logged, triggered by a human or a process. The identity usually lists the process which produced the log entry or the user who request something.

```
1 <complexType name="initiated">
2   <complexContent>
3     <sequence>
4       <element name="reason" type="string"
5         minOccurs="0" maxOccurs="unbounded" />
6     </sequence>
7     <attribute name="by" type="p6:identity" />
8     <attribute name="at" type="date" />
9   </complexContent>
10 </complexType>
```

B Schema: Pause6 file formats

B.1 Schema wrapper

```
1 <schema
2   xmlns="http://www.w3.org/2001/XMLSchema"
3   targetNamespace="http://cpan6.net/200609/pause6-files"
4   schemaLocation="https://xml.cpan6.net/schema/200609/pause6-files.xsd"
5   version="1.0"
6
7   xmlns:ff="http://cpan6.net/200609/pause6-files"
8   xmlns:p6="http://cpan6.net/200609/pause6-basic"
9   xmlns:c6="http://cpan6.net/200609/cpan6-basic"
10  >
11
12  <import
13    namespace="http://cpan6.net/200609/pause6-basic"
14    location="https://xml.cpan6.net/schema/200609/pause6-basic.xsd" />
15
16  <import
17    namespace="http://cpan6.net/200609/cpan6-basic"
18    location="https://xml.cpan6.net/schema/200609/cpan6-basic.xsd" />
```

B.2 File release-log.xml

release-log-entry

```
1 <simpleType name="release-log-entry">
2   <choice>
3     <element name="request" type="state-change-request" />
4     <element name="confirmed" type="state-change-confirmed" />
5     <element name="trace" type="daemon-trace" />
6     <element name="signature" type="state-approved" />
7   </choice>
8 </simpleType>
```

state-change-request

```
1 <complexType name="state-change-request">
2   <complexContent>
3     <extension base="p6:initiator">
4       <sequence>
5         <element name="state" type="p6:release-state" />
6       </sequence>
7     </extension>
```

```
8     </complexContent>
9 </complexType>
```

state-change-confirmed

```
1 <complexType name="state-change-confirmed">
2   <complexContent>
3     <extension base="p6:initiator">
4       <sequence>
5         <element name="state" type="p6:release-state" />
6         <element name="votes" type="p6:signatures-count" />
7       </sequence>
8     </extension>
9   </complexContent>
10 </complexType>
```

state-change-confirmed

```
1 <complexType name="state-approved">
2   <complexContent>
3     <extension base="p6:initiated">
4       <sequence>
5         <element name="signature" type="p6:signature" />
6       </sequence>
7     </extension>
8   </complexContent>
9 </complexType>
```

B.3 File archive-log.xml

archive-log-entry

```
1 <complexType name="archive-log-entry">
2   <complexContent>
3     <extension base="initiated">
4       <sequence>
5         <element name="release" type="ff:visible-release-change" />
6       </sequence>
7     </extension>
8   </complexContent>
9 </complexType>
```

visible-release-change

```

1 <complexType name="visible-release-change">
2   <sequence>
3     <element name="state" type="p6:release-state" />
4     <element name="total-size" type="c6:size" />
5   </sequence>
6   <attributeGroup ref="p6:release-label" />
7 </complexType>

```

B.4 File archive-list.xml

archive-list-entry

```

1 <complexType name="archive-list-entry">
2   <complexContent>
3     <extension base="p6:initiated">
4       <sequence>
5         <element name="archive" type="ff:archive-definition" />
6       </sequence>
7     </extension>
8   </complexContent>
9 </complexType>

```

archive-definition

```

1 <complexType name="archive-definition">
2   <sequence>
3     <element name="address" type="c6:archive-address" />
4     <element name="trust" type="p6:trust" />
5     <element name="project" type="c6:project-name" />
6   </sequence>
7   <attribute name="alias" type="c6:alias" />
8 </complexType>

```

B.5 Files status-current.xml and status-next.xml

release-state-which

```

1 <simpleType name="release-state-which">
2   <restriction base="token">
3     <enumeration value="current" />
4     <enumeration value="next" />
5   </restriction>
6 </simpleType>

```

status-to-sign

```
1 <complexType name="status-to-sign">
2   <sequence>
3     <element name="status-seqnr" type="int" />
4     <element name="release" type="p6:release" />
5     <element name="state" type="p6:release-state" />
6   </sequence>
7 </complexType>
8
```

release-status-description

```
1 <complexType name="release-status-description">
2   <sequence>
3     <element name="to-sign" type="ff:status-to-sign" />
4     <element name="signature" type="p6:signature"
5       minOccurs="0" maxOccurs="unbounded" />
6   </sequence>
7 </complexType>
```

B.6 File constitution.xml

constitution

```
1 <complexType name="constitution">
2   <sequence>
3     <element name="board" type="p6:board" />
4     <element name="permissions" type="p6:permission-set"
5       maxOccurs="unbounded" />
6     <element name="name-space-layout"
7       type="p6:name-space-layout" />
8     <element name="scribe" type="p6:identity"
9       maxOccurs="unbounded" />
10  </sequence>
11 </complexType>
```

archive-hint

```
1 <complexType name="archive-hint">
2   <sequence>
3     <element name="address" type="p6:archive-address" />
```

```

4         <element name="alias"    type="p6:alias"           />
5     </sequence>
6 </complexType>

```

name-space-layout

```

1 <complexType name="name-space-layout">
2     <sequence>
3         <!-- ordered matches! -->
4         <choice minOccurs="0" maxOccurs="unbounded">
5             <element name="valid-project-names" type="pattern" />
6             <element name="valid-version-labels" type="pattern" />
7             <element name="deployers"           type="pattern" />
8             <element name="scribes"            type="pattern" />
9             <element name="publications"       type="pattern" />
10            <element name="reserved"           type="pattern" />
11        </choice>
12
13        <element name="constitution" type="c6:project-name"
14            default="pause6-constitution" minOccurs="0" />
15        <element name="index"         type="c6:project-name"
16            default="pause6-index"    minOccurs="0" />
17    </sequence>
18 </complexType>

```

board

```

1 <complexType name="board">
2     <sequence>
3         <element name="member" type="p6:identity"
4             minOccurs="1" maxOccurs="unbounded" />
5         <element name="minimum-members" type="int"
6             minExclusive="0" default="1" />
7     </sequence>
8 </complexType>

```

B.7 File repository.xml

The deployers and commissioner may have different plans how to keep the files. The file may also be compressed in different ways. Therefore, the file locations are not signed themselves, but only the uncompressed content.

repository

Each released file may appear more than once, but at least once. It may be so that the file's original name (in path) can be recognized from the location URI, but it may not be.

```

1 <element name="repository" type="ff:repository" />
2 <complexType name="repository">
3   <choice minOccurs="0" maxOccurs="unbounded">
4     <element name="file" type="ff:stored-file" />
5   </choice>
6 </complexType>
7

```

stored-file

```

1 <complexType name="stored-file">
2   <sequence>
3     <element name="path" type="c6:fs-path" />
4     <element name="compression" type="c6:compression-type"
5       default="none" />
6     <element name="encoding" type="c6:encoding-type"
7       default="none" />
8     <element name="location" type="anyURI" />
9   </sequence>
10 </complexType>

```

C Schema: Pause6 messages

The schema defined in this section defines messages types used for Pause6. They extend the basic types of CPAN6.

C.1 Schema wrapper

```

1 <schema
2   xmlns="http://www.w3.org/2001/XMLSchema"
3   targetNamespace="http://cpan6.net/200609/pause6-messages"
4   schemaLocation="https://xml.cpan6.net/schema/200609/pause6-messages.xsd"
5   version="1.0"
6
7   xmlns:p6="http://cpan6.net/200609/pause6-basic"
8   xmlns:c6="http://cpan6.net/200609/cpan6-basic"
9   xmlns:msg="http://cpan6.net/200609/cpan6-messages"
10 >
11
12 <import
13   namespace="http://cpan6.net/200609/pause6-basic"
14   location="https://xml.cpan6.net/schema/200609/pause6-basic.xsd" />
15
16 <import
17   namespace="http://cpan6.net/200609/cpan6-basic"
18   location="https://xml.cpan6.net/schema/200609/cpan6-basic.xsd" />

```

```

19
20     <import
21         namespace="http://cpan6.net/200609/cpan6-messages"
22         location="https://xml.cpan6.net/schema/200609/cpan6-messages.xsd" />

```

This messages asks a list all releases, only supporting simple filtering. It is used by scribes to make an inventory of all releases to be copied. The data is directly taken from the archive-log.

This message is not meant for normal users. They should use the search interface to get a much smaller set of answers, probably optimized with database searches.

messageListReleasesRequest

The scribe can request only releases with certain states. By default, only released and deprecated releases are returned. With the `only-last` flag set, only the last version of each project is returned.

```

1  <element name="messageListReleasesRequest"
2      substitutionGroup="msg:request">
3      <complexContent>
4          <extension base="msg:messageListReleasesRequest">
5              <sequence>
6                  <element name="states" type="p6:release-state"
7                      minOccurs="0" maxOccurs="unbounded" />
8                  <element name="types" type="p6:project-type"
9                      minOccurs="0" maxOccurs="unbounded" />
10             </sequence>
11         </extension>
12     </complexContent>
13 </element>

```

messageListReleasesAnswer

```

1  <element name="messageListProjectsAnswer"
2      substitutionGroup="msg:answer">
3      <complexContent>
4          <extension base="msg:messageListProjectsAnswer">
5              <sequence>
6                  <element name="release" type="p6:visible-release-change"
7                      minOccurs="0" maxOccurs="unbounded" />
8              </sequence>
9          </extension>
10     </complexContent>
11 </element>

```

C.2 Accept signature

messageAcceptSignatureRequest

```

1 <element name="messageAcceptSignatureRequest "
2   substitutionGroup="msg:request">
3   <sequence>
4     <element name="release" type="p6:release-label" />
5     <element name="signature" type="p6:signature" />
6     <element name="status-seqnr" type="positiveInteger" />
7   </sequence>
8 </element>

```

messageAcceptSignatureAnswer

```

1 <element name="messageAcceptSignatureAnswer "
2   substitutionGroup="msg:answer">
3   <sequence>
4     <element name="votes-received" type="p6:vote-count" />
5     <element name="votes-required" type="p6:vote-count" />
6   </sequence>
7 </element>

```

C.3 Get release status

messageGetReleaseStatusRequest

```

1 <element name="messageGetReleaseStatusRequest "
2   substitutionGroup="msg:request">
3   <sequence>
4     <element name="release" type="p6:release-label" />
5     <element name="which" type="ff:release-state-which" />
6   </sequence>
7 </element>

```

messageGetReleaseStatusAnswer

```

1 <element name="messageGetReleaseStatusAnswer "
2   substitutionGroup="msg:answer">
3   <sequence>
4     <element name="status" type="p6:release-status-description" />
5     <element name="votes-received" type="p6:vote-count" />
6     <element name="votes-required" type="p6:vote-count" />
7   </sequence>
8 </element>

```

References

- [1] Overmeer and Vilain, *CPAN6 and Pause6 Design*.
- [2] Overmeer and Vilain, *CPAN6 Implementation*.

Index

admin-rights, 21
archive-definition, 27
archive-hint, 28
archive-list-entry, 27
archive-log-entry, 26

blocked, 18
board, 29

checksum, 23
checksum-type, 23
constitution, 28

deprecate, 19

embargo-time, 18
expiration, 18

has-followup, 19

identity, 20
initiated, 24

messageAcceptSignatureAnswer, 32
messageAcceptSignatureRequest, 31
messageGetReleaseStatusAnswer, 32
messageGetReleaseStatusRequest, 32
messageListReleasesAnswer, 31
messageListReleasesRequest, 31
meta-data, 20

name-space-layout, 29

permission-class, 21
permission-set, 21

release, 15
release-component, 16
release-log-entry, 25
release-plan, 17
release-state, 17
release-state-change, 16
release-state-which, 27
release-status-description, 28
repository, 29
required-signatures, 22
signature, 22
signature-type, 22
signatures-count, 22
state-change-confirmed, 26
state-change-request, 25
status-to-sign, 28
stored-file, 30

trust, 24

visible-release-change, 26