CPAN6 Implementation

Mark overmeer*

July 28, 2009

Abstract

This document contains implementation details for CPAN6, such as file formats and use cases. These details do change rapidly, and will continue to do so until the implementation is realized.

Contents

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

Introduction

The CPAN6/Pause6 project is described in a few different papers. This paper describes file formats and use cases specific to the CPAN6 implementation. This information is useless without the terminology definitions from paper [1] ("CPAN6 and Pause6 Design"). Please use the latest information you can find.

This paper only defines the archive distribution network, which does not reach to the internals of an actual archive itself. The Pause6 archiver is one of the feasible implementations for that, and described in paper [2] ("Pause6 Implementation").

Syntax

It is important to note that the examples in the first part of this paper are informal, and may contain many mistakes. Better examples will be included when the implementation gets started. The XML specification in the appendices is leading.

A Schema: CPAN6 Stable

This schema is relatively small and shall **never change** in any incompatible way. It is designed to assure any *unforeseen* upgrade path. Therefore, it does not have a version number in its namespace.

CPAN6 daemons handle tasks of different origin. It is very well possible that these origins use different versions of CPAN6 based software. They, however, will always use the same `cpan6-stable` specification.

Tasks which run in a daemon may convert one specification in an other. The abstraction level of CPAN6-stable does not care about these interpretations, but only the routing of the requests and answers.

A.1 Schema wrapper

The schema's prelude defines the used name-spaces.

```
1  <schema
2    xmlns="http://www.w3.org/2001/XMLSchema"
3    elementFormDefault="qualified"
4
5    targetNamespace="http://cpan6.net/cpan6-stable"
6    location="https://xml.cpan6.net/schema/cpan6-stable.xsd"
7    version="1.0"
8
9    xmlns:c6s="http://cpan6.net/cpan6-stable"
10 >
```

A.2 Message base

frame

The `frame` is used to wrap each communicated message, for requests, answers and errors.

Compared to SOAP, these CPAN6 messages

- are allways `document style`, which means: allways well defined;
- can contain multiple requests (SOAP body objects) and answers in one data exchange;
- can use various protocol versions within the same message, because various tasks of the server may use different versions of the protocol.

Of course, this type also support a lot of features which are provided by XMPP. Again, the main difference is that different revisions of the same specifications can co-exist.

```
<element name="frame">
  <complexType>
    <sequence>
      <element ref="c6s:header"  minOccurs="0" />
      <element ref="c6s:message"
          minOccurs="0" maxOccurs="unbounded" />
    </sequence>
    <attribute name="nr" type="int" />
  </complexType>
</element>
```

header

Both sides of the communication can tell the other side what it can do. Before you can send any descend payload with the message, the receiver must tell you what it accepts from the sender. This can be differ per component.

The initial message in any connection requires the header to be included. Follow-up messages may contain the header, which will change the settings of the connection. However, this may cause the interaction to fail or redirected to an other server.

```
<element name="header">
  <complexType>
    <sequence>
      <element name="accept" type="c6s:transport-accept"
        minOccurs="0" />
      <any processContents="lax"
        minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>
</element>
```

message

Messages are from one task to an other task in the network. The message contains one or more requests, answers, and/or errors, which all must extend `c6:message-part` elements.

It may very well possible that some server does not understand some of these message parts, because it does not have access to the schema's. For that reason, we simply ask for `any` elements.

```
<element name="message" type="c6s:message" />
<complexType name="message">
  <sequence>
    <element ref="c6s:route" />
    <any processContents="lax"
       minOccurs="0" maxOccurs="unbounded" />
  </sequence>
```

```
    <attributeGroup ref="c6s:order" />
  </complexType>
```

order

Requests may get followed by one or more answers from the other side, or maybe even more requests. Each request and answer has an obligatory sequence label which must be unique during a connection. Relabeling (renumbering) *will* usually take place in any service task which reroutes the requests and answers.

In most cases, it is most practical to use an auto-incrementing number for this. How requests and answers interact is not defined on this abstraction layer.

A party may flag `last` to indicate that it will discontinue the conversation. The intermediate daemons will cleanup their administration.

SOAP defines a ”notification”, which is a server initiated message to a client. In CPAN6, we communicate between two (probably unevenly sized) daemons with comparible status: there is no client nor server. Therefore, the SOAP server initiated notification is simply a request without answer in CPAN6. Read this as ”I request you to accept this information”. If the other parties server implementation answers with an error (f.i: ”I do not understand this”), then the sender should refrain from repeating these notifications.

```
  <attributeGroup name="order">
    <attribute name="seq"     type="ID" use="required" />
    <attribute name="follows" type="string" />
    <attribute name="last"    type="boolean" default="false" />
  </attributeGroup>
```

route

Explains how to reach the destination. The services which are addressed can either be a server or a task on a server. Any step may result in redirects which change the list.

```
  <element name="route">
    <complexType>
      <sequence>
        <element name="from" type="c6s:service-id"
           minOccurs="0" />
        <element name="via"  type="c6s:service-id"
           minOccurs="0" maxOccurs="unbounded"     />
        <element name="to"   type="c6s:service-id" />
      </sequence>
    </complexType>
  </element>
```

service-id

Any functionality in the CPAN6 network has its abstract address, which is used as namespace. With the full intention to be a location independent URN (Universal Resource Name, RFC2141 and RFC3986).

People should use the following URN structure:

```
urn:cpan6:{\em domain}:{\em task}:{\em name}
```

The `domain` is a registered domain name, owned by the founders of the CPAN6 functionality or a subdomain of that domain. Tasks are desciptive like `archive` or `proxy`. without hint on how is has been implemented. Example:

```
urn:cpan6:cpan.org:archive:perl5-modules
```

```
1  <simpleType name="service-id">
2    <restriction base="anyURI">
3      <pattern value="urn:cpan6:*" />
4    </restriction>
5  </simpleType>
```

A.3 Negotiation

transport-accept

Explains how the content –the payload, mainly the releases– may be processed. Like the `Accept-*` headers of MIME. Both sides of the communication line know what the other side can do.

```
1  <complexType name="transport-accept">
2    <sequence>
3      <element ref="c6s:algorithms"
4        minOccurs="0" maxOccurs="unbounded"/>
5      <any processContents="lax"
6        minOccurs="0" maxOccurs="unbounded"/>
7    </sequence>
8  </complexType>
```

algorithms

Algorithms refer to all kinds of pluggable components. The client tells the server in the initial message after the connection which algorithms it supports, and the server will pick from that list what it likes best.

We do not want to change the transport schema ever, so although we know beforehand which algorithm classes we can expect, we will not hard-code them in the schema.

Only the following official algorithm classes are currently permitted: `authentication`, `compression`, `data-format`, `encoding`, `encryption`, `language`, `packaging` and `protocol`. The interpretation of these fields is defined in the CPAN6 "basic types" chapter, and may change (slightly) over time.

The `c6s:algorithm` type is not to be used on higher levels of the specification: once the initiator and the server applications have negotiated their way of interaction, much better defined types will be used.

We do want to list our features from the beginning for following reasons:

1. the server may never be able to serve us, which we need to detect on connection time;
2. based on these facts, the server may decide that you are better helped by an other server. Redirections can best to happen as early as possible for performance reasons;
3. you can use this information to optimize the messages which is being exchanged, to optimize the message content. For instance, you do only need to send error messages back in the language the user understands.
4. the information may be required to transport data, and it is costly for the server to get back to the client to ask for these details. For instance, which forms of compression are supported.

MIME defines header field which start with `Accept-` for the same purpose. So, when CPAN6 messages are exchanged via HTTP or email, we can use those fields. However, CPAN6 may be used over other kinds of connections (like local file access) which require the same facts.

```
<element name="algorithms">
  <complexType>
    <sequence>
      <element name="can" type="c6s:algorithm"
        minOccurs="0" maxOccurs="unbounded" />
      <any processContents="lax"
        minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>
</element>
```

algorithm
: Algorithms are indicated by a token. Basically, those tokens are like mime-types. However, because close to none of the required tokens are defined by IANA, it seems useless to use real mime-types.

```
<complexType name="algorithm">
  <simpleContent>
    <extension base="NMTOKENS">
    <attribute name="class" type="token" use="required"/>
   </extension>
  </simpleContent>
</complexType>
```

B Schema: CPAN6 Basic types

The schema defined in this section defines basic types used for CPAN6.

B.1 Schema wrapper

The schema's prelude defines the used name-spaces.

```
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"

  targetNamespace="http://cpan6.net/2008/cpan6-basic"
  location="https://xml.cpan6.net/schema/2008/cpan6-basic.xsd"
  version="1.0"

  xmlns:c6s="http://cpan6.net/cpan6-stable"
  xmlns:c6="http://cpan6.net/2008/cpan6-basic"
>

  <import
    namespace="http://cpan6.net/cpan6-stable"
    schemaLocation="https://xml.cpan6.net/schema/cpan6-stable.xsd" />
```

B.2 Generic types

algo-data
: Any kind of algorithm (see the extensions of `p6:algorithm`, like checksum and authorization) may add information to data-structures.

The structure of that data which needs to be transported can be quite different, and therewith would complicate the schemas enormously. In this type, these algorithms can enclose their serialized data in this structure, with some basic serialization attributes provided.

```
1  <complexType name="algo-data">
2    <simpleContent>
3      <extension base="string">
4        <attribute name="algorithm" type="token" use="required" />
5        <attribute name="encoding"  type="token" />
6        <attribute name="compress"  type="token" />
7      </extension>
8    </simpleContent>
9  </complexType>
```

algo-param

Simply a key-value pair, only to be used within a `algo-params` structure.

```
1  <complexType name="algo-param">
2    <simpleContent>
3      <restriction base="string">
4        <attribute name="name" type="token" />
5      </restriction>
6    </simpleContent>
7  </complexType>
```

algo-params

List of parameters to be used with the an algorithm.

```
1  <complexType name="algo-params">
2    <sequence>
3      <element name="param" type="c6:algo-param"
4        minOccurs="0" maxOccurs="unbounded" />
5    </sequence>
6    <attribute name="algorithm" type="token" use="required" />
7  </complexType>
```

error-code

The error codes will use the same numbers defined by HTTP, whenever possible. Additional codes can be picked above `1000`.

```
1  <complexType name="error-code" type="int" />
```

help

This content targets the end-user, providing hints on what went wrong or what is happening.

Texts may be provided in different lengths, for instance a short version and a detailed long explanation. The client can decide which suites best, dependent on the user's choice of verbosity. The texts will be limited to the negotiated languages.

The `web` uri can point to a web-page (with fragment to point within that page) which contains a more detailed explanation that provided in the text messages. The texts should not contain examples and reasoning, but the web-page can help the user in full extend.

```
<element name="help" type="c6:help" />

<complexType name="help">
  <choice maxOccurs="unbounded">
    <element name="text" type="c6:lang-string" />
    <element name="web"  type="c6:lang-uri"    />
  </choice>
</complexType>
```

label

A `label` is used wherever a string is required to give a component a name. All labels can contain unicode. Empty labels are not permitted. Archives may restrict the labels to certain patterns.

White-spaces are collapsed, which means that all series of visual white-spaces are replaced by one blank (`\x020`). Blanks are removed from the beginning and the end of the string.

```
<simpleType name="label">
  <restriction base="string">
    <minLength  value="1" />
    <whiteSpace value="collapse" />
  </restriction>
</simpleType>
```

label-map

CPAN6 defines most names, like filenames of released items, as `c6:label`, which is a non-empty unlimited length unicode string. However, during transport, with packaging or in storage, there will always be some limitiations. In such case, the `label-map` will express the mutilations made.

Applications shall always attempt to minimalize the mutilations, keeping the size of these maps as small as possible. A separate map must be made for each (small) logical group of names, where splitting list of maps during processing must be avoided: we do not want to inspect the content of the messages to be able to do those tasks, and we must avoid collissions between sets of names.

```
<element name="label-map">
  <complexType>
    <sequence>
      <element name="map" minOccurs="0" maxOccurs="unbounded">
        <simpleType>
          <extension base="string" />
          <attribute name="label" type="c6:label" />
        </simpleType>
      </element>
    </sequence>
  </complexType>
</element>
```

lang-string

A string targeted for the end user, and therefore with a required language attribute.

```
<complexType name="lang-string">
  <simpleContent>
    <restriction base="string">
```

```
      <attribute name="lang" type="language" default="en" />
    </restriction>
  </simpleContent>
</complexType>
```

lang-uri

A URI targeted for the end user, and therefore with a required language attribute.

```
<complexType name="lang-uri">
  <simpleContent>
    <restriction base="anyURI">
      <attribute name="lang" type="language" default="en" />
    </restriction>
  </simpleContent>
</complexType>
```

meta-data

Collection of item information as collected by end-user applications. The exact content of these fields may not always be standardizable.

Applications may feel the need to add information per item, which is not yet is this basic set. These fields may be added to the next version of the CPAN6 specification. However, care must be taken not to put the whole world of information in this low-lever `meta-data` type: checksums and file version information is not included because these are less than core features. These may get standardized in a separate namespace.

We may very well need to parse the meta-data information without the knowledge of all additional schemas. Therefore, the `any` is added. If we would use a base element of the `meta-data-more` in this schema, then the application would break over missing substitutionGroup definitions.

```
<element name="meta-data" type="c6:meta-data" />

<complexType name="meta-data">
  <sequence>
    <element name="mime-type"   type="c6:mime-type"       minOccurs="0" />
    <element name="size"        type="nonNegativeInteger" minOccurs="0" />
    <element name="executable"  type="boolean"            default="false" /
    <element name="newline"     type="c6:newline"         default="\n"    /

    <element name="created"     type="dateTime"           minOccurs="0" />
    <element name="modified"    type="dateTime"           minOccurs="0" />

    <choice minOccurs="0" maxOccurs="unbounded">
      <element name="abstract"    type="c6:lang-string" />
      <element name="description" type="c6:lang-string" />
      <element name="keywords"    type="c6:lang-string" />
      <element name="more"        type="c6:lang-uri"    />
    </choice>

    <element ref="c6:checksum"                            minOccurs="0" />

    <any processContent="lax" minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>

```

```
<complexType name="more-meta-data" abstract="true">
  <sequence />
</complexType>
```

mime-type

The official mime-type definition is far more complex, but for now we will stick to the string super-class.

```
<simpleType name="mime-type">
  <restriction base="string" />
</simpleType>
```

message-part

Base element for all ”payload” constructions in messages.

```
<element name="message-part" type="c6:message-part"
   abstract="true" />

<complexType name="message-part">
  <complexContent>
    <sequence>
      <element ref="c6:help" />
    </sequence>
    <attribute name="protocol" type="string" use="required" />
  </complexContent>
</complexType>
```

B.3 Encapsulation

Encapsulation is needed to transport or store information where the mechanisms comflict with the transported data.

encapsulated-release

When releases are being transported, a whole directory tree must move between two servers.

Elements may be transported separately or packaged. A mixture is possible as well, for instance small files packaged and the big files as separate elements. This way, it may be easier to recover from network or other service problems.

The `transport` element is used as default transport mechanism for each encapsultated item. For each item, you may select a different way to collect the data. Reordering of the items may be needed to use the transport connections optimally.

```
<complexType name="encapsulated-release">
  <sequence>
    <element name="release"     type="c6:release-id"   />
    <element name="total-size"  type="positiveInteger" />
    <element name="item"        type="c6:encapsulated-item"
       minOccurs="0" maxOccurs="unbounded" />
    <element name="transport"   type="c6:transport" minOccurs="0" />
  </sequence>
</complexType>
```

encapsulated-item

Used to express details about transported items. The item is first compressed, and then

encoded. The size is the resulting size. The path contains the original item-name, the location and actual storage location. may (probably will) reflect the applied algorithms.

Both packaging and transport mechanism may restrict the unlimited size, case-sensitivity, and unicode character space of items. Therefore, a map can be included to correct the conflicts. For each path name which does need a work-around, the map can be left-out.

```
<complexType name="encapsulated-item">
  <complexContent>
    <sequence>
      <element name="path"        type="c6:label"        default="/"    />
      <element name="size"        type="positiveInteger" default="0"    />
      <element ref="c6:storage"   />
      <element name="path-map"    type="c6:label-map"    minOccurs="0"  />
    </sequence>
    <attribute name="compression" type="c6:compression"  default="none" />
    <attribute name="encoding"    type="c6:encoding"     default="none" />
    <attribute name="packaging"   type="c6:packaging"    default="none" />
  </complexContent>
</complexType>
```

storage

Abstract base-class to contain storage specifications.

```
<element name="storage" abstract="true" />
```

inlined-data

The data is inlined in XML message, which may be very useful for very short data. The data must be encoded with something the receiver understands, for instance base64, reflected in the `encoded` element.

```
<element name="inlined-data" type="string"
   substitutionGroup="c6:storage" />
```

attached

Only available when the transport mechanism supports some kind of data labelling, like mime-headers. The label may change between transporter connections. The content of the label may have restrictions imposed by the transport mechanism.

```
<element name="attached" type="c6:label"
   substitutionGroup="c6:storage" />
```

download

Open a separate connection to a different service to fetch the data. By default, the transporter of some encapsulating xml element will be used, for instance the `transport` element of the `encapsulated-release`.

```
<element name="download" substitutionGroup="c6:storage">
  <complexType>
    <complexContent>
      <sequence>
        <element name="recursive" type="boolean" default="true" />
        <element ref="c6:transport" />
      </sequence>
    </complexContent>
  </complexType>
</element>
```

sum

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 SHA-1 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 checksumed by different algorithms, which usually adds trust.

The checksum type is defined as `token`. Predefined types include `MD5`, `SHA1`, and `SHA128`. The software should ignore checksums it does not know of, and check *all* checksums it understands, unless it knows which one is best: in that case only check the best for reasons of performance.

```
<complexType name="sum">
  <complexContent>
    <extension base="c6:algorithm">
      <sequence>
        <element name="sum" type="string"/>
      </sequence>
      <attribute name="encoding" type="token" use="required" />
    </extension>
  </complexContent>
</complexType>
```

B.4 Transport protocol

transport

Base-type for transport protocol configuration. Each transport protocol may require an authentication, and may have alternatives for it.

The extension to this type all instruct a remove process how to contact something to drop data or ask questions. See `c6d:interface` to for the daemon counterpart.

```
<element name="transport" type="c6:transport" abstract="true" />
<complexType name="transport">
  <sequence />
</complexType>
```

transport-disk

Specifies how to use a local disk. The authentication mechanism of the system itself is used.

```
<element name="transport-disk" substitutionGroup="c6:transport">
  <complexType>
    <complexContent>
      <extension base="c6:transport">
        <sequence>
          <element name="directory" type="c6:directory" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</element>
```

transport-ftp

Specifies how to use ftp. An `authentication-basic` for the host or domain which is addressed can be provided, of course for service `ftp`. The default username and password are `anonymous`.

When secure ftp (`sftp`) is needed, then provide sufficient authenication elements for that. The client should prefer authenticated downloads over anonymous ftp.

```
<element name="transport-ftp" type="c6:transport-ftp"
  substitutionGroup="c6:transport" />

<complexType name="transport-ftp">
  <complexContent>
    <extension base="c6:transport">
      <sequence>
        <element name="hostname" type="string" default="localhost" />
        <element name="port"     type="int"    default="21" />
        <element name="root"     type="string" default="/" />
      </sequence>
    </extension>
  </complexContent>
</complexType>

```

transport-http

Specifies how to use http. Secure http (`https`), basic-authenication and cookies (`authentication-known-sec` may be needed.

```
<element name="transport-ftp" type="c6:transport-ftp"
  substitutionGroup="c6:transport" />

<complexType name="transport-ftp">
  <complexContent>
    <extension base="c6:transport">
      <sequence>
        <element name="hostname"  type="string" default="localhost" />
        <element name="port"      type="int"    default="21" />
        <element name="directory" type="string" default="/" />
      </sequence>
    </extension>
  </complexContent>
</complexType>

```

transport-limits

Explains what the limitations of the communication are. For instance, the number of requests per connection, the number of bytes per connection, and so forth.

Limitations are to be enforced by the receiving party (the daemon who publishes these limitations), and may improve the communication when the sending party understands them. So, the simplest clients may ignore this info and simply trust on their error handling.

When no `service-id` is provided with `for`, then these limits are set for all services on that server. You may also set limits on one or more specific services.

When limits items are specified for a service, these prevail over the default server-wide settings. When some limit item is missing, then the server-wide setting will be used. In other words: the settings for a specific server extend or overrule the global defaults.

```
<complexType name="transport-limits">
  <sequence>
    <element name="limit" type="c6s:limit"
       minOccurs="0" maxOccurs="unbounded"/>
    <any processContents="lax"
       minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

limit

Report a limitiation to the other party. Do not include limitiations of the task (like information about the archiver) but only which have to do with the transport of data. These limitations are reported informational only: they may not be enforced.

Limitiations can be specified with

min : minimal value, often used with numerical values for comparison, but may use additional syntax;

max : maximum value;

only : other values are not acceptable;

pick : a comma separated list of values where to pick one or more from;

per : tells when to restart measuring

Initially, the limitation names are

conn-time : timeout for connections by this user;

connections : number of connections permited, usually per time-interfal;

msgs-per-conn : number of messages exchanged per connection;

exch-per-conn : number of requests and answers exchanged per connection;

exch-per-msg : number of exchanges request and answer components per message;

msg-length : the maximum size of the uncompressed XML messages. This needs to be used to determine the number of requests and answers can be packaged in one message, and how much data inlining can take place;

bytes-send-per-conn : number of bytes which can be sent per connection;

bytes-get-per-conn : number of bytes which will be received per connection;

bytes-send : bytes sent per timespan;

bytes-get : bytes received per timespan;

You should not add limitations like downloads, uploads or search-results, because they depend on the component which gets addressed. Those are application specific.

```
<complexType name="limit">
  <complexContent>
    <sequence>
      <attribute name="name" value="token"    use="required"/>
      <attribute name="min"  value="string"   />
      <attribute name="max"  value="string"   />
      <attribute name="only" value="string"   />
```

```
        <attribute name="pick" value="string"   />
        <attribute name="per"  value="duration" />
      </sequence>
    </complexContent>
  </complexType>
```

B.5 Authorization

Authorization and authentication is handled by the archive implementations (like Pause6), but to provide a generic interface to various of these implementations, the definition of the required structures is on CPAN6 level.

authorization

Any set of authorization mechanisms can be specified at once. Only one of them has to succeed (under normal circumstances) to establish contact.

```
<complexType name="authorization">
  <complexContent>
    <sequence>
      <element name="authentication" type="c6:authentication"
         minOccurs="0" maxOccurs="unbounded" />
    </sequence>
    <attribute name="use" type="xs:duration" default="0D" />
  </complexContent>
</complexType>
```

authentication

When shared secret authentication is used, it needs to listed in the private identities of both sender and receiver.

The `realm` is used to relate public and private authentication components. The `destination` limits the application of the information; if any of the destinations match, the authentication method can be used.

```
<complexType name="authentication">
  <complexContent>
    <extension base="c6:algo-params">
      <sequence>
        <element name="destination" type="anyURI"
          minOccurs="0" maxOccurs="unbounded" />
      </sequence>
      <attribute name="realm" type="ID" use="required" />
    </extension>
  </complexContent>
</complexType>
```

authentication-basic

HTTP basic authenication, but *realm* is not needed be specified explicitly: it's the addressed archive. This type can also be used to specify a PAUSE-ID with password.

```
<element name="authentication-basic" type="c6:authentication-basic"
  substitutionGroup="c6:authentication" />

<complexType name="authentication-basic">
  <complexContent>
```

```
      <extension base="c6:authentication">
        <sequence>
          <element name="username" type="string" />
          <element name="password" type="string" />
          <element name="password-encoding" type="c6:encoding" default="none"
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

authentication-known-secret

Any known secret, for instance a private key. Of course, this string may be passed over network connections, so please be sure that the connection itself is *safe* in these situations. The available types need to be defined.

```
<element name="authentication-known-secret"
  substitutionGroup="c6:authentication">
  <complexType>
    <complexContent>
      <extension base="c6:authentication-type">
        <sequence>
          <element name="secret"   type="string" />
          <element name="type"     type="token" default="any" />
          <element name="encoding" type="token" default="base64" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</element>
```

B.6 Release meta

release

Combines all release related meta-data from the publishers point of view. The source address is fixed to the archive the originating archive. In other archives, this release may have addresses as well.

```
<element name="release" type="c6:release" />
<complexType name="release">
  <sequence>
    <element name="id"        type="c6:release-id"        />
    <element name="meta-data" type="c6:release-meta-data" />
    <element name="payload"   type="c6:release-payload"   />
    <element ref="c6:release-extension"
      minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

release-id

This points to one release in some archive, or defines one. The archive is specified by the address of the commissioner, on the moment that the release was entered. The three fields together shall be world-wide uniquely identify the release.

The `source` must be a unique identifier for an archive, which could be the uri for a web-interface which interacts with the archive. Just like the namespace of an XML

schema, which usually points to the location where you can get the schema. Applications shall not trust the source-name-version combination to be authentic, but always include a checksum on the index file in the comparison.

```
<complexType name="release-id">
  <sequence>
    <element name="source"  type="c6s:service-id"       />
    <element name="name"    type="c6:label"             />
    <element name="version" type="c6:label" default="0" />
  </sequence>
</complexType>
```

release-payload

Collects the groups of items which are released.

```
<complexType name="release-payload">
  <sequence>
    <element name="item-group" type="c6:item-group"
      minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

release-meta-data

Application dependent information about a release. This type must be extended to be useable. Shared needs by application, but without a clear definition or quality should be added in this type hierarchy level.

```
<element name="release-meta-data" type="c6:release-meta-data"
  abstract="true" substitutionGroup="c6:meta-data" />

<complexType name="release-meta-data">
  <complexContent>
    <extension base="c6:meta-data" />
  </complexContent>
</complexType>
```

release-extension

Any CPAN6 extension implementation will define extra attributes to a release. These extra attributes will be transported, even between different implementations, which may be able to convert it.

Even when available, parties may decide not to send any extension meta-data. They may also maintain meta-data in more than one separate extension.

```
<element name="release-extension" type="c6:release-extension"
   abstract="true" />

<complexType name="release-extension">
  <sequence />
</complexType>
```

B.7 Release content

directory

Marks the existence of a directory.

```
<complexType name="directory">
  <complexContent>
    <extension base="c6:item-released"/>
  </complexContent>
</complexType>
```

direntry

Any directory element which is not a file or symbolic link. The mime-type must help us out, in this case.

```
<complexType name="direntry">
  <complexContent>
    <extension base="c6:item-released"/>
  </complexContent>
</complexType>
```

file

A file which is released: a sequence of bytes with a name. The size of the file is the uncompressed size: the repository may contain compressed versions of this file.

```
<complexType name="file">
  <complexContent>
    <extension base="c6:item-released">
      <sequence>
        <element name="checksum" type="c6:algo-data"
          minOccurs="0" maxOccurs="unbounded" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

file-inlined

A file which is smaller in size than its combined checksums can better be included directly in the index.

```
<complexType name="file-inlined">
  <complexContent>
    <extension base="c6:item-released">
      <sequence>
        <element name="encoding" type="string" />
        <element name="content"  type="string" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

item-group

A set of released items. The attributes can be used to define subsets, which may be used to selectively load parts of releases.

The `lang` specifies the language in which the files are written. This can be used to, for instance, provide a README in different languages. The `class` works like CSS DOM class selectors.

```
<complexType name="item-group">
  <choice minOccurs="0" maxOccurs="unbounded">
```

```
      <element name="file"         type="c6:file" />
      <element name="directory"    type="c6:directory" />
      <element name="symlink"      type="c6:symlink" />
      <element name="file-inlined" type="c6:file-inlined" />
      <element name="direntry"     type="c6:direntry" />
    </choice>
    <attribute name="lang"  type="language" />
    <attribute name="class" type="NMTOKENS" />
  </complexType>
```

item-released

Some element of a release. This defines the substitutionGroup for elements `file`, `symlink`, and `direnty`.

The C¡path¿ is the name of the item as found on the source system. This may be very incompatible with the platform where it will be used, so it is considered to be a `label`.

Be aware that you cannot limit the path names used in the distribution (because you do not know about all existing operating systems) Therefore, `Pause6::Store` objects needs to be called for help: it will translate path labels used within the release into save filenames on local disk.

The `mime-type` is used to distinguish between the types of items. One item may have different revision strings attached to it, for instance, Perl5 defines a public `$VERSION` variable for each file and some people use CVS or SVK versions for the same file.

```
<complexType name="item-released">
  <sequence>
    <element name="meta-data" type="c6:meta-data" minOccurs="0" />
    <element name="revision"  type="c6:algo-data"
      minOccurs="0" maxOccurs="unbounded" />
  </sequence>
  <attribute name="path"      type="c6:label" use="required"/>
</complexType>
```

item-revision

Defines a revision/version for an item, unrelated to the CPAN6/Pause6 administration. The `application` contains strings like `CVS`, `SVK`, or `Perl5`, indicating the versioning scheme used.

Perl5 installation tools will need to be able to search these revisions, because Perl5 dependencies are based on versions of seperate files within a released distribution.

```
<element name="item-revision">
  <complexType>
    <simpleContent>
      <extension base="string">
        <attribute name="application" type="string" use="required" />
      </extension>
    </simpleContent>
  </complexType>
</element>
```

newline

Various operating systems have different ideas about how to encode the end of a line. This variable also shows whether a file is `binary`. The `unknown` is possible when the file is empty or the line not terminated with a carriage return or line feed.

```
<simpleType name="newline">
  <restriction base="token">
    <enumeration value="lf"      />
    <enumeration value="crlf"    />
    <enumeration value="cr"      />
    <enumeration value="binary"  />
    <enumeration value="empty"   />
    <enumeration value="unknown" />
  </restriction>
</simpleType>
```

symlink
A symbolic link which is released. Symbolic links are not supported on all platforms.

```
<complexType name="symlink">
  <complexContent>
    <extension base="c6:item-released">
      <sequence>
        <element name="to"   type="c6:label" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

C Schema: CPAN6 Daemon types

The schema described in this section defines very common needs for the implementation of the daemon and for tasks. The use of this schema is optional, however, the data-structures match that of some objects created by `CPAN6::Daemon`

C.1 Schema wrapper

The schema's prelude defines the used name-spaces.

```
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"

  targetNamespace="http://cpan6.net/2008/cpan6-daemon"
  location="https://xml.cpan6.net/schema/2008/cpan6-daemon.xsd"
  version="1.0"

  xmlns:c6s="http://cpan6.net/cpan6-stable"
  xmlns:c6="http://cpan6.net/2008/cpan6-basic"
  xmlns:c6d="http://cpan6.net/2008/cpan6-daemon"
>

  <import
    namespace="http://cpan6.net/cpan6-stable"
    schemaLocation="https://xml.cpan6.net/schema/cpan6-stable.xsd" />

  <import
    namespace="http://cpan6.net/cpan6-basic"
    schemaLocation="https://xml.cpan6.net/schema/cpan6-basic.xsd" />
```

C.2 Server

General server configuration needs. Used by `CPAN6::Daemon::Default`.

server

The configuration structure for a server. A file with this content can be passed as parameter to the `cpan6d` server start-up script, but there are other ways to use it.

```
<element name="server" type="c6d:server" />

<complexType name="server">
  <sequence>
    <element ref="c6d:interface" maxOccurs="unbounded" />
  </sequence>
</complexType>

```

task

Describes a task run by the server.

```
<element name="task" type="c6t:task" />

<complexType name="task">
  <complexContent>
    <sequence>
      <element name="accept" type="c6d:preferences" />
      <element name="output" type="c6d:preferences" />
    </sequence>
    <attribute name="name" type="c6s:service-id" />
  </complexContent>
</complexType>
```

C.3 Algorithms

The `algorithms` specify the platform on which the daemon is running. Some algorithms may (also) play a role in the tasks, but in most cases, these elements define conversions.

One could expect these types to appear in the `cpan6-basic` namespace, however: this is all about freezing the configuration of algorithms, not about specifying what has been used (which is only represented by a string)

algorithm

The algorithms are used in the communication between two parties. Mime-types are used to indicate specific algorithms, which are grouped and extended per area of application (see `encoding`, `compression`, `packaging`, `checksum`, and `data-format`)

Relative qualifications for the algorithm are optional. They are used in negotiation between the parties, case there is a choice. Higher numbers for C¡quality¿ and C¡performance¿ mean 'better'. It does not matter what scale you use (for instance percentage or "1 to 5") because the numbers will be treated relatively. The C¡preference¿ value indicates order: lowest number is tried first. The application which has to provide the data will decide.

```
<complexType name="algorithm">
  <sequence>
    <element name="implementation" type="token"  />
    <element name="plugin"         type="string" />
```

```
      <element name="quality"        type="int"    minOccurs="0" />
      <element name="performance"    type="int"    minOccurs="0" />
      <element name="preference"     type="int"    />
    </sequence>
    <attribute name="algorithm"      type="token"  />
  </complexType>
```

compression

Examples of compression algorithms with their implementations are

- `bzip2` and `perl5.compress-bzip2`;
- `gzip` and `perl5.compress-gzip`; and
- `zip` and `perl5.archive-zip`, apply zip on single file basis.

```
<element name="compression">
  <complexType>
    <complexContent>
      <extension base="c6d:algorithm">
        <sequence>
          <element name="deflate" type="string"   minOccurs="0" />
          <element name="inflate" type="string"   minOccurs="0" />
          <element name="minsize" type="nonNegativeInteger" default="0" />
          <element name="maxsize" type="nonNegativeInteger" minOccurs="0" /
          <element name="extension" type="string" minOccurs="0" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</element>
```

encoding

Various fields contain encoded binary data, like signatures. This binary data is converted into ASCII text using some mapping. Some mappings produce smaller results than other mappings. On the other hand, the application may not always support the most efficient encoding.

Commonly supported encoding algorithms with their implementations are

1. `base64`, `perl5.mime-decode-base64`; and
2. `hex`, `perl5.pause6-hex`.

```
<element name="encoding">
  <complexType>
    <complexContent>
      <extension base="c6d:algorithm">
        <sequence>
          <element name="encode" type="string" minOccurs="0" />
          <element name="decode" type="string" minOccurs="0" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</element>
```

checksum

```
<element name="checksum">
  <complexType>
    <complexContent>
      <extension base="c6d:algorithm" />
    </complexContent>
  </complexType>
</element>
```

data-format

All communication may accept the data in various formats, in many different implementations.

- `perl5-script` implementation `perl5.data-dumper`;
- `perl5-storable`, implementation `perl5.storable`;
- `yaml`, YAML;
- `json`, JSON; and
- `xml`, implementation `perl5.xml-compile`

You will be able to translate one format into the other easily, based on the defined XML schemas. The only reason to use XML in this specification, is to have a formal definition of the data-structures, not because it is particulary nice.

The `charset` specifies the character-set to be used when writing the file (for some formatters). It shall be a full unicode character-set, for instance `utf8` or `utf16`. If you use other character-sets, then you have to be very careful about name restriction rules in all archives hosted on the store.

The file-name `extension` is added to simplify the connection to platforms which use those types for application binding, or editors which provide syntax highlighting.

```
<element name="data-format">
  <complexType>
    <complexContent>
      <extension base="c6d:algorithm">
        <element name="charset"   type="string" default="utf8" />
        <element name="extension" type="string" default="" />
      </extension>
    </complexContent>
  </complexType>
</element>

```

packaging

Multiple release items can get bundled together (before getting compressed). Option `single-files` states whether a release which contains only one file will be packaged as well.

Be warned that there is a difference between data which is released as package (like Perl5 distributions), and the effect of this option. The former is one single file within a release, this option is about combining all the files in a release.

Defined algorithms with their implementations are

- `tar`, `perl5.archive-tar`;
- `cpio`, `perl5.archive-tar`; and
- `zip`, `perl5.achive-zip`

Using "zip" will ignore additional compression, because it is already compressed internally. "tar" will often be combined with the "gzip" compression, creating an `.tar.gz` or `.tgz` archive, or with "bzip2" compression into a `.tar.bz2` or `.tjz`.

```
<element name="packaging">
  <complexType>
    <complexContent>
      <extension base="c6d:algorithm">
        <sequence>
          <element name="pack"   type="string" minOccurs="0" />
          <element name="unpack" type="string" minOccurs="0" />
          <element name="single-files" type="boolean" default="true" />
          <element name="extension" type="string" minOccurs="0" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</element>
```

protocol

Specifies the interaction protocol version. Per `message-part`, you may contact a different task, running a different application. Some applications may be somewhat compatible, but run different versions of the interaction protocol.

```
<element name="protocol">
  <complexType>
    <complexContent>
      <extension base="c6d:algorithm">
        <element name="name-space" type="anyURI" />
        <element name="version" type="string" />
      </extension>
    </complexContent>
  </complexType>
</element>
```

C.4 Interface

interface

Various components send-out or read-in data streams, each connection is represented by a separate interface definition. The `name` is required for logging purposes.

As `socket`, you specify an IP-address or a hostname, optionally followed by a colon and portnumber; just like the hostname/portnumber combination in a URL. For example: "`localhost:8080`".

The case-insensitive `protocol` is the service protocol. In the first implementation, only `http` is supported.

The `accept` will restrict the available algorithms which are usually auto-detected. Some (simple) daemons may not use autodetection, and hence need to configure the information sent to the other side this way.

```
<complexType name="interface">
  <complexContent>
    <sequence>
      <element name="socket" type="string" maxOccurs="unbounded" />
      <element name="proto"  type="string" />
      <element name="accept" type="c6s:transport-accept" minOccurs="0" />
      <element name="access" type="c6d:transport-access" minOccurs="0" />
    </sequence>
    <attribute name="name" type="c6:label" use="required" />
  </complexContent>
</complexType>
```

network
: Defines a set of hosts by IP or domain name, like Apache accepts in its configuration: CIDR and wildcards accepted.

```
<simpleType name="network" type="string" />
```

transport-access
: Defines which remote party to tell what about using the server. This information is needed when the command-line information from the server differs from the remote view on the configuration, for instance caused by firewalls.

```
<element name="transport-access">
  <complexType>
    <complexContent>
      <sequence>
        <element name="for" type="c6d:network"
           minOccurs="0" maxOccurs="unbounded" />
        <element ref="c6:transport" maxOccurs="unbounded" />
        <element ref="c6:transport-limits" minOccurs="0"  />
      </sequence>
    </complexContent>
  </complexType>
</element>
```

C.5 Authentication

The server side of the authentication.

D Schema: CPAN6 Messages

The schema defined in this section defines messages used for CPAN6.

D.1 Schema wrapper

CPAN6 messages only use cpan6-basic types.

```
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"

  targetNamespace="http://cpan6.net/2008/cpan6-messages"
  schemaLocation="https://xml.cpan6.net/schema/2008/cpan6-messages.xsd"
```

```
    version="1.0"

    xmlns:xml="http://www.w3.org/XML/1998/namespace"
    xmlns:c6s="http://cpan6.net/cpan6-stable"
    xmlns:c6="http://cpan6.net/2008/cpan6-basic"
    xmlns:c6m="http://cpan6.net/2008/cpan6-messages"
  >

    <import
      namespace="http://www.w3.org/XML/1998/namespace"
      schemaLocation="http://www.w3.org/2001/xml.xsd" />

    <import
      namespace="http://cpan6.net/2008/cpan6-basic"
      schemaLocation="https://xml.cpan6.net/schema/2008/cpan6-basic.xsd" />

    <import
      namespace="http://cpan6.net/cpan6-stable"
      schemaLocation="https://xml.cpan6.net/schema/cpan6-stable.xsd" />
```

D.2 General messages

error

Some problem is to be reported, which needs to be handled. We will not create a new object type for each error which may appear in the network, because there certainly are too many.

The error report is usually triggered by a request which is listed by the `follows` attribute which is available on all messages, so also the messages which contain this error part.

On the protocol level, applications may add message-parts which help the clients deal with the reported problem.

```
  <element name="error">
    <complexType>
      <complexContent>
        <extension base="c6:message-part">
          <element name="code" type="c6:error-code" />
        </extension>
      </complexContent>
    </complexType>
  </element>
```

answer-redirect

Multiple alternative locations can be specified, of which one will be used by the client, where other may fail to be accessed. The remote server may provide an additional authorization to fulfil this request. The user will add these to its own, for a limited amount of time.

When the `use` is permitted for `0D` (zero days), then it can be used only once. A next connection will retry on the original server address.

```
  <element name="answer-redirect" substitutionGroup="c6:message-part">
    <complexType>
      <complexContent>
```

```
        <extension base="c6s:message-part">
          <sequence>
            <element ref="c6s:route" maxOccurs="unbounded" />
            <element name="authorization" type="c6:authorization"
              minOccurs="0" />
          </sequence>
        </extension>
      </complexContent>
    </complexType>
  </element>
```

request-connect

Try to connect to a certain task. The `route` contains a `do` which addresses a task to connect to.

A list of `protocol` versions have to be provided, to establish the interaction protocol. The party who establishes a connection will list which protocols it understands. The other party will answer using one of those listed, or produce a protocol error.

```
<element name="request-connect">
  <complexType>
    <complexContent>
      <extension base="c6:message-part">
        <element name="protocol"  type="anyURI" maxOccurs="unbounded" />
        <element name="read-only" type="boolean" default="false" />
      </extension>
    </complexContent>
  </complexType>
</element>
```

D.3 Download release

The user (which can be a person or a scribe) asks the commissioner or deployer for download a set of files which relate to one release.

The user may exclude the download of files which it already has from previous downloads. Those files may even come from different projects or archives. For instance, the licence file may already be present. Because the use of strong checksums on the content of the file, we can know for sure that we can share the data between downloads.

download-instruction

Contains the store location of an element of a release. The user requested the address because it requires the data. This element does not contain the checksum of the file, because those need signatures and are therefore kept in the meta-data record of the release.

```
<complexType name="download-instruction">
  <sequence>
    <element name="item"  type="c6:release-component" />
    <element name="where" type="c6:store-location" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

download-release-request

Request a list of locations to get the content of released files from.

```
<element name="download-release-request"
  type="c6m:message-drr-type"
  substitutionGroup="c6m:message-request" />

<complexType name="message-drr-type">
  <complexContent>
    <extension base="c6m:message-request-type">
      <sequence>
        <element name="release" type="c6:release-label" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

download-release-answer

Returns a list of file locations. Be aware that store locations suffer from the file-system limitations imposed by the server system. Therefore, the store location may look like the released filename, but certainly doesn't have to be that in all situations.

```
<element name="download-release-answer"
  type="message-dra-type"
  substitutionGroup="c6m:answer-message" />

<complexType name="message-dra-type">
  <complexContent>
    <extension base="c6m:answer-message-type">
      <sequence>
        <element name="item" type="c6m:download-instruction"
          minOccurs="0" maxOccurs="unbounded" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

D.4 List releases

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.

list-releases-request

With the `only-last` flag set, only the last version of each project is returned, if that is supported by the archiver.

The result can also be limited to all archive changes after the specified date, probably the previous successful request for the release list.

```
<element name="list-releases-request"
  type="c6m:message-lrr-type"
  substitutionGroup="c6m:request-message" />

<complexType name="message-lrr-type">
  <complexContent>
    <extension base="c6m:request-message-type">
```

```
        <sequence>
          <element name="only-last"     type="boolean" default="true" />
          <element name="changed-after" type="date"    minOccurs="0"  />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

list-releases-answer

```
<element name="list-releases-answer"
  type="c6m:message-lra-type"
  substitutionGroup="c6m:answer-message" />

<complexType name="message-lra-type">
  <complexContent>
    <extension base="c6m:answer-message-type" />
  </complexContent>
</complexType>
```

References

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

[2] Overmeer and Vilain, *Pause6 Implementation*.

Index