

---

# Application Packaging Standard - Package Format Specification

1.1

Copyright © 1999, 2008 Parallels, Inc

All rights reserved.

## Table of Contents

1. Introduction .....	2
2. Conventions .....	4
3. Document Structure .....	4
4. Basic Package Format .....	4
4.1. File Format .....	4
4.2. Files .....	4
4.3. Metadata File .....	4
5. Metadata Descriptor .....	4
5.1. Common Application Properties .....	10
5.1.1. Package Name .....	10
5.1.2. Package Version .....	10
5.1.3. Homepage .....	10
5.1.4. Software Vendor Information .....	10
5.1.5. Software Packager Information .....	10
5.1.6. Summary .....	11
5.1.7. Description .....	11
5.1.8. Icon .....	11
5.1.9. Screenshots .....	11
5.1.10. Changelog .....	11
5.1.11. Categories .....	12
5.1.12. Languages .....	12
5.1.13. Updates .....	12
5.1.14. Content Delivery Methods .....	13
5.1.15. Global Settings .....	13
5.2. Services .....	13
5.2.1. Service Summary .....	14
5.2.2. License Agreement .....	15
5.2.3. Information Links .....	15
5.2.4. Entry Points .....	16
5.2.5. Service Settings .....	17
5.2.6. Requirements .....	21
5.2.7. Service Provision .....	22
5.3. Service Provision Methods .....	23
5.3.1. URL Mapping .....	23
5.3.2. Configuration Script .....	25
6. Points of Extensibility .....	28
6.1. Requirement Types .....	28
6.2. URL Handlers Types .....	28
6.3. Additional Files .....	28
6.4. Environment Variables .....	29
6.5. Additional Scripts .....	29
6.6. Configuration Script Language .....	29
6.7. Application Provision Method .....	29
6.8. Rules .....	29

7. Common Aspects .....	29
7.1. PHP Aspect .....	29
7.1.1. Requirement Types .....	30
7.1.2. URL Handlers .....	31
7.1.3. Configuration Script Language .....	31
7.2. ASP.NET Aspect .....	31
7.2.1. Requirement Types .....	32
7.2.2. URL Handler Type .....	32
7.2.3. Configuration Script Language .....	32
7.3. Database Aspect .....	32
7.3.1. Environment Variables .....	33
7.3.2. Database Server Types .....	34
7.3.3. Upgrade .....	34
7.3.4. MySQL Specific Settings .....	34
7.4. Apache Aspect .....	35
7.5. CGI Support .....	35
7.6. Hardware Resources .....	36
7.7. Operating Environment .....	37
7.8. Mail .....	38
7.8.1. Environment Variables .....	39
7.8.2. Upgrade .....	40
References .....	40

## 1. Introduction

Application Packaging Standard defines the packages structure and rules of processing packages. Package is a file that contains an application files and metadata required to create and manage instances of the application.

Package is a ZIP file that contains the following:

- main package metadata file `APP-META.xml`
- additional metadata files, such as icons and screenshots, referenced from `APP-META.xml`
- management scripts in the `scripts/` directory. The `scripts/configure` script performs application-specific tasks during the application installation, upgrade, patching, reconfiguration and removal.

Here is a structure of a typical package.

```
APP-META.xml      # Metadata container. XML file.
scripts/
  configure       # This script will be invoked when application
  ...             # instance is managed
  ...
  ...             # Additional files to be used by the 'configure'
  ...             # reside in the same directory
images/
  icon1.png       # Icon and screenshots of the application
  screenshot2.jpg
  screenshot.jpg
  ...
htdocs/          # Application files
  index.php
  logo.png
  ...
```

The `APP-META.xml` file contains all the metadata required to manage the application. This includes name, version, description and changelog of the application, resources required for the application and its services to function properly and description of user-supplied configuration settings. Typical structure of metadata:

```
<application xmlns="http://apstandard.com/ns/1"
```

## Application Packaging Standard - Package Format Specification

---

```
    version="1.1" packaged="2008-11-02T09:30:10+06:00">

<!-- common properties -->

<name>Broombla</name>
<version>1.0.11</version>
<release>4</release>
<homepage>http://broombla.com/</homepage>

<!-- application and package vendors -->

<vendor>
  <name>Broombla Corporation</name>
  <homepage>http://broombla.com/</homepage>
  <icon path="icons/corp_logo.gif"/>
</vendor>

<packager>
  <name>Broombla Packaging</name>
  <homepage>http://broombla.com/packages</homepage>
  <icon path="icons/corp_logo.gif"/>
</packager>

<!-- application description -->

<presentation>
  <summary>...</summary>
  <description>
    ...
  </description>
  <icon path="icons/logo.gif"/>
  <screenshot path="img/screenshot1.gif">
    <description>...</description>
  </screenshot>
  <changelog>
    <version version="1.0.11" release="4">
      <entry>Fixed bug in ...</entry>
    </version>
  </changelog>
  <categories/>
  <languages/>
</presentation>

...

<service>

  <license must-accept="true">
    ...
  </license>

  <requirements xmlns:php="http://apstandard.com/ns/1/php">

    <!-- PHP version and extensions requirements -->
    <php:version min="5.0"/>
    <php:extension>mysql</php:extension>

    <!-- Database requirement -->
    <db:db xmlns:db="http://apstandard.com/ns/1/db">
      <db:id>main</db:id>
      <db:default-name>phpbb</db:default-name>
      <db:server-type>mysql</db:server-type>
      <db:server-min-version>3.22</db:server-min-version>
    </db:db>

    <!-- Probably more requirements -->

    ...
  </requirements>

  <provision>
```

```
<url-mapping>
  <!-- Mapping URLs to the files and URL handlers -->
  <mapping url="/" path="htdocs">
    <php:handler/>
  </mapping>
</url-mapping>

</provision>

</service>

</application>
```

## 2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119]

## 3. Document Structure

This specification is divided into the following two parts:

- basic package format
- common aspects

The first part of the specification describes basic metadata required for application to instantiate and operate, and points of extensibility.

The second part consists of several *aspects*. Aspect is a document describing the specific extension to the basic format. The format is modular to accommodate new programming languages, operating systems, software components, etc.

## 4. Basic Package Format

### 4.1. File Format

Package is a ZIP file [ZIP] with the `.app.zip` extension.

### 4.2. Files

Package MUST contain only regular files and directories.

There MUST NOT be two files or directories in one directory which file names differ only in case.

Names of the files included in a package SHOULD contain only printable ASCII characters (except for the TAB, NL, and CR characters (ASCII codes 32-127)). To ensure web application compatibility with Microsoft Windows, names of the files included in a package SHOULD NOT contain the following characters: `<`, `>`, `:`, `"`, `/`, `\`, `|`, `*`, `?`.

Special Windows device names (`con`, `con.*`, `nul`, `nul.*`, `lpt` etc) MUST NOT be used in packages. It is recommended to check for such files during package unpacking on Windows.

### 4.3. Metadata File

Each package MUST contain a well-formed XML file named `APP-META.xml` in the package root directory. This file contains all the package metadata.

## 5. Metadata Descriptor

RELAX NG schema of metadata descriptor:

```
default namespace sa = "http://apstandard.com/ns/1"
namespace          local = ""
```

Application Packaging Standard  
- Package Format Specification

---

```
grammar {

    start = Application

    Application = element application {
        ## Version of APS format used
        attribute version { text }?,
        ## Packaging date
        attribute packaged { xsd:dateTime }?,

        element name          { text },
        element version       { text },
        element release       { text },
        element homepage      { xsd:anyURI }?,

        element vendor {
            element name      { attribute xml:lang { text }?,
                               text }+,
            element homepage  { attribute xml:lang { text }?,
                               xsd:anyURI }*,
            element icon      { attribute path { text } }?
        }?,

        element packager {
            element name      { attribute xml:lang { text }?,
                               text }+,
            element homepage  { attribute xml:lang { text }?,
                               xsd:anyURI }*,
            element icon      { attribute path { text } }?,
            element uri       { xsd:anyURI }?
        }?,

        element presentation {
            element summary   { attribute xml:lang { text }?,
                               text }*,
            element description { attribute xml:lang { text }?,
                                  text }*,
            element icon      { attribute path { text } }?,

            element screenshot { attribute path { text },
                                  element description {
                                      attribute xml:lang { text }?,
                                      text }+
            }*,

            element changelog {
                element version { attribute version { text },
                                   attribute release { text },
                                   attribute date { xsd:dateTime }?,

                                   element entry {
                                       attribute xml:lang { text }?,
                                       text }+
                }*
            }?,

            element categories { element category { text }+ }?,

            element languages { element language {
                                   xsd:string { pattern = "[a-z]{2,3}" } }+ }?,

            element extension { AnyElement* }?
        }?,

        element global-settings { ( Setting | SettingGroup )* }?,

        element patch { attribute match { text },
                        attribute recommended { "true" }? }?,

        element upgrade { attribute match { text } }?,

        element content { ContentDeliveryMethod }?,
```

Application Packaging Standard  
- Package Format Specification

---

```
Service*
}

AnyElement = element * { attribute * { text }*,
                        ( text | AnyElement)* }

## Update version and release
Version = attribute version { text }, attribute release { text }
VersionRecommended = Version, attribute recommended {"true"}?

ContentDeliveryMethod = DefinedByAspect

## Services
Service = element service {
    attribute id { text }?,
    attribute class { "account" |
                    "service" |
                    "ecommerce" |
                    text }?,

    ( element license { EULA } |
      element license-info { ProductCode } )?,

    element presentation { Presentation }?,
    element settings { ( Setting | SettingGroup )* }?,

    ## Requirements specification [5.4.4]
    element requirements {
        element choice { element requirements { attribute id { text },
                                                Requirement+ }+
                        },
        Requirement*
    }?,

    element provision {
        element when-chosen { attribute requirements-id { text },
                             ProvisionMethod+
                          },
        ProvisionMethod+
    }?,

    Service*
}

## End-user license agreement
EULA = ( attribute must-accept { xsd:boolean },
        ( element free { empty } |
          element commercial { empty } )?,

        element text { attribute xml:lang { text }?,
                      element name { text }?,
                      ( element url { xsd:anyURI } |
                        element file { text } )
        )+

)

# Product license identifier
ProductCode = element product-code { xsd:string { minLength = "1" } }

## GUI-related settings
Presentation =
    ## Textual description of the service
    element name { attribute xml:lang { text }?, text }*,
    element summary { attribute xml:lang { text }?, text }*,
    element icon { attribute path { text } }?,

    ## Informational link specification [5.2.2]
    element infolinks {
        element link { attribute xml:lang { text }?,
                      attribute class { "official" |
                                        "community" |
```

Application Packaging Standard  
- Package Format Specification

```

        "howto" |
        "support" |
        "demo" |
        text }? ,
    attribute href { xsd:anyURI },
    text }+
}?,

element entry-points {
    element entry { attribute class { "control-panel" |
        "login" |
        "frontpage" |
        "check" |
        text }? ,

        ## URI templates are allowed
        attribute dst { xsd:anyURI | text }? ,
        attribute method { "GET" |
            "POST" |
            text }? ,

        element label { attribute xml:lang { text }? ,
            text }+ ,
        element description { attribute xml:lang { text }? ,
            text }* ,
        element icon { attribute path { text } }? ,

        ## Optional parameters for using entry point
        element variable { attribute name { text } ,
            attribute class { "login" |
                "password" |
                "locale" |
                text }? ,
            attribute value-of-setting { text }? ,
            text
        }* ,

        element extension { AnyElement* }?
    }+
}?,

    element extension { AnyElement* }?

## Settings
div {
    SettingGroup = element group { attribute class { "authn" |
        "presentation" |
        "web" |
        "vcard" |
        "type" |
        text }? ,

        ## groups with name of @class "type" are used to
        ## express hCard subproperties
        # Due to XML Schema restrictions, distinct definitions were joined
        element name { attribute xml:lang { text }? ,
            attribute class { "type" }? ,
            text }* ,

        ## Nested anonymous groups are allowed for the sake of
        ## microformats compliance
        ( SettingGroup | Setting )*
    }

    Setting = element setting { attribute id { text } ,
        attribute class { text }? ,
        ( attribute installation-only { "true" } |
            attribute track-old-value { "true" } )? ,

        element name { attribute xml:lang { text }? , text }* ,
        element description { attribute xml:lang { text }? , text }* ,
        element error-message { attribute xml:lang { text }? , text }* ,

```

Application Packaging Standard  
- Package Format Specification

```
# Settings types
( ( attribute type          { "boolean" },
  attribute default-value { "true" | "false" }? )

| ( attribute type          { "string" | "password" },
  attribute min-length     { xsd:nonNegativeInteger }?,
  attribute max-length     { xsd:nonNegativeInteger }?,
  attribute default-value  { text }?,
  ( attribute regex        { text } |
    attribute charset      { text } )? )

| ( attribute type          { "integer" },
  attribute min             { xsd:integer }?,
  attribute max             { xsd:integer }?,
  attribute default-value  { xsd:integer }? )

| ( attribute type          { "float" },
  attribute min             { xsd:float }?,
  attribute max             { xsd:float }?,
  attribute default-value  { xsd:float }? )

| ( attribute type          { "email" },
  attribute default-value  { text }? )

| ( attribute type          { "domain-name" },
  attribute default-value  { text }? )

| ( attribute type          { "enum" },
  attribute default-value  { text }?,
  element choice { attribute id { text },
                  element name { attribute xml:lang { text }?,
                               text }*
  }+ )

| ( attribute type          { "static-text" },
  ( attribute default-value { text } |
    attribute value-of-setting { text } ) )

| ( attribute type          { "hidden" },
  ( attribute default-value { text } |
    attribute value-of-setting { text } ) )
)
}
}

Requirement = DefinedByAspect

## Aspect must use own namespace
DefinedByAspect = element * - ( sa:* | local:* ) {
  attribute * { text }*,
  ( text | DefinedByAspect )*
}

## Provision specification
div {
  ProvisionMethod = element url-mapping          { UrlMapping } |
                   element configuration-script { ConfScript } |
                   DefinedByAspect

  ## URL Mapping
  UrlMapping = element default-prefix { xsd:string { minLength = "1" } }?,
              element installed-size { xsd:nonNegativeInteger }?,
              element mapping        { Mapping }*

  Mapping = attribute url { xsd:string { minLength = "1" } },
            ( attribute path { xsd:string { minLength = "1" } } |
              attribute virtual { "virtual" } )? ,
            DefinedByAspect*,
            element mapping { Mapping }*

  ## Configuration Script
```

## Application Packaging Standard - Package Format Specification

---

```
ConfScript = attribute name { text },
              attribute privileged { "true" }?,
              ( element configuration-script-language { text } |
                element binary-executable { empty } ),
              element status-control { empty }?
}
}
```

The APP-META.xml file MUST be valid according to the schema above. XML namespace of the metadata will be changed when incompatible changes are introduced.

Package SHOULD declare the version of APS specification with which it complies with the help of the version attribute.

```
<application xmlns="http://apstandard.com/ns/1" version="1.1">
</application>
```

The version of APS format specified herein is 1.1. The numbering scheme for APS format versions is major.minor. The major and minor numbers MUST be treated as separate integers and each number MAY be incremented higher than a single digit. Thus, APS 1.2 would be a lower version than APS 1.10. Leading zeros (e.g., APS 1.01) MUST be ignored by APS controllers and MUST NOT be specified.

Package SHOULD use the lowest possible version of APS specification that is sufficient for describing the application's capabilities and requirements.

Package SHOULD declare packaging date with the help of the packaged attribute.

```
<application xmlns="http://apstandard.com/ns/1"
              version="1.1" packaged="2008-11-02T09:30:10+06:00">
</application>
```

Metadata schema contains several places where arbitrary elements may be added: requirements may be added to the requirements elements, provision methods may be added to the provision element and URL handlers may be added to the mapping elements. The allowed types of requirements, provision methods and URL handlers are described in aspects. Structure of these elements are described in subsequent specification sections.

When a Controller encounters requirement which it does not know how to handle, the Controller should consider this requirement as non-satisfiable. If such requirement is found outside of the choice element, the package MUST NOT be installed. If such requirement is found inside the choice element, then the branch in which it is contained MUST NOT be selected during installation.

If a Controller encounters unknown elements in the mapping section, the package installation must be aborted.

All user-visible strings in metadata descriptor are localizable. Localization is performed by using the standard XML xml:lang attribute: every localizable string has optional attribute xml:lang:

```
...
<description>Some description</description>
<description xml:lang='de-DE'>...</description>
<description xml:lang='ja-JA'>...</description>
...
```

String without explicit xml:lang is always required.

## 5.1. Common Application Properties

The following properties are common for all applications:

### 5.1.1. Package Name

```
<name>phpbb</name>
```

Free-formed string specifies user-visible name of application in a package. Package name is a package identifier - it **MUST NOT** be changed in consequent versions of packages, otherwise package upgrade and patch from older versions will be not feasible.

### 5.1.2. Package Version

```
<version>2.0.22</version>  
<release>6</release>
```

Package version consists of two parts: application version and package release, the former corresponds to the version of application packaged, and the later to the release of the package containing the same version of application (packages may be released many times, e.g., for fixing bugs in packaging or adding localizations).

Version format and the algorithm for determining the chronological relationship between different package versions are specified by the Debian Policy: Version Format in Debian Policy [<http://www.us.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Version>].

Unlike the Debian version-release approach, application version and package release are separated to ease parsing.

### 5.1.3. Homepage

```
<homepage>http://phpbb.com/</homepage>
```

URL of the application official site.

### 5.1.4. Software Vendor Information

```
<vendor>  
  <name>Broombla Corporation</name>  
  <homepage>http://broombla.com</homepage>  
  <homepage xml:lang="ja-JA">http://broombla.com/ja/</homepage>  
  <icon path="icons/broombla-corp-logo.gif"/>  
</vendor>
```

Characteristics of software vendor whose application is packaged. Both the name and homepage allow localization. The `icon/@path` attribute **MUST** contain a full path in archive to the icon file. The icon must be a 64x64 pixels image in JPEG, PNG or GIF format.

### 5.1.5. Software Packager Information

```
<packager>  
  <name>Parallels</name>  
  <homepage>http://parallels.com</homepage>  
  <icon path="icons/parallels-package-logo.gif"/>  
  <uri>uuid:15d041e8-34c6-409a-b165-3290d2c9d599</uri>  
</packager>
```

Characteristics of package manufacturer. Both the name and the homepage elements allow localization. The `icon/@path` attribute **MUST** contain a full path in archive to the icon file. The icon must be a 64x64 pixels image in JPEG, PNG or GIF format.

The `uri` element is an arbitrary URI unique for each packager (it is **RECOMMENDED** to use `uuid`: URI scheme to minimize the possibility of clash). This URI is needed to distinguish packages with

the same name but created by different packagers. Note that consequent versions of the same package **MUST** have the same URI, as otherwise package Controllers **MAY** refuse to update package from one version to another.

Controllers **SHOULD** allow to upgrade and patch package from the version which does not specify `uri` to the one which specifies in order to support smooth upgrade path for the packages which did not use `uri` from the beginning.

### 5.1.6. Summary

```
<presentation>
  <summary>
    High powered, fully scalable, and highly customizable Open Source bulletin
    board package.
  </summary>
  <summary xml:lang="es-ES">...</summary>
  ...
</presentation>
```

Single-sentence summary of the package for end users.

### 5.1.7. Description

```
<description>
  phpBB is a high powered, fully scalable, and highly customizable
  Open Source bulletin board package. phpBB has a user-friendly
  interface, simple and straightforward administration panel, and
  helpful FAQ. phpBB is the ideal free community solution for all web
  sites.
</description>
<description xml:lang="it-IT">...</description>
```

One-paragraph description of the package for end users.

### 5.1.8. Icon

```
<icon path="images/phpbb.png" />
```

Icon may be provided to be displayed in GUI for the application. The `path` attribute **MUST** contain a full path in archive to the icon file. The icon must be a 64x64 pixels image in JPEG, PNG or GIF format.

### 5.1.9. Screenshots

```
<screenshot path="images/admin.png">
  <description>Administrative interface</description>
  <description xml:lang="he-IL">...</description>
</screenshot>
<screenshot path="images/main.png">
  <description>Main page</description>
  <description xml:lang="ja-JA">...</description>
</screenshot>
```

Several screenshots with descriptions may be provided. The `path` attribute **MUST** contain a full path in archive to the screenshot file. It must be JPEG, PNG or GIF image.

### 5.1.10. Changelog

```
<changelog>
  <version version="2.1.22" release="1">
    <entry>New upstream version</entry>
    <entry xml:lang="de-DE">...</entry>
  </version>
  <version version="2.1.21" release="5">
    ...
  </version>
```

```
...  
</changelog>
```

Changelog contains the human-readable list of changes between consecutive package versions. Order of entries in changelog is not specified, Controller should sort them.

### 5.1.11. Categories

```
<categories>  
  <category>Collaboration/Portals</category>  
  <category>Web/Content management</category>  
</categories>
```

Package may include a set of categories. Category is a Unicode string without attached semantics. The first category should be "primary category" in the sense the first category should be adequate for sorting packages in the user interface.

A list of predefined categories is available at [APS Categories]. The specified categories names SHOULD be used. Other categories names MAY be used, but handling them in Controller is OPTIONAL.

### 5.1.12. Languages

```
<languages>  
  <language>en</language>  
  <language>de</language>  
  <language>ru</language>  
</languages>
```

Package may declare a set of languages for the presentation purposes. The first language is the "default language" of the application. Languages are identifiers from ISO 639.

### 5.1.13. Updates

Application may declare versions of packages which can be updated to the current package. Two update modes are supported:

- Patch - version change without major changes in application settings and without any changes in deployment logic. In particular, all allocated resources are left as is, and no changes in application mapping scheme are allowed. Moderate changes in application settings are allowed, however several classes of changes which may lead to ambiguity are prohibited. See details in the Settings and Requirements sections.

Such restrictions allow unattended update of all application instances, thus making patches is a preferable way to apply crucial changes, such as security fixes.

If a patch fixes problem that affects all software users, it SHOULD declare the recommended element. Otherwise, it is assumed that patch fixes some specific problem, or implements additional functionality and is intended to be installed only by those users who are experiencing the problem.

- Upgrade - version change which allows complex changes in application settings and deployment logic. This operation may require user attendance.

A Package may specify which versions it can update with help of `match` attribute. This attribute contains XPath expression that will be evaluated against metadata of the installed packages with the same name/packager pair and lower version/release numbers. If the specified XPath expression results in non-empty nodeset on some examined application metadata - the package is considered to be an update for the examined application.

The XPath expression must assume `http://apstandard.com/ns/1` to be the default namespace.

For exact version comparison a Controller MUST be able to compare packages versions as specified in Package Version section of specification with help of `<` `>` and `=` XPath operators.

### Example 1. Specification of exact updatable versions

```
<patch match="( /application/version = '2.0' and /application/release='1')
    or ( /application/version = '2.0' and /application/release='2')" recommended="true" />
<upgrade match="/application/version = '1.0' and /application/release='1'" />
```

Such specification means that package can patch the installed version 2.0.1 or 2.0.2 and upgrade version 1.0.1.

### Example 2. Specification of updatable version ranges

```
<patch match="/application/version > '2.0'" recommended="true" />
<upgrade match="/application/version > '1.0'" />
```

Such specification means the following:

- If the installed version is greater than 2.0, patch is possible. Patch contain crucial fixes, affecting all application users.
- If the installed version is greater than 1.0 and less or equal to 2.0, upgrade is possible.

If the update specification is absent, it is supposed that updates are not supported by the package at all.

#### 5.1.14. Content Delivery Methods

```
<content xmlns:pvc="http://apstandard.com/ns/1/pvc">
  <pvc:templates class="win">
    ...
  </pvc:templates>
</content>
```

Application may require a special processing of application's archive content prior to actual usage. Allowed content processing options are defined in aspects.

#### 5.1.15. Global Settings

```
<global-settings>
  <setting id="host">
    ...
  </setting>
</global-settings>
```

Package may have global settings that are visible and affect all instances of application services. Global settings SHOULD be set prior to any service provision. Such settings are declared by the `global-settings` element within the application description. When a global setting is changed, all instances of the application services need to be reconfigured immediately. Global settings MUST NOT use the `value-of-setting` attribute or settings of `type="hidden"`.

### 5.2. Services

Application is supposed to provide services to its users. Complex applications can provide several services of different nature and logic. Distinct services are declared with the `service` elements:

```
<application>
  <name>Personal Information Manager</name>
  ...
  <service id="contacts">
    ...
  </service>

  <service id="calendar">
    ...
  </service>
```

```
</application>
```

Each service may feature its own settings, use its own resources and require a special provisioning activity. Each service in the package **MUST** be uniquely identified, that is, either be the only service in the package, or have a unique `id` within application package.

Services can be nested. Provision of child services is performed in the scope of already provisioned parent service. Parent service is assumed to be an environment for child services. Entire host is an environment for the root services.

```
<application>
  <name>Webmail</name>
  ...
  <service id="email">
    ...
    <service id="mailbox">
      ...
    </service>
    <service id="addressbook">
      ...
    </service>
  </service>
</application>
```

Child services are allowed to reference settings values and resources of parent service.

During updates package **SHOULD NOT** alter the services hierarchy in a way that may introduce ambiguity in update of already provisioned services. In particular, new version of package **SHOULD NOT**:

- remove services from hierarchy
- move services from one hierarchy level to another
- move services from one hierarchy branche to another

A Controller **MUST** refuse to update already provisioned service instance if its place in the service hierarchy has been changed. Service instances may be updated individually if the provision method allows this.

A service may define:

- end-user license agreement
- control elements to be added to user interface
- settings for the service instance
- required resources
- activities which must be performed upon service provisioning

In order to help Control Panel to display services information to customers service **MAY** provide hints on its kind with help of `class` attribute. This specification declares the following classes of services:

`account` - a personal user account in parent service

`service` - discrete function of the application

`ecommerce` - E-commerce service

If a Controller encounters unknown value of this attribute it should treat the attribute as unspecified and ignore it.

### 5.2.1. Service Summary

Service **SHOULD** be described with a brief summary information.

### Example 3. Service summary information

```
<service id="email">
  <presentation>
    <name>Email</name>
    <name xml:lang="de"> ... </name>
    <summary>Electronic mail address with mailbox</summary>
    <icon path="images/email.gif"/>
    ...
  </presentation>
  ...
</service>
```

#### 5.2.2. License Agreement

```
<license must-accept="true">
  <free/>
  <text>
    <name>GPLv2</name>
    <file>licenses/gplv2.txt</file>
  </text>
  <text xml:lang="de-DE">
    <name>GPLv2</name>
    <file>licenses/gplv2-de_DE.txt</file>
  </text>
</license>
```

or

```
<license>
  <text>
    <name>Revised BSD</name>
    <url>http://opensource.org/licenses/bsd-license</url>
  </text>
</license>
```

A license declaration MAY include name of the license, whether the license must be accepted by a user, and either a full path to the license file in the package or a URL of the full text of the license. The `file` element MUST be a full path in the archive to the existing file. Application license may be characterized as free or commercial by using appropriate `free` or `commercial` element.

#### 5.2.3. Information Links

The packager may declare links to external web resources that a control panel should show for the given service. Text to be shown for the link is specified as the `link` element's value. A control panel SHOULD show only informational links relevant for current user's locale.

### Example 4. Information links for service

```
<presentation>
  <infolinks>
    <link xml:lang="en" class="official" href="http://example.com">
      Official site
    </link>
    <link xml:lang="de" class="official" href="http://example.com/de">
      Offizielle Website
    </link>
  </infolinks>
</presentation>
```

The `class` attribute MAY be used to inform a control panel about the destination web resource meaning. The following values of this attribute are predefined:

`official` - official site of application or application vendor  
`community` - application community portal

howto - resource contains howto articles

support - support service site

demo - online application demo

Controller MAY use this information when grouping the links or optionally displaying them. Controller SHOULD ignore unknown values of the `class` attribute and treat it as unspecified.

#### 5.2.4. Entry Points

Packager may declare entry points to be used for a given service. When a Controller provisions a service with entry points, it MAY provide user with the choice of what entry points to show in control panel interface and where.

#### Example 5. Entry points

```
<entry-points>

  <entry dst="/info">
    <label>Information service</label>
  </entry>

  <entry dst="/guest">
    <label>Guest view</label>
    <label xml:lang="de-DE">...</label>
    <description>View the gallery as user not logged in</description>
    <icon path="images/icon-guest.png"/>
  </entry>

  <entry class="control-panel" dst="/admin" method="POST">
    <label>Administrative interface</label>
    <icon path="images/icon-admin.png"/>
    <variable name="username" class="login">admin</variable>
    <variable name="password" class="password"
      value-of-setting="admin_password"/>
  </entry>

  <entry dst="/stats{-prefix|/category|default-category}" method="GET">
    <label>Statistics</label>
    <icon path="images/icon-stats.png"/>
    <variable name="default-category"
      value-of-setting="category_to_show_by_default"/>
  </entry>

  <entry dst="{isp}/account/{account}" method="GET">
    <label>My Account</label>
    <icon path="images/icon-money.png"/>
    <variable name="isp"
      value-of-setting="isp_url">http://example.com</variable>
    <variable name="account" class="login" value-of-setting="login_in_isp"/>
  </entry>

</entry-points>
```

The `dst` attribute of an entry point may contain absolute or relative URI of the entry point. Templates conforming to [URI Templates] are allowed. For relative URIs, absolute URL will be generated when entry point is being displayed, pointing to the base URL of application + URI of entry point. GET or POST HTTP request method MAY be declared to be used to access HTTP entry point. GET is assumed by default.

When displaying an entry point, Controller SHOULD use the provided label, description and icon. The `icon/@path` attribute MUST contain a full path in archive to the icon file. Icon MUST be a 64x64 pixels image in JPEG, PNG or GIF formats.

The `entry/@class` attribute MAY be used to inform Controller about the entry point designation. The following values of this attribute are predefined:

`control-panel` - entry point leads to the service control panel

`login` - entry point leads to the login page

`frontpage` - entry point leads to the application's front page  
`check` - entry point leads to page that displays application health information  
Controller MAY use this information for displaying the entry point. Controller MUST ignore unknown `entry/@class` and treat such entry point as being without explicit class specification. Application MAY declare several entry points with the same `class` attribute.

Entry point MAY declare request variables to be specified when user clicks entry point in control panel interface. Variables values may be taken from actual service settings using the `value-of-setting` attribute or explicitly specified as the `variable` element value. If the specified setting has empty value, the `variable` element value must be used. These variables are also used for URI Templates expansion. Variables with duplicate names are allowed and may be used for arrays representation.

If the POST request method is used, variables MUST be submitted using the `application/x-www-form-urlencoded` encoding. In case of the GET method, variables must be submitted as specified by URI template syntax. If URI template is not used, variables are submitted using the `application/x-www-form-urlencoded` encoding.

The `variable/@class` attribute MAY be used to inform a Controller about the request variable meaning. The following values of this attribute are predefined:

`login` - request variable contains login  
`password` - request variable contains password  
`locale` - request variable carries localization information. Value of this parameter must be as described in the `Settings for Service Appearance Customization` section.

Controller MAY provide arbitrary values for such variables basing on its own considerations. Controller MUST ignore unknown `variable/@class` and treat such variable as variable without explicit class specification. Application MAY declare several variables with the same `class` attribute. Controller SHOULD provide equal values for variables with the same class.

### 5.2.5. Service Settings

Applications often need additional parameters for successful installation and configuration. While most of the questions asked during conventional application installation are related to various resources and answers may be provided by Controller without user intervention, some settings need to be entered by user. Controller MUST keep state of application settings values to pass them to the `configuration/update` scripts.

Settings are declared in the `settings` elements in services description. Each `setting` element represents a single setting to be asked from user.

To make user interface more convenient, the following information is supplied for each setting:

`name` - short textual label for the setting  
`description` - description of the setting  
`default-value` - default value for the setting  
`error-message` - error message to be presented to user when the setting is not validated  
`type` - data type of setting (string, number, enum, etc.)

Settings may be declared as "installation only". Settings of this type need to be set up during service provisioning and should not be reconfigured later. This includes all settings which may be configured from the application, as any reconfiguration in the control panel will effectively reset user customizations done in application. Such settings are marked with the `installation-only` attribute.

If a setting has the `track-old-value` attribute and value of the setting is changed, Controller MUST provide old value together with new one when performing service instance reconfiguration.

Regular settings are visible only to service which declares them. Settings with the same name/id are allowed in different services.

Child services may reference values of parent services settings using the `value-of-setting` attribute. This attribute specifies ID of parent service setting whose value must be used for the given

setting. The appropriate parent setting is searched in ascending order. If a parent service does not contain such setting, the next parent is examined upward. Thus, referencing through several levels of nesting is allowed. Referenced setting **MUST** be declared in the package. When the referenced setting value is changed, the affected service instances must be reconfigured. It is prohibited to reference to settings which, in turn, references another settings.

When application is upgraded, it is prohibited to transform settings with the same `id` from being installation-only and to replace setting value with reference to value of another setting (using the `value-of-setting` attribute). This restriction is introduced to protect application instance settings values.

For patches it is also prohibited to introduce new settings without default values, as this prevents unattended patch installation.

Settings may be grouped by the `group` element. Group **MAY** have a name declared by the `name` element. Groups without specified name are called *anonymous*. Group contains a list of settings and **MAY** contain anonymous groups. Settings and groups are listed in the order suggested to be used in interface.

### 5.2.5.1. Data Types

Settings are typed, Controller **SHOULD** use the type information to validate input. The following types are defined:

Type	Values	Optional restrictions	
boolean	true, false		
string, password	Unicode string	min-length	Minimum acceptable length in Unicode characters.
		max-length	Maximum acceptable length in Unicode characters.
		regex	Regular expression, subset of PCRE: <code>. * + ?</code> meta symbols, <code>()</code> grouping, <code>[]</code> character classes (only individual characters, ranges and class negation <code>^</code> ), <code>{,N}</code> , <code>{M,}</code> , <code>{M,N}</code> repetition suffixes.
integer	A 64-bit signed integer number	min	Minimum acceptable integer value (inclusive). The least possible value is -9223372036854775808.
		max	Maximum acceptable integer value (inclusive). The greatest possible value is 9223372036854775807.
float	A double-precision 64-bit floating point number with values from value space $m \times 2^e$ , where $m$ is an integer whose absolute value is less than $2^{53}$ , and $e$ is an in-	min	Minimum acceptable real value (inclusive).
		max	Maximum acceptable real value (inclusive).

Type	Values	Optional restrictions
	teger between -1075 and 970.	
email	email address, as defined in [RFC 2822]	
domain-name	DNS domain name, as defined in [RFC 1035]	
enum	One of supplied identifiers	
static-text	Unicode string	Value should not be changed by user. Controller SHOULD display it as a static text. Application MAY hardcode value of this setting. Value is either a static string declared with the <code>default-value</code> attribute, or equal to value of parent instance setting whose name is specified by the <code>value-of-setting</code> attribute.
hidden	Unicode string	Neither setting name nor value can appear anywhere on application configuration screen. Value is either a static string declared with the <code>default-value</code> attribute, or equal to value of parent instance setting whose name is specified by the <code>value-of-setting</code> attribute.

### 5.2.5.2. Settings Semantics

There are settings which are often required by applications. To give implementers a way to create better interfaces (predefined settings may be used by control panel to provide values without user interaction), the `class` attribute of setting or settings group may be used to inform Controller about the setting meaning.

The following values of the `class` attribute are predefined:

`authn` - settings related to user authentication

`presentation` - settings which carry presentation and UI information

`web` - web site-related settings

`vcard` - personal preferences of service user

Controller MUST ignore unknown `class` value and treat such element as being without explicit semantics specification.

#### 5.2.5.2.1. Authentication Settings

Settings group which declares `class="authn"` MAY use the following classes of settings:

`login` - login name of service user

`password` - password of service user

#### Example 6. Authentication Settings

```
<group class="authn">
  <setting id="user_account_name" type="string" class="login">
    <name>User login</name>
  </setting>
  <setting id="user_pwd" type="password" class="password">
    <name>Password</name>
  </setting>
</group>
```

#### 5.2.5.2.2. Settings for Service Appearance Customization

Settings group which declares `class="presentation"` MAY use the following classes of settings:

`locale` - default locale of service

Values of the setting with `class="locale"` MUST be in format defined in [RFC 3066]. It is REQUIRED to use two-part locale identifiers with [ISO 639] language name as the first part and [ISO 3166] country code as the second part. In other words, 'i-' or 'x-' locale names MUST NOT be used.

It is RECOMMENDED to use the enum type for the setting with `class="locale"` to declare all languages supported by the application.

#### Example 7. Service Appearance Settings

```
<group class="presentation">
  <setting id="user_locale" type="enum" default-value="en-GB" class="locale">
    <name>Default locale</name>
    <choice id="en-GB">
      <name>English</name>
    </choice>
    <choice id="fr-FR">
      <name>French</name>
    </choice>
    <choice id="de-DE">
      <name>German</name>
    </choice>
  </setting>
</group>
```

#### 5.2.5.2.3. Settings for Web Site Preferences

Settings group which uses `class="web"` MAY use the following classes for settings:

`title` - Web site title

`description` - Web site description

#### Example 8. Web Site Settings

```
<group class="web">
  <setting id="site_title" type="string" class="title"
    default-value="Homepage">
    <name>Site Title</name>
  </setting>
  <setting id="site_description" type="string" class="description"
    default-value="Personal Playground">
    <name>Site Description</name>
  </setting>
</group>
```

#### 5.2.5.2.4. Personal Information Settings

Settings group which declares `class="vcard"` MAY use classes of settings defined in [HCARD-PROFILE]

In order to create [HCARD]-compatible XML document, an anonymous nested settings groups MUST be used for declaring hCard properties and the name elements with attribute `class` set to type value MUST be used for declaring subproperties.

### Example 9. Usage of nested groups of settings

```
<group class="vcard">
  <group class="fn n">
    <setting id="user_first_name" class="given-name" ... />
    <setting id="user_last_name" class="family-name" ... />
  </group>
  <group class="tel">
    <name class="type">work</name>
    <setting id="work_phone" class="value" .../>
  </group>
  <group class="tel">
    <name class="type">cell</name>
    <setting id="mobile_phone" class="value" .../>
  </group>
</group>
```

Controller MAY perform implied optimizations according to [HCARD].

The name element MUST NOT contain both the `class` and the `xml:lang` attributes. Only one name element with declared `class` attribute is allowed within one settings group.

### 5.2.6. Requirements

The Requirements section in metadata describes what conditions should be met to provision a service. Requirements usually request particular resource to be allocated, or specific configuration to be performed.

### Example 10. Requirements of Web Application

```
<requirements
  xmlns:php="http://apstandard.com/ns/1/php"
  xmlns:db="http://apstandard.com/ns/1/db">
  <php:version min="5.0.2"/>
  <choice>
    <requirements id="mysql">
      <php:extension>mysql</php:extension>
      <db:db>
        <db:id>main</db:id>
        <db:default-name>wiki</db:default-name>
        <db:can-use-tables-prefix>true</db:can-use-tables-prefix>
        <db:server-type>mysql</db:server-type>
        <db:server-min-version>4.0</db:server-min-version>
      </db:db>
    </requirements>
    <requirements id="postgresql">
      <php:extension>postgresql</php:extension>
      <db:db>
        <db:id>main</db:id>
        <db:default-name>wiki</db:default-name>
        <db:can-use-tables-prefix>false</db:can-use-tables-prefix>
        <db:server-type>postgresql</db:server-type>
        <db:server-min-version>7.4</db:server-min-version>
      </db:db>
    </requirements>
  </choice>
</requirements>
```

Requirements belong to the "requirement types". Requirement types are defined in aspects. Each requirement type has an associated unique XML QName (element name + namespace) and an element schema (preferably in RELAX NG). Every requirement should be described as an XML element according to its schema.

Every requirement type has associated rules (in natural language) of how to satisfy the requirement during the instantiation as the part of the requirement type definition in aspect.

Individual requirements form a complex requirement which needs to be satisfied by Controller. This is performed by logical 'AND' of requirements (when they are just placed in the `requirements` section side-by-side), and logical 'OR' of requirements when they are placed in the `choice` sub-element inside the `requirements` element. Controller has to ensure the logical truth of a complex requirement (this means not all individual requirements need to be satisfied).

Only one level of choices is allowed to simplify building the GUI.

For each `choice` element, the `CHOICE_<id>` environment variable must be passed to the configure script, with the value of the `id` attribute of the selected `requirements` element.

Every resource allocated for application instance as a result of satisfying requirements **MUST** be preserved during upgrade. Exact semantics of preservation is left to the requirement specification.

No changes in application instance resources are allowed during patches. Therefore, Controller **MUST** preserve all resources allocated for previous version of service and **MUST NOT** analyze requirements of new version.

### 5.2.6.1. Requirements Propagation

Information about requirement being satisfied in parent service must be available to child services. That is, the set of environment variables passed to the configuration script of child service must also contain appropriate variables with information about satisfied requirements of parent service. Requirements of child service override value of propagated variables.

### 5.2.7. Service Provision

Application service may be provisioned in a variety of ways depending on available environment or application nature. Provision methods are declared with the `provision` element.

Application may support different provision methods depending on which requirements were satisfied by Controller. This is performed by using the `when-chosen` element. It references selected requirements branch by specifying `requirements-id` attribute's value.

Controller **MUST** choose appropriate provision method according to the satisfied requirements branch. When there is no `when-chosen` element referencing the chosen requirements branch, or no requirements branches are defined, the default provision method specified without the `when-chosen` element **MUST** be used. If it is absent, provision process **MUST** be aborted.

#### Example 11. Different Provision Activities

```
<provision>
  <when-chosen requirements-id="arch-x86_64">
    <mapping url="/" path="cgi/x86_64"/>
  </when-chosen>

  <mapping url="/" path="cgi/i386"/>
</provision>
```

The snippet above demands creation of URL mapping pointing to the `cgi/x86_64/` directory when the service is provisioned in `x86_64` environment, and into `cgi/i386/` for other architectures.

When application is updated, the following two types of provisioned services are distinguished:

- those which require individual update procedure
- those which will be updated automatically during update of the parent service

This division depends on provision methods being used for the service. If at least one of used provision methods requires individual update, the update procedure **MUST** be performed for all the service's

provision methods. If no provision methods require individual update, Controller MUST NOT perform update procedures for the service.

### 5.3. Service Provision Methods

This specification defines the following provision methods: URL Mapping and Configuration Script .

#### 5.3.1. URL Mapping

Web applications are designed to handle incoming requests. URL mapping describes how to map the requested URLs to the particular handlers or files.

#### Example 12. URL Mapping of Web Application

```
<url-mapping>
  <default-prefix>forum</default-prefix>
  <installed-size>5242880</installed-size>

  <mapping url="/" path="htdocs"
    xmlns:php="http://apstandard.com/ns/1/php"
    xmlns:mod-python="http://apstandard.com/ns/1/mod-python">
    <php:handler>
      <php:extension>php</php:extension>
      <php:extension>pinc</php:extension>
    </php:handler>

    <mapping url="upload">
      <php:handler><php:disabled/></php:handler> <!-- Disabling PHP -->
      <php:permissions writable="true"/>
    </mapping>
    <mapping url="stat" virtual="virtual">
      <php:handler><php:disabled/></php:handler>
      <mod-python:handler>com.example.StatHandler</mod-python:handler>
    </mapping>
  </mapping>
</url-mapping>
```

The example mapping describes three contexts: URLs starting from the /, URLs starting from the /upload and URLs starting from the /stat. The two first are mapped to the file system, the third one is virtual (the mod\_python aspect is not defined in this specification, so this is placed here just for illustrative purposes).

The default-prefix element defines a segment of URL path component which should precede the service instance root by default. Controller is allowed to change it. Controller MUST normalize leading and trailing slashes of default prefix when forming the final URL.

Package may include a declaration of approximate size of a single URL mapping instance, to help Controllers estimate the disk space resources needed for service provisioning. For this, the installed-size element should be specified with the declared size of URL mapping instance size in bytes.

Mappings may be nested, and there MUST NOT be two mappings with such URLs that the first URL is the prefix of the second URL. In other words, the construction

```
<mapping url="/">
  <mapping url="foo/bar"/>
  <mapping url="foo/bar/baz"/>
</mapping>
```

is prohibited, and MUST be rewritten as

```
<mapping url="/">
  <mapping url="foo/bar">
    <mapping url="baz"/>
  </mapping>
</mapping>
```

All `url`s in mappings are relative to the parent URL mapping. Absolute URLs are PROHIBITED in the `url` attributes except the root mapping where `url` value MUST be `'/'`.

The `path` attribute of mapping is always path from the root of archive to the directory inside the archive, it must not have the `/` character at the start.

If mapping has an explicit `path` attribute, then the mapping is associated with the directory specified by this attribute. The association means that any URL which is in the scope of the given mapping (URLs in scopes of inner mappings are not in the scope of outer mapping) and is not handled by the handlers declared in the mapping needs to be served as the file from the directory specified.

If a mapping has neither an explicit `path` attribute nor a `virtual` attribute, then the directory associated with the given mapping is calculated from the directory associated with the parent mapping appending the relative URL that the given mapping has. E.g., for the following declarations

```
<mapping url="/" path="htdocs">
  <mapping url="foo/bar">
    <mapping url="baz"/>
    <mapping url="quux" path="somedir"/>
  </mapping>
</mapping>
```

mapping `'/'` is associated with the `htdocs/` directory, mapping `'foo/bar'` is associated with the `htdocs/foo/bar` directory, mapping `'foo/bar/baz'` is associated with the `htdocs/foo/bar/baz` directory, and mapping `'foo/bar/quux'` is associated with the `htdocs/foo/bar/somedir` directory.

If mapping has an explicit `virtual` attribute, then the mapping is not associated with any directory, and requests to the URLs in the scope of this mapping must return 'Not found' error, if not handled by handlers declared in the given mapping.

If an outer mapping does not have an associated directory, then the inner mappings without explicit `path` element do not have an associated directory too.

If the whole application mapping is not virtual (meaning that there is at least one mapping without the `virtual` attribute), the root mapping MUST have the `path` attribute.

Controller MUST use mapping information for deployment and upgrade of application instance files. Files deployment MUST be driven by URL mapping rules.

Aspects declare types of URL handlers, rules of how to process them. Rules regulating propagation of specific URL handlers to inner mappings are also declared in aspects.

The handler defined latter overrides the former. In the following example, all `*.php` files will be processed by CGI handler.

```
<mapping url="users">
  <php:handler>
    <php:extension>php</php:extension>
  </php:handler>
  <cgi:handler>
    <cgi:extension h:handler-type="php">php</cgi:extension>
  </cgi:handler>
</mapping>
```

### 5.3.1.1. Mapping Information Propagation

Information about parent service URL mappings made must be available to child services. A Controller MUST provide details of parent's URL mapping in form of appropriate environment variables supplied for configuration script of child service. The set of environment variables is the same as was supplied for parent's service configuration script. Information propagates until child service declares it's own URL mapping.

### 5.3.1.2. Update

URL Mapping provision method requires individual update of provisioned services. During update, application files will be overwritten. Any scheme of files overwriting during upgrade is permitted, given that it satisfies the following statements:

- Every file existing in both old and new packages gets replaced by new version.
- All files existing in old package, but not in new, are deleted.
- All files existing in new package are installed, overwriting any files present on file system.
- No other file is touched - all files created by users are kept intact.

Thus, files expected to be modified during application work need to be created manually by the configuration scripts upon service provision.

During patch, no changes in mapping structure are allowed. Controller **MUST** leave mapping as it was for previous version of application. Mapping files **MUST** be replaced by new versions. File overwriting semantics is the same as for upgrade.

### 5.3.2. Configuration Script

A service may be provisioned with the help of configuration script. This script will be invoked during application service provisioning, cancelling, updating and reconfiguration. Availability of the script is declared by the `configuration-script` element. Different configuration scripts for different services are allowed.

```
<configuration-script name="configure">  
  <configuration-script-language>php</configuration-script-language>  
</configuration-script>
```

Configuration scripts **MUST** reside in the `scripts` directory in the package root directory. Name of configuration script is specified with the `name` attribute.

Configuration script **MUST** declare programming language it is written in using the `configuration-script-language` element. Controller will use appropriate interpreter to run the configuration script. Valid interpreters are defined in aspects specification . This specification defines a set of `php` , `js` and `vb` interpreters. Binary executable configuration scripts are allowed. In this case, the `binary-executable` element must be used.

Controller **MUST** run configuration script within environment where the application is being installed. Configuration script which requires high privileges in order to run **MUST** use the `privileged` attribute. Controller **MUST** run such script with superuser privileges. Controller **MUST NOT** run in privileged mode configuration script which does not declare this attribute.

During the configuration script execution, the working directory **MUST** be set to the actual location of the script. All content of the `scripts` directory **MUST** also reside in the script current working directory.

If any script invocation fails (script returns a non-zero exit code), it **MUST** be treated as fatal error and Controller **MUST** refuse to continue operation. Stdout and stderr I/O streams of the script should be captured to log error in this case.

#### 5.3.2.1. Configuration Script Actions

Configuration script is invoked during a service is provisioning, reconfiguration, upgrading or cancelling.

##### 5.3.2.1.1. Service Provisioning

Configuration script is invoked when a service is provisioned. By the moment of invocation, all resources declared by the service **MUST** be already allocated and instance files unpacked and placed to the file system. The script is invoked with the following arguments:

```
install
```

#### 5.3.2.1.2. Upgrading Application

Configuration script provision method does not require individual update.

When a service instance is upgraded, the script of new application version is invoked with the following arguments:

```
upgrade <old version> <old release>
```

where <old version> is the old version, and <old release> is the old release of the application being upgraded.

By the moment of invocation, all resources needed by the new version of application MUST be already allocated.

#### 5.3.2.1.3. Changing Settings

Configuration script is invoked when already provisioned service is being configured (this does not include provisioning and canceling). The script is invoked with the following arguments:

```
configure
```

#### 5.3.2.1.4. Canceling Service

Configuration script is invoked during a service cancelation before all allocated resources are freed and instance files removed. The script is invoked with the following arguments:

```
remove
```

#### 5.3.2.1.5. Enabling/Disabling Service

Configuration script MAY be able to manipulate service status by enabling and disabling service. A service which status is *disabled* MUST NOT serve its users or operate on behalf of service owner. Configuration script MUST declare ability to manipulate service status with the `status-control` element.

```
<configuration-script name="reseller">  
  <configuration-script-language>php</configuration-script-language>  
  <status-control/>  
</configuration-script>
```

In this case, Controller MUST invoke the script with the following arguments:

```
enable  
disable
```

to make the appropriate service status changes. Configuration script MUST return success if service is already has a requested status.

#### 5.3.2.2. Environment Variables

All information about application, resources and settings is passed to configuration script through environment variables.

If environment variables in operating system contain bytes (opposed to Unicode codepoints), then UTF-8 encoding is used. All the IDN DNS names passed through environment MUST be passed in Unicode, not in Punycode encoding (in xn--blahblah form).

Several predefined environment variables are always passed to script, and any aspect may declare additional environment variables.

### 5.3.2.2.1. URL Mapping Variables

Services which utilize URL Mapping provision method demand the following environment variables to be passed to configuration script.

Full URL specifying where the service is available, represented by the four environment variables corresponding to the URL parts as defined in [RFC 1738]:

BASE\_URL\_SCHEME - URL scheme. Allowed values: http, https

BASE\_URL\_HOST - URL host.

BASE\_URL\_PORT - URL port (may be omitted if default port for protocol is used: 80 for http, 443 for https).

BASE\_URL\_PATH - URL path including trailing slash.

For example:

```
BASE_URL_SCHEME=http
BASE_URL_HOST=example.com
BASE_URL_PORT not defined
BASE_URL_PATH=phpBB/
```

Note that leading slash is not included in BASE\_URL\_PATH.

Also, for each mapping, except for those which do not map to file system, the WEB\_<id>\_DIR environment variable must be passed with the absolute path to the directory to which the mapping maps, where id is the full URL path of the mapping, with all '/' characters converted to '\_'.  
I.e., for the following URL mapping declaration:

I.e., for the following URL mapping declaration:

```
<mapping url="/" path="htdocs">
  <mapping url="foo/bar">
    <mapping url="baz"/>
    <mapping url="quux" path="somedir"/>
  </mapping>
</mapping>
```

instantiated by the URL http://domain.name/example, the configuration script may be provided with the following environment variables:

```
WEB__DIR=/var/www/vhosts/domain.name/public_html/example/htdocs
WEB__foo_bar_DIR=/var/www/vhosts/domain.name/public_html/example/htdocs/foo/bar
WEB__foo_bar_baz_DIR=\
  /var/www/vhosts/domain.name/public_html/example/htdocs/foo/bar/baz
WEB__foo_bar_quux_DIR=/var/www/vhosts/domain.name/public_html/example/somedir
```

### 5.3.2.2.2. Settings

For each setting declared in a service, corresponding environment variable SETTINGS\_<id> MUST be passed on to the installation script, where <id> is a value of the id attribute in the setting description.

For each setting with the track-old-value attribute, corresponding environment variable OLDSETTINGS\_<id> MUST be passed on to the configuration script, holding the previous value of the setting.

During the service provisioning, all the declared service settings and package global settings MUST be passed to the configuration script. No old values should be passed.

During the service reconfiguration, cancelation and service status change, all the declared service settings and package global settings, except marked as installation-only, MUST be passed to the configuration script.

During the service instance upgrade and patching, all settings of the new version of service which exist in old version **MUST** be set to corresponding values from the old instance. If a settings validator does not allow a value from the old instance, such setting **MUST** be set to the default value. All settings from new package which do not correspond to the settings from old package **MUST** get the default values.

For the `boolean`, `string`, `float`, and `integer` property value data type elements, the corresponding environment variables must contain values entered by user.

For the `enum` setting, the environment variable must contain the identifier of one of the values (defined by the `id` attribute of the `enum/choice` element) selected by the user (e.g., if you have choice with `id interface_color` containing options with IDs `black` and `blue`, then variable `SETTINGS_interface_color` with value `black` or `blue` will be exported).

#### 5.3.2.2.3. Aspect-Defined Environment Variables

Aspects also may define environment variables to be passed to configuration scripts: there might be variables passed when the aspect is used by the package and variables which are passed when a particular requirement is satisfied during configuration. Consult the aspect specifications for the list of environment variables that each aspect defines. See Points of extensibility - Environment variables.

## 6. Points of Extensibility

Aspects are additional specifications that declare how to describe specific needs of applications in packages, and how Controllers must process the descriptions.

Aspects may extend basic specification in the following ways:

- Aspect may declare requirement type.

- Aspect may declare URL handler type.

- Aspect may declare additional files to be packaged.

- Aspect may declare additional environment variables to be passed to configuration script and rules of how to construct their values by Controller.

- Aspect may declare additional languages to be used to run configuration scripts.

- Aspect may declare maintenance scripts in addition to the usual `configure` script in the `scripts` directory.

- Aspect may declare application provision method.

- Aspect **MUST** declare rules of how Controllers should process additional data supplied in a package.

### 6.1. Requirement Types

Each aspect may declare several requirement types. Every requirement type describes how to declare specific requirement of the application.

Every requirement type consists of XML element schema which describes the structure of element representing requirement, and rules of how to process the requirement.

### 6.2. URL Handlers Types

Each aspect may declare several URL handlers types. Every URL handler type declares how to describe rules on how to handle URLs which are specific to the declaring aspect in the scope of a particular mapping.

Every URL handler type consists of XML element schema which describes the structure of element representing URL handler, and rules of how to process the URL handler. Especially, this should include the rules of inheriting URL handlers from the outer mappings in inner mappings.

### 6.3. Additional Files

Aspect may declare that additional files need to be packaged. Aspect **MUST NOT** declare additional files in the `scripts` directory, it is reserved for processing configuration scripts.

## 6.4. Environment Variables

Aspect may declare additional environment variables to be passed to configuration scripts. Such variables will be passed to all invocations of configuration script. It is RECOMMENDED to prefix such variables with the upper-cased name of aspect top-level XML element.

## 6.5. Additional Scripts

Aspect may declare additional scripts to be run during the package lifetime.

Aspect must provide rules of when to run additional scripts, which arguments and environment variables need to be passed to the scripts.

## 6.6. Configuration Script Language

Aspect may declare additional language to be used to run configuration script. In this case, aspect MUST declare the name of this language to be used in basic metadata, and the rules of how to run configuration script.

As there is no established registry of programming language names, it is RECOMMENDED to use lowercased name from the Wikipedia list of programming languages [Langs]

## 6.7. Application Provision Method

To widen a range of supported applications, it is possible to introduce new application provision methods in aspects.

## 6.8. Rules

An aspect must declare the rules of how to process metadata supplied in a package when the package uses this particular aspect: how to process metadata and files, how to generate values for environment variables.

# 7. Common Aspects

This specification defines a number of "common aspects": a set of aspects which are expected to be implemented in Controllers. However, if an aspect is inapplicable to a particular Controller, it may be omitted.

## 7.1. PHP Aspect

This aspect is to be used by web applications written in PHP. This aspect uses the `http://apstandard.com/ns/1/php` XML namespace.

RELAX NG schema of PHP aspect

```
namespace php="http://apstandard.com/ns/1/php"

## URL Handlers
div {
  UrlHandler |= element php:handler {
    ( element php:disabled { empty }
    | element php:extension { text }* )?
  }
  UrlHandler |= element php:permissions {
    attribute writable { "true" }?,
    attribute readable { "false" }?
  }
}

## Requirements
div {
  Requirement |= element php:version {
    attribute min { text }?,
    attribute max-not-including { text }?
  }
}
```

```
Requirement |= element php:extension { text }
Requirement |= element php:function { text }

Requirement |= element php:allow-url-fopen { xsd:boolean }
Requirement |= element php:file-uploads { xsd:boolean }
Requirement |= element php:magic-quotes-gpc { xsd:boolean }
Requirement |= element php:register-globals { xsd:boolean }
Requirement |= element php:safe-mode { xsd:boolean }
Requirement |= element php:short-open-tag { xsd:boolean }

Requirement |= element php:memory-limit { xsd:integer }
Requirement |= element php:max-execution-time { xsd:integer }
Requirement |= element php:post-max-size { xsd:integer }
}
```

### 7.1.1. Requirement Types

PHP aspect declares the following requirement types: PHP version requirement type, PHP extension requirement type, PHP function requirement type, and a set of PHP settings requirement types.

#### Example 13. PHP Version Requirement

```
<php:version min="4.1" max-not-including="5.0"/>
```

Requirement of this type is satisfied if the version of PHP enabled on a site is in [min, max-not-including) interval taken from the requirement. Both limits are optional.

The PHP\_VERSION environment variable MUST be passed on to the configuration script with the version of PHP (as a string value) installed on the Web site where the package is to be installed.

#### Example 14. PHP Extension Requirement

```
<php:extension>curl</php:extension>
<php:extension>Zend Optimizer</php:extension>
<php:extension>ionCube Loader</php:extension>
```

Requirement of this type is satisfied if the specified PHP extension is enabled for the web application.

The names of PHP extensions are specified in the zend\_module\_entry structure of extension code and may be obtained by the get\_loaded\_extensions PHP function.

#### Example 15. PHP Function Requirement

```
<php:function>system</php:function>
```

Requirement of this type is satisfied when the specified PHP function is enabled in PHP.

#### Example 16. PHP Settings Requirements

```
<php:allow-url-fopen>true</php:allow-url-fopen>
```

If a web application needs one of allow\_url\_fopen, file\_uploads, safe\_mode, short\_open\_tag, register\_globals or magic\_quotes\_gpc settings to be true or false, the appropriate requirement must be used. Requirements of those types are satisfied when the corresponding PHP setting has the specified value.

#### Example 17. PHP Limits Requirements

```
<php:memory-limit>16777216</php:memory-limit>
```

If a web application needs a particular value of memory\_limit, max\_execution\_time or post\_max\_size settings, it should use these requirement types (defining values in bytes and seconds).

Requirements of those types are satisfied when the corresponding PHP setting has the value specified in the requirement or larger.

### 7.1.2. URL Handlers

#### Example 18. PHP URL Handler

```
<php:handler>  
  <php:extension>php</php:extension>  
  <php:extension>pinc</php:extension>  
</php:handler>
```

```
<php:handler><php:disabled/></php:handler>
```

Handlers of this type require that files with the specified extensions (if no extensions are specified, a single php extension is assumed) are handled by the PHP interpreter.

Inner mappings inherit handlers from the outer mappings, so the presence of `php:disabled` disables PHP in the given mapping.

#### Example 19. PHP Permission Handler

```
<php:permissions writable="true">
```

```
<php:permissions readable="false">
```

The situation when a directory and files in the directory to which the given mapping maps should be writable by PHP interpreter is specified by using the `writable` attribute with the "true" value. The "false" value is default, so no explicit attribute for this is required.

The situation when files in the directory should be protected from reading by PHP interpreter is specified by the `readable` attribute with the "false" value. The "true" value is default, so no explicit attribute for this is required.

Inner mappings do not inherit permission handlers from the outer mappings.

### 7.1.3. Configuration Script Language

This aspect defines the `php` identifier to be used by configuration scripts. When a configuration script uses the `php` language, it **MUST** be run by stand-alone PHP interpreter. All the requirements described in the application metadata apply to the interpreter running configuration scripts.

## 7.2. ASP.NET Aspect

This aspect should be used by web applications which use ASP.NET. This aspect uses the `http://apstandard.com/ns/1/aspnet` XML namespace.

RELAX NG schema of ASP.NET aspect:

```
namespace aspnet = "http://apstandard.com/ns/1/aspnet"  
  
## URL Handlers  
div {  
  UrlHandler |= element aspnet:permissions {  
    attribute writable { "true" }?  
  }  
  UrlHandler |= element aspnet:handler {  
    ( element aspnet:disabled { empty }  
    | element aspnet:extension { text }* )?  
  }  
}  
  
## Requirements  
div {  
  Requirement |= element aspnet:version { "1.0" | "1.1" | "2.0" | "3.0" | "3.5" }
```

```
}
```

### 7.2.1. Requirement Types

This aspect declares a single requirement type: ASP.NET version requirement. Requirement of this type is satisfied when the application is installed in virtual directory with the specified version of ASP.NET enabled.

### 7.2.2. URL Handler Type

#### Example 20. ASP.NET URL Handler

```
<aspnet:handler/>  
<aspnet:handler>  
  <aspnet:disabled/>  
</aspnet:handler>
```

Handler of this type requires that files with the specified extensions (if no extensions are specified, the default set of ASP.NET extensions is assumed) are handled by ASP.NET.

Inner mappings inherit handlers from the outer mappings, so the presence of `aspnet:disabled` disables ASP.NET in the given mapping.

#### Example 21. ASP.NET Permission Handler

```
<aspnet:permissions writable="true">
```

The situation when files in the directory to which the given mapping maps should be writable by the ASP.NET interpreter is specified by using the `writable` attribute with the "true" value. The "false" value is default, so no explicit attribute for this is required.

### 7.2.3. Configuration Script Language

This aspect defines `javascript` and `vbscript` identifiers to be used by configuration scripts (written in JScript and VBScript correspondingly).

## 7.3. Database Aspect

This aspect should be used by web applications which need a database to operate. This aspect uses the `http://apstandard.com/ns/1/db` namespace. Requirements specific for database management systems use their own namespaces. This specification defines the `http://apstandard.com/ns/1/db/mysql` namespace for requirements specific for MySQL.

RELAX NG schema of database requirement type:

```
namespace db      = "http://apstandard.com/ns/1/db"  
namespace mysql  = "http://apstandard.com/ns/1/db/mysql"  
  
## Requirements  
div {  
  Requirement |= element db:db {  
    element db:id { text },  
    element db:default-name { text }?,  
    element db:can-use-tables-prefix { xsd:boolean },  
    element db:server-type { text },  
    element db:server-min-version { text },  
  
    element db:features { DbFeature+ }?  
  }  
  
  DbFeature |= element mysql:privilege {  
    attribute disabled { "true" }?,  
    text  
  }  
}
```

## Example 22. Database Requirements

```
<requirements>
  <db:db xmlns:db="http://apstandard.com/ns/1/db">
    <db:id>storage</db:id>
    <db:default-name>broombla</db:default-name>
    <db:can-use-tables-prefix>true</db:can-use-tables-prefix>
    <db:server-type>mysql</db:server-type>
    <db:server-min-version>4.1.0</db:server-min-version>

    <db:features xmlns:mysql="http://apstandard.com/ns/1/db/mysql">
      <mysql:privilege>Create_tmp_table_priv</mysql:privilege>
    </db:features>
  </db:db>
</requirements>
```

This aspect declares database requirement type. This requirement type should be used when a service needs a database. The following elements comprise the database requirement type:

id	Identifier of the database. This identifier will be used in environment variables passed to configuration scripts.
default-name	Optional. Proposed name of a database. It is not guaranteed that the allocated database will use this name.
can-use-tables-prefix	This element must have the <code>true</code> value only if a service is able to cope with sharing a database with another services by using prefixed tables. Otherwise, its value should be <code>false</code> .
server-type	This element must be one of the database server identifiers described below.
server-min-version	This element describes the minimum acceptable version of database server software.
features	This element describes peculiarities of required database and database user that are specific for database management system.

Requirement of this type is satisfied when the following is true:

New database is allocated for the application.

If `can-use-tables-prefix` is `true`, then, instead, an existing database is used, a unique prefix is chosen so that no tables in this database have this prefix and all other applications using this database get prefixes.

Allocated database satisfies the `server-type` and `server-min-version` constraints.

User is created, or exists already, having a full access to the database.

If the allocated database uses database management system for which specific requirements are imposed in the `features` section, they must also be satisfied.

Controller **MUST** deallocate database when service is canceled. When `can-use-table-prefix` is `true` and existing database was used, Controller **MUST** remove relevant tables from database.

Database requirements with the same `id` value are allowed only in different branches of a single choice grouping. It is illegal to declare database requirements with the same `id` in different choice `s` or in a choice and outside it at the same time.

### 7.3.1. Environment Variables

When a requirement is satisfied, the following environment variables must be passed to configuration scripts:

- `DB_<identifier>_TYPE`. The database server type (contents of the `server-type` element).

- DB\_<identifier>\_NAME . The name of allocated (or reused) database.
- DB\_<identifier>\_LOGIN . The database user login name. This is the full database access user.
- DB\_<identifier>\_PASSWORD . The database user password. This is the full database access user.
- DB\_<identifier>\_HOST . The database server host name or IP address.
- DB\_<identifier>\_PORT . The port number for connecting to the database server. If the port number is default for the selected DB server, this variable may be omitted.
- DB\_<identifier>\_VERSION . The version of the database server
- DB\_<identifier>\_PREFIX . The prefix of tables in database. **MUST NOT** exist if the application owns a whole database. **SHOULD NOT** exist if a Controller does not support sharing databases between different applications. Application **MUST NOT** create or alter tables without this prefix if it is supplied.

Environment variables DB\_<identifier>\_HOST and DB\_<identifier>\_PORT **MUST NOT** be specified if an application is to use local transport (UNIX sockets or named pipes) to connect to database.

### 7.3.2. Database Server Types

The `server-type` element describes the name of database server. Currently defined names are:

mysql - MySQL  
postgresql - PostgreSQL  
microsoft:sqlserver - Microsoft SQL Server

Another names **SHOULD** be taken from the JDBC drivers registry [JDBCDRIVERS] . Official database server driver **SHOULD** be used if more than one drivers are available. JDBC driver name (and a sub-name if the name specifies the company, as with 'microsoft:sqlserver') is used.

### 7.3.3. Upgrade

If both old and new version of package require a database with the same `id` , then this database and its content need to be preserved. Controller **MUST** refuse to upgrade database to another database type.

All databases which are declared in old or new packages **MUST** be accessible during upgrade script invocation. Thus, application **MAY** perform database upgrades by issuing new database `id` in the package release which requires cross-database upgrade.

### 7.3.4. MySQL Specific Settings

This specification defines requirements type for privileges of MySQL user on database being allocated.

#### Example 23. Privileges for MySQL Database

```
<requirements>
  <db:db xmlns:db="http://apstandard.com/ns/1/db">
    <db:id>storage</db:id>
    <db:default-name>broombla</db:default-name>
    <db:can-use-tables-prefix>true</db:can-use-tables-prefix>
    <db:server-type>mysql</db:server-type>
    <db:server-min-version>4.1.0</db:server-min-version>

    <db:features xmlns:mysql="http://apstandard.com/ns/1/db/mysql">
      <mysql:privilege>Create_tmp_table_priv</mysql:privilege>
      <mysql:privilege disabled="true">Lock_tables_priv</mysql:privilege>
    </db:features>
  </db:db>
</requirements>
```

The names of MySQL privileges are specified in the db table of the mysql database. Controversial permissions MUST NOT be specified.

## 7.4. Apache Aspect

This aspect is to be used by web applications which use Apache-specific features. This aspect uses the `http://apstandard.com/ns/1/apache` XML namespace.

RELAX NG schema of Apache aspect:

```
namespace apache = "http://apstandard.com/ns/1/apache"

## Requirements
div {
  Requirement |= element apache:htaccess { empty }
  Requirement |= element apache:required-module { text }
}
```

This aspect declares two requirement types: Apache module requirement type and Apache `.htaccess` requirement type. Those requirements should be used only if a web application works exclusively with Apache due to some Apache-specific features.

### Example 24. Apache Module Requirement

```
<requirements>
  <apache:required-module xmlns:apache="http://apstandard.com/ns/1/apache">
    mod_python
  </apache:required-module>
</requirements>
```

Requirement of this type is satisfied when the specified Apache module is enabled for the application.

### Example 25. Apache `.htaccess` Requirement

```
<requirements>
  <apache:htaccess xmlns:apache="http://apstandard.com/ns/1/apache"/>
</requirements>
```

Requirement of this type is satisfied when the `.htaccess` processing is enabled for the application.

## 7.5. CGI Support

This aspect allows declaring CGI scripts in a package. This aspect uses the `http://apstandard.com/ns/1/cgi` XML namespace. Explicit handlers notation uses the `http://apstandard.com/ns/1/cgi/handlers` namespace.

RELAX NG schema of CGI URL handler type:

```
namespace cgi = "http://apstandard.com/ns/1/cgi"
namespace h   = "http://apstandard.com/ns/1/cgi/handlers"

## URL Handlers
div {
  UrlHandler |= element cgi:handler {
    ( element cgi:disabled { empty }
      | element cgi:extension { handlerType?, text }*
      | element cgi:all-files { handlerType? }
    )
  }

  UrlHandler |= element cgi:permissions {
    attribute writable { "true" }?,
    attribute readable { "false" }?
  }

  handlerType = attribute h:handler-type {
```

```
"executable" | "perl" | "php" | "python" | "ssi"  
}  
}
```

### Example 26. CGI URL Handlers

```
<cgi:handler  
  xmlns:cgi="http://apstandard.com/ns/1/cgi"  
  xmlns:h="http://apstandard.com/ns/1/cgi/handlers">  
  <cgi:extension h:handler-type="perl">pl</cgi:extension>  
  <cgi:extension h:handler-type="perl">cgi</cgi:extension>  
</cgi:handler>
```

```
<cgi:handler xmlns:cgi="http://apstandard.com/ns/1/cgi">  
  <cgi:all-files/>  
</cgi:handler>
```

A handler of this type requires that files with the specified extensions (if no extensions are specified, a single `cgi` extension is assumed) are handled by running them as CGI scripts. If the `all-files` option is declared, then all files under the current mapping are to be handled as CGI scripts.

Inner mappings inherit handlers from the outer mappings, so the presence of `cgi:disabled` disables handling of CGI in the given mapping.

Some web servers need additional information about programs to run CGIs. This information is optionally supplied in the `h:handler` attribute. Possible values of this attribute consist of several predefined strings, each denoting CGI handler of a particular type, namely:

- `executable` - CGI itself is an executable program and is to be executed *per se*
- `perl` - CGI is to be executed by Perl interpreter.
- `php` - CGI is to be executed by PHP CGI interpreter.
- `python` - CGI is to be executed by Python interpreter.
- `ssi` - CGI is to be executed by SSI preprocessor.

Web servers which do not need an additional information about CGI handlers should ignore the `h:handler` attributes.

### Example 27. CGI permission handler

```
<cgi:permissions writable="true">
```

```
<cgi:permissions readable="false">
```

The situation when a directory and files in the directory to which the given mapping maps should be writable by running CGI script is specified by using the `writable` attribute with the "true" value. The "false" value is default, so no explicit attribute for this is required.

The situation when files in the directory should be protected from reading by running CGI script is specified by the `readable` attribute with the "false" value. The "true" value is default, so no explicit attribute for this is required.

Inner mappings do not inherit permission handlers from the outer mappings.

## 7.6. Hardware Resources

RELAX NG schema of Hardware Resources aspect:

```
namespace hw = "http://apstandard.com/ns/1/hardware"
```

```
## Requirements
div {

  Requirement |= element hw:hardware {
    element hw:minimal { CPU?, RAM? }?,
    element hw:recommended { CPU?, RAM? }?
  }

  CPU = element hw:cpu { xsd:nonNegativeInteger }
  RAM = element hw:ram { xsd:nonNegativeInteger }
}
```

This aspect allows declaration of hardware requirements for application services. This aspect uses the `http://apstandard.com/ns/1/hardware` XML namespace.

### Example 28. Specifying the Minimum and Recommended Hardware Characteristics

```
<requirements>
  <hw:hardware xmlns:hw="http://apstandard.com/ns/1/hardware">
    <hw:minimal>
      <hw:cpu>200</hw:cpu>
      <hw:ram>50</hw:ram>
    </hw:minimal>
    <hw:recommended>
      <hw:cpu>600</hw:cpu>
      <hw:ram>100</hw:ram>
    </hw:recommended>
  </hw:hardware>
</requirements>
```

This aspect declares the following requirement types: the minimum and recommended amount of CPU and RAM resources required for service to work. Requirements of this type are satisfied if Controller is aware that necessary amount of resources is available. Required CPU resources are specified in megahertz, RAM resources are specified in megabytes.

## 7.7. Operating Environment

RELAX NG schema of Operating Environment aspect:

```
namespace env = "http://apstandard.com/ns/1/environment"

## Requirements
div {
  Requirement |= element env:environment {
    ( element env:x86 { empty } |
      element env:x86_64 { empty } )? ,

    ( element env:windows { empty | text } |
      element env:linux { empty | text } |
      element env:freebsd { empty | text } |
      element env:macos { empty | text } )?
  }
}
```

This aspect allows declaring operating environment requirements for platform-dependent applications. This aspect uses the `http://apstandard.com/ns/1/environment` XML namespace.

### Example 29. Specifying Required OS and Architecture

```
<requirements>
  <env:environment xmlns:env="http://apstandard.com/ns/1/environment">
    <env:x86/>
    <env:windows/>
  </env:environment>
</requirements>
```

This aspect declares the following requirement types: operating system, and operating system architecture. Requirements of this type are satisfied if the appropriate operating system and OS architecture are used for services instances. Requirements of this type are allowed only within top-level service.

Controller MAY honor exact operating system name, specified as a content of operating system class element.

## 7.8. Mail

RELAX NG schema of Mail aspect:

```
namespace mail = "http://apstandard.com/ns/1/mail"

Requirement |= element mail:mailbox {

  element mail:id { text },

  element mail:access {
    element mail:imap      { empty }? ,
    element mail:imap-ssl  { empty }? ,
    element mail:imap-tls  { empty }? ,
    element mail:pop3      { empty }? ,
    element mail:pop3-apop { empty }? ,
    element mail:pop3-tls  { empty }?
  },

  element mail:outgoing {
    element mail:smtp      { empty }? ,
    element mail:smtp-ssl  { empty }? ,
    element mail:smtp-tls  { empty }?
  }?
}

Requirement |= element mail:smtp { empty }
```

This aspect is to be used by applications implementing Mail User Agents, requiring access to results of mail delivery or sending mail. This aspect uses the `http://apstandard.com/ns/1/mail` XML namespace.

### Example 30. Requirements for mailbox access Requirement for sending emails

```
<requirements xmlns:mail="http://apstandard.com/ns/1/mail">
  <mail:mailbox>
    <mail:access>
      <mail:imap/>
      <mail:imap-ssl/>
      <mail:imap-tls/>
      <mail:pop3/>
      <mail:pop3-apop/>
    </mail:access>
    <mail:outgoing>
      <mail:smtp/>
    </mail:outgoing>
  </mail:mailbox>
</requirements>
```

```
<requirements xmlns:mail="http://apstandard.com/ns/1/mail">
  <mail:outgoing>
    <mail:smtp/>
    <mail:smtp-tls/>
  </mail:outgoing>
</requirements>
```

The following elements comprise the 'mailbox' requirement type:

id	Identifier of mailbox. This identifier will be used in environment variables passed to configuration scripts
----	--

---

access	A list of protocols that must be used for accessing mailbox
imap	IMAP version 4 revision 1 protocol as defined by RFC 3501
imap-ssl	IMAP version 4 revision 1 tunneled over SSL
imap-tls	IMAP version 4 revision 1 with Transport Layer Security enabled
pop3	POP3 protocol as defined by RFC 1939
pop3-apop	POP3 protocol with APOP extension
pop3-tls	POP3 protocol access with TLS enabled
outgoing	Requirement for access to outgoing mail server
smtp	Requirement for access to outgoing mail server over SMTP protocol as defined by RFC 5321
smtp-ssl	Requirement for access to outgoing mail server over SMTP with SSL tunnelling
smtp-tls	Requirement for access to outgoing mail server over SMTP protocol with TLS enabled

Requirement of this type is satisfied when the following is true:

New mailbox is created accessible over the specified protocols or existing mailbox is configured to be accessible over the specified protocols.

User authorized to accessing the mailbox is created or exists already.

If 'outgoing' element is used - the same user is authenticated by outgoing mail server.

Mail delivery for at least one email address is configured to the mailbox.

Outgoing mail server is accessible over the protocols specified within 'outgoing' element.

Controller MAY deallocate mailbox when service is cancelled.

The 'outgoing' requirement type demands access to outgoing mail server over SMTP protocol as defined by RFC 5321. Requirement of this type is satisfied when:

There is a new or existing user, that is authenticated by outgoing mail server.

Outgoing mail server supports the specified protocol(s)

### 7.8.1. Environment Variables

When a requirement is satisfied, the following environment variables must be passed to configuration scripts for 'mailbox' requirement:

- MAIL\_<id>\_IMAP\_HOST - FQDN or IP address of IMAP and IMAP with TLS server
- MAIL\_<id>\_IMAP\_PORT - port number of the IMAP and IMAP with TLS server
- MAIL\_<id>\_IMAP\_PORT\_SSL - port number of the IMAP over SSL server
- MAIL\_<id>\_IMAP\_MAILBOX - default mailbox name
- MAIL\_<id>\_POP3\_HOST - FQDN or IP address of POP3, APOP and POP3 with TLS server
- MAIL\_<id>\_POP3\_PORT - port number of POP3, APOP and POP3 with TLS server
- MAIL\_<id>\_USER - login of user, authorized to access the mailbox and outgoing mail server (if required)
- MAIL\_<id>\_PASSWORD - password of the user

- MAIL\_<id>\_EMAIL - primary email address, delivering into the specified mailbox
- MAIL\_<id>\_ALIAS\_<identifier> - aliases of the email address. 'identifier' must have different value for different aliases
- MAIL\_<id>\_SMTP\_HOST - FQDN or IP address of SMTP or SMTP with TLS server
- MAIL\_<id>\_SMTP\_PORT - port number of the SMTP and SMTP with TLS server
- MAIL\_<id>\_SMTP\_PORT\_SSL - port number of the SMTP over SSL server

When a requirement is satisfied, the following environment variables must be passed to configuration scripts for 'outgoing' requirement:

- MAIL\_SMTP\_HOST - FQDN or IP address of SMTP or SMTP with TLS server
- MAIL\_SMTP\_PORT - port number of the SMTP and SMTP with TLS server
- MAIL\_SMTP\_PORT\_SSL - port number of the SMTP over SSL server
- MAIL\_SMTP\_USER - login of user to be used to authenticate in SMTP server
- MAIL\_SMTP\_PASSWORD - password of the user

### 7.8.2. Upgrade

If both old and new versions of application require mailbox then the mailbox, its content and user access credentials MUST be preserved. If new version of application requires different set of mailbox access protocols, then mailbox MUST be accessible over new set of protocols during upgrade.

## References

### Satellite Specifications

[APS Categories] *APS Application Categories* [<http://apsstandard.com/r/doc/aps--application-categories.pdf>]. List of predefined APS categories.

### XML Technologies

[XMLNS] *Namespaces in XML (Second Edition)* [<http://www.w3.org/TR/REC-xml-names/>]. W3C Recommendation 16 Aug 2006.

[XMLLANG] *Extensible Markup Language (XML) 1.0 (Fourth Edition). 2.12 Language Identification* [<http://www.w3.org/TR/REC-xml/#sec-lang-tag>]. W3C Recommendation 16 August 2006.

[RNG] *RELAX NG specification* [<http://www.oasis-open.org/committees/relax-ng/spec.html>]. OASIS Committee Specification 3 December 2001

[RNC] *RELAX NG compact syntax specification* [<http://relaxng.org/compact.html>]. OASIS Committee Specification 21 November 2002

### Other

[HCARD] *hCard specification* [<http://microformats.org/wiki/hcard>]

[HCARD-PROFILE] *hCard Profile* [<http://www.w3.org/2006/03/hcard>] W3C experimental XMDP profile for the hCard specification.

[JDBC DRIVERS] *JDBC drivers registry* [<http://developers.sun.com/product/jdbc/drivers>]

[ZIP] *ZIP specification* [<http://www.info-zip.org/pub/infozip/doc/zip>]

[Langs] *Wikipedia list of programming languages* [[http://en.wikipedia.org/wiki/Alphabetical\\_list\\_of\\_programming\\_languages](http://en.wikipedia.org/wiki/Alphabetical_list_of_programming_languages)]

[RFC 2119] *RFC 2119, BCP 14: Key words for use in RFCs to Indicate Requirement Levels* [<http://www.ietf.org/rfc/rfc2119.txt>]

[RFC 3920] *Extensible Messaging and Presence Protocol (XMPP): Core* [<http://www.ietf.org/rfc/rfc3920.txt>]

[RFC 2872] *Application and Sub Application Identity Policy Element for Use with RSVP* [<http://www.ietf.org/rfc/rfc2872.txt>]

[RFC 2822] *RFC 2822: Internet Message Format* [<http://www.ietf.org/rfc/rfc2822.txt>]

[RFC 1035] *RFC 1035: DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION* [<http://www.ietf.org/rfc/rfc1035.txt>]

[RFC 3066] *RFC 3066: Tags for the Identification of Languages* [<http://www.ietf.org/rfc/rfc3066.txt>]

[RFC 1738] *RFC 1738: Uniform Resource Locators (URL)* [<http://www.ietf.org/rfc/rfc1738.txt>]

[ISO 639] *ISO 639: Codes for the Representation of Names of Languages* [<http://www.loc.gov/standards/iso639-2/>]

[ISO 3166] *ISO 3166 Code Lists* [[http://www.iso.org/iso/country\\_codes.htm](http://www.iso.org/iso/country_codes.htm)]

[URI Templates] *URI Templates* [<http://bitworking.org/projects/URI-Templates/>]