

---

# Application Packaging Standard - Package Format Specification

1.0

Copyright © 1999, 2008 Parallels, Inc

All rights reserved.

## Table of Contents

1. Introduction .....	1
2. Conventions .....	3
3. Document Structure .....	3
4. Basic Package Format .....	3
4.1. File format .....	3
4.2. Files .....	3
4.3. Metadata Descriptor .....	3
4.4. Requirements .....	15
4.5. URL mapping .....	16
4.6. Configuration script .....	18
5. Points of Extensibility .....	20
5.1. Requirement types .....	21
5.2. URL handlers types .....	21
5.3. Additional files .....	21
5.4. Environment variables .....	21
5.5. Additional scripts .....	21
5.6. Configuration script language .....	21
5.7. Rules .....	21
6. Common aspects .....	21
6.1. PHP aspect .....	21
6.2. ASP.NET aspect .....	25
6.3. Database Aspect .....	26
6.4. Apache Aspect .....	27
6.5. CGI Aspect .....	28
References .....	29

## 1. Introduction

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

The package is a ZIP file that contains

main package metadata file `APP-META.xml`

additional metadata files, such as icons and pictures, referenced from `APP-META.xml`

management scripts in the `scripts/` directory. The `scripts/configure` script performs application-specific tasks during installation, upgrade, configuration and uninstallation of the application.

Here is a structure of typical package.

```
APP-META.xml      # Metadata container. XML file.
scripts/
  configure       # This script will be invoked when application
```

Application Packaging Standard  
- Package Format Specification

---

```
...           # instance is being 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/      # Metadata may specify Web content directories
  index.php
  logo.png
  ...
```

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

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

  <!-- common properties shared by all packages -->

  <name>phpBB</name>
  <version>2.0.22</version>
  <release>6</release>
  <homepage>http://phpbb.com/</homepage>
  <description>...</description>

  ...

  <changelog>
    <version version="2.0.22" release="6">
      <entry>Fixed bug in ...</entry>
    </version>
  </changelog>

  ...

  <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>

  <!-- Mapping URLs to the files and URL handlers -->

  <mapping url="/" path="htdocs">
    <php:handler/>
  </mapping>

</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 web 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

A package is a ZIP file [ZIP] file 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 Descriptor

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.

RELAX NG schema of metadata descriptor:

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

grammar {
```

```
start = Application

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

DefinedByAspect = AnyElement

Requirement = DefinedByAspect

UrlHandler = DefinedByAspect

Permission = DefinedByAspect

Setting = element sa:setting {
    attribute id { text },
    attribute global { "true" }?,
    attribute installation-only { "true" }?,
    element sa:name {
        attribute xml:lang { text }?,
        xsd:string
    }*,
    element sa:description {
        attribute xml:lang { text }?,
        xsd:string
    }*,
    element sa:error-message {
        attribute xml:lang { text }?,
        xsd:string
    }*,
    ( (
        attribute type { "boolean" },
        attribute default-value { xsd:boolean }?
    )
    | (
        attribute type { "string" | "password" },
        attribute min-length { xsd:nonNegativeInteger }?,
        attribute max-length { xsd:nonNegativeInteger }?,
        (attribute regex { xsd:string } | attribute charset { xsd:string })?,
        attribute default-value { xsd:string }?
    )
    | (
        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 { xsd:string }?
    )
}
```

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

```
SettingGroup = element sa:group {  
  element sa:name {  
    attribute xml:lang { text }?,  
    xsd:string  
  }*,  
  Setting+  
}
```

```
Requirements = element sa:requirements {  
  element sa:choice {  
    element sa:requirements {  
      attribute id { text },  
      Requirement+  
    }+  
  }*,  
  Requirement*  
}
```

```
SubMapping = element sa:mapping {  
  attribute url { text },  
  (attribute path { xsd:string { minLength = "1" } } | attribute virtual { "v"  
  
  (UrlHandler|Permission)*,  
  
  SubMapping*  
}
```

```
Mapping = element sa:mapping {  
  attribute url { text },  
  (attribute path { xsd:string { minLength = "1" } } | attribute virtual { "v"  
  
  (UrlHandler|Permission)*,  
  
  SubMapping*  
}
```

```
Application = element sa:application {  
  element sa:name { text },  
  element sa:packager-uri { xsd:anyURI }?,
```

```
element sa:version { text },
element sa:release { text },
element sa:homepage { xsd:anyURI }?,
element sa:package-homepage { xsd:anyURI }?,
element sa:default-prefix { text }?,
element sa:summary {
  attribute xml:lang { text }?,
  text
}*
element sa:description {
  attribute xml:lang { text }?,
  text
}*
element sa:icon {
  attribute path { text }
}?,
element sa:screenshot {
  attribute path { text },
  element sa:description {
    attribute xml:lang { text }?,
    text
  }+
}*
# Either URL or file and name of license are allowed.
element sa:license {
  attribute must-accept { xsd:boolean },

  element sa:text {
    attribute xml:lang { text }?,
    element sa:name { text }?,
    ( element sa:url { xsd:anyURI }
    | element sa:file { text } )
  }+
}?,
element sa:configuration-script-language { text }?,

# Upgrades and patches specification

# Versions which may be patched by this package
(
  # Any version starting from the specified
  element sa:patchable-from {
    attribute version { text },
    attribute release { text }
  }
|
  # Explicit versions enumeration
  element sa:patches {
    attribute version { text },
    attribute release { text }
  }+
)?,

# Versions which may be upgraded by this package
(
  # Any version starting from the specified
  element sa:upgradable-from {
    attribute version { text },
```

```
        attribute release { text }
    }
    |
    # Explicit versions enumeration
    element sa:upgrades {
        attribute version { text },
        attribute release { text }
    }+
    )?,

    element sa:changelog {
        element sa:version {
            attribute version { text },
            attribute release { text },
            element sa:entry {
                attribute xml:lang { text }?,
                text
            }+
        }*
    },
    element sa:entry-points {
        element sa:entry {
            element sa:path { xsd:anyURI },
            element sa:label {
                attribute xml:lang { text }?,
                text
            }+,
            element sa:description {
                attribute xml:lang { text }?,
                text
            }*
        }+
    }?,
    element sa:installed-size { xsd:nonNegativeInteger }?,
    element sa:categories {
        element sa:category { text }+
    }?,
    element sa:languages {
        element sa:language { xsd:string { pattern = "[a-z]{2,3}" } }+
    }?,
    element sa:settings {
        Setting*,
        SettingGroup*
    },
    Requirements,
    Mapping
}
}
```

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

Metadata schema contains several places where arbitrary elements may be added: requirements may be added to the requirements elements, and URL handlers may be added to the mapping elements. Allowed types of requirements and URL handlers are described in aspects. Requirements and URL handlers structure 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 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 standard XML `xml:lang` attribute: every localizable string has optional attribute `xml:lang`:

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

String without explicit `xml:lang` is always required.

The following basic items are described in the metadata:

#### 4.3.1. Package name

```
<sa:name>phpbb</sa:name>
```

Free-formed string specifies user-visible name of web 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.

#### 4.3.2. Packager URI

```
<sa:packager-uri>uuid:15d041e8-34c6-409a-b165-3290d2c9d599</sa:packager-uri>
```

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 `packager-uri` to the one which specifies to support smooth upgrade path for the packages which did not use `packager-uri` from the beginning.

#### 4.3.3. Package version

```
<sa:version>2.0.22</sa:version>
<sa:release>6</sa: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's version-release approach, application version and package release are separated to ease parsing.

#### 4.3.4. Homepage

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

URL of the web application official site.

#### 4.3.5. Package homepage

```
<sa:package-homepage>http://swsoft.com/</sa:package-homepage>
```

URL of the application packager official site.

#### 4.3.6. Default Installation Prefix

```
<sa:default-prefix>/forum/</sa:default-prefix>
```

Relative URL with the default path where the application is supposed to be installed on domain.

#### 4.3.7. Summary

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

Single-sentence summary of the package for end users.

#### 4.3.8. Description

```
<sa: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.  
</sa:description>  
<sa:description xml:lang="it-IT">...</sa:description>
```

One-paragraph description of the package for end users.

#### 4.3.9. Icon

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

Icon may be provided to be displayed in GUI for the application. The path attribute MUST contain full path in archive to the 64x64 pixels image file. Icon MUST be in JPEG, PNG or TIFF formats.

#### 4.3.10. Screenshots

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

Several screenshots with descriptions may be provided. The path attribute MUST contain full path in archive to the image. Images MUST be in JPEG, PNG or TIFF formats.

#### 4.3.11. License

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

```
<sa:file>licenses/gplv2-de_DE.txt</sa:file>
</sa:text>
<sa:license>
```

or

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

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 to the full text of the license. The `file` element MUST be a full path in the archive to the existing file.

#### 4.3.12. Configuration script language

```
<sa:configuration-script-language>php</sa:configuration-script-language>
```

Interpreter to be used to run configuration script. Valid interpreters are described in the specification aspects. This specification defines a set of php, jscript and vbscript interpreters.

#### 4.3.13. Versions of package which can be updated to the current package

Two update modes are supported:

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

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

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

Package may document which versions it can update by either specifying minimal updatable version, or enumerating all the supported updatable versions.

```
<sa:upgradable-from version="1.0" release="1"/>
<sa:patchable-from version="2.0" release="1"/>
```

Minimal version of the package from which the current package can update. Such specification means that:

- If installed version is greater than or equal to 1.0.1 and less than 2.0.1, upgrade is possible.
- If installed version is greater than or equal to 2.0.1, patch is possible.

```
<sa:upgrades version="1.0" release="1"/>
<sa:upgrades version="2.0" release="0"/>
<sa:patches version="2.0" release="1"/>
<sa:patches version="2.0" release="2"/>
```

Exact versions of package which may be updated by package. Multiple specification allowed.

```
<sa:upgradable-from version="1.0" release="1"/>
<sa:patches version="2.0" release="1"/>
<sa:patches version="2.0" release="2"/>
```

Combinations of minimal and exact updatable versions specifications for different update modes are allowed.

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

#### 4.3.14. Changelog

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

#### 4.3.15. Entry points

```
<sa:entry-points>
  <sa:entry>
    <sa:path>/admin</sa:path>
    <sa:label>Administrative interface</sa:label>
  </sa:entry>
  <sa:entry>
    <sa:path>/guest</sa:path>
    <sa:label>Guest view</sa:label>
    <sa:label xml:lang="de-DE">...</sa:label>
    <sa:description>View the gallery as user not logged in</sa:description>
  </sa:entry>
</sa:entry-points>
```

If a web application contains entry points except the main one (base URL of web application), the packager may declare it using the entry points extension.

When a Controller installs a application with entry points to a site, it MAY provide user with the choice what entry points to show in control panel interface and where. Installed entry points MUST point to the base URL of application + path of entry point.

A Controller SHOULD use provided label and description. A Controller should use the icon of application, if it needs an icon to show entry point in user interface.

#### 4.3.16. Categories

```
<sa:categories>
  <sa:category>blog</sa:category>
  <sa:category>cms</sa:category>
</sa:categories>
```

Package may include a set of categories. Category is just 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.

Several categories are defined in this specification and recommended to use. However, this list of categories is not normative, and any other categories may be used.

album  
blog  
cms  
communication  
e-commerce  
forum  
groupware  
other  
portal  
statistics  
wiki

#### 4.3.17. Languages

```
<sa:languages>  
  <sa:language>en</sa:language>  
  <sa:language>de</sa:language>  
  <sa:language>ru</sa:language>  
</sa: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].

#### 4.3.18. Installed size

```
<sa:installed-size>5242880</sa:installed-size>
```

Package may include a declaration of approximate size of a single application instance, to help Controllers estimate the disk space resources needed to create instance. The declared size is in bytes.

#### 4.3.19. Application settings

Applications often need additional parameters for successful installation and configuration. While most of the questions asked during conventional web application installation are related to various resources and answers may be provided by a Controller without user intervention, some settings need to be entered by user.

Settings are declared in the `settings` element in basic metadata. Settings may be grouped by the `group` element. Each group has a name declared by the `name` element and a list of settings. Groups may not nest.

Each `setting` element represents the single setting to be asked from user.

Settings may be regular/installation-only, and local/global (hence, four types of settings: regular local, regular global, installation-only local and installation-only global).

Regular settings are settings which need to be set up during installation and may be reconfigured later. This usually includes settings which are stored in configuration files. This type is by default.

Installation-only settings need to be set up during installation, but 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="true"` attribute.

Local settings are set during application instantiation. This type is by default.

Global settings affect all instances of application and SHOULD be set prior to application instantiation. Such settings are marked with the `global="true"` attribute.

When global regular setting is changed, all instances of respective package need to be reconfigured immediately.

When application is updated, transformations of settings with the same id from 'installation-only' to 'regular' and from 'local' to 'global' classes are prohibited to protect application instance's settings values.

For patching, it is also prohibited to introduce new 'installation-only' settings without default values, as this prevents from unattended patch installation.

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 for the setting.
- Data type of setting (string, number, enum, etc.).

Settings and groups are listed in the order suggested to be used in interface.

Controller **MUST** keep state of application settings values to pass them to the configuration/update scripts.

### 4.3.19.1. Data types

Settings are typed, control panels **SHOULD** use the type information to validate input. The optional error-message element contains error message to be presented to user when the corresponding setting is not validated. The following types are defined:

Name	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: . * + ? meta symbols, ( ) grouping, [ ] character classes (only individual characters, ranges and class negation ^), { , N }, { M , }, { M , N } 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	min	Minimum acceptable real value (inclusive).
		max	Maximum acceptable real value (inclusive).

Name	Values	Optional restrictions
	value is less than 2 <sup>53</sup> , and e is an integer 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	

### 4.3.19.2. Predefined settings identifiers

There are settings which are often required by applications. Several identifiers are predefined in this specification to give implementers a way to create better interfaces (predefined settings may be used by control panel to provide values without asking user).

The following identifiers are predefined:

`title`. Title of application instance.

`admin_name`. Name of user administering application instance.

`admin_password`. Password of administrative account of application instance.

`admin_email`. Email of administrative account of application instance.

`locale`. Locale of application instance.

#### 4.3.19.2.1. Example usage of predefined identifiers

```
<sa:settings>
  <sa:group>
    <sa:name>Administrator's preferences</sa:name>
    <sa:setting id="admin_name" type="string" default-value="admin">
      <sa:name>Administrator's login</sa:name>
    </sa:setting>
    <sa:setting id="admin_password" type="password">
      <sa:name>Password</sa:name>
    </sa:setting>
    <sa:setting id="admin_email" type="email">
      <sa:name>Administrator's email</sa:name>
    </sa:setting>
  </sa:group>
</sa:settings>
```

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

It is **RECOMMENDED** to use the `enum` type for the `locale` setting to declare all languages supported by the application.

It is **RECOMMENDED** to use the `string` type for the `title` and `admin_name` settings.

It is **RECOMMENDED** to use the `password` type for the `admin_password` setting.

It is **RECOMMENDED** to use the `email` type for the `admin_email` setting.

## 4.4. Requirements

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

### 4.4.1. Example

```
<sa:requirements>
  <php:version min="5.0.2" />
  <sa:choice>
    <sa: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>
    </sa:requirements>
    <sa: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>
    </sa:requirements>
  </sa:choice>
</sa:requirements>
```

### 4.4.2. Structure

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 is described as XML element valid according to the schema.

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

Requirements are placed to the `requirements` element in the `APP-META.xml` file.

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.

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

Controller has to ensure the logical truth of a complex requirement (this means not all individual requirements need to be satisfied).

### 4.4.3. Environment variables

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

#### 4.4.4. Patch

No changes in application instance resources are allowed for patches. Therefore, Controller **MUST** preserve all resources allocated for previous version of instance and may not even try to satisfy requirements of new version.

#### 4.4.5. Upgrade

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.

### 4.5. 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.

#### 4.5.1. Example

```
<sa:mapping url="/" path="htdocs"
  xmlns:sa="http://apstandard.com/ns/1"
  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>

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

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

```
<sa:mapping url="/" xmlns:sa="http://apstandard.com/ns/1">
  <sa:mapping url="foo/bar"/>
  <sa:mapping url="foo/bar/baz"/>
</sa:mapping>
```

is prohibited, and **MUST** be rewritten as

```
<sa:mapping url="/" xmlns:sa="http://apstandard.com/ns/1">
  <sa:mapping url="foo/bar">
    <sa:mapping url="baz"/>
  </sa:mapping>
</sa:mapping>
```

All urls in mappings are relative to the parent URL mapping. Absolute URLs are **PROHIBITED** in the url attributes, except the root mapping, where url **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 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 does not have an explicit `path` attribute nor the `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

```
<sa:mapping url="/" path="htdocs" xmlns:sa="http://apstandard.com/ns/1">
  <sa:mapping url="foo/bar">
    <sa:mapping url="baz"/>
    <sa:mapping url="quux" path="somedir"/>
  </sa:mapping>
</sa: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 `'htdocs/foo/bar/quux'` is associated with the `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 associated directory too.

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

Controller MUST use mapping information for deployment and upgrade of application's instance files.

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.

```
<sa: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>
</sa:mapping>
```

#### 4.5.2. Patch

No changes in mapping structure are allowed for patches. Controller MUST leave mapping as it was for previous version of application. File overwriting semantics is determined in Upgrade.

#### 4.5.3. Upgrade

During upgrade 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 on application installation.

#### 4.5.4. Environment Variables

For each mapping, except 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:

```
<sa:mapping url="/" path="htdocs" xmlns:sa="http://apstandard.com/ns/1">
  <sa:mapping url="foo/bar">
    <sa:mapping url="baz"/>
    <sa:mapping url="quux" path="somedir"/>
  </sa:mapping>
</sa:mapping>
```

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

```
WEB_example_DIR=/var/www/vhosts/domain.name/public_html/example/htdocs
WEB_example_foo_bar_DIR=/var/www/vhosts/domain.name/public_html/example/htdocs/
WEB_example_foo_bar_baz_DIR=/var/www/vhosts/domain.name/public_html/example/htd
WEB_example_foo_bar_quux_DIR=/var/www/vhosts/domain.name/public_html/example/so
```

#### 4.6. Configuration script

Package may include the web application configuration script. This script will be invoked during adding application to a site, updating application, changing settings of application and removing application from the site.

Configuration script **MUST** be named `configure` and reside in the `scripts/` directory in the Package root directory.

The script file and script language declaration in package metadata **MUST** be either both included in the package or both omitted from the package.

Configuration script may be written in any programming language specified in the Basic metadata section.

Configuration script **MUST** be run on the host where the application is being installed on, with the permissions of user which owns this application instance, so only unprivileged actions may be performed in the configuration script.

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's current working directory.

If a package does not contain configuration script, Controller **MUST** skip all configuration script invocations.

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 of the script should be captured to log error in this case.

#### 4.6.1. Configuration script actions

Configuration script is invoked when package is installed, configured, patched, upgraded or uninstalled.

##### 4.6.1.1. Installing Application on a Site

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

```
install
```

##### 4.6.1.2. Upgrading Application

Configuration script is invoked when application instance is being updated. By the moment of invocation all resources needed by the new version of application **MUST** be already allocated. The script of new application version is invoked with the following arguments, where `<old version>` is the old version, and `<old release>` is the old release of the application being upgraded or patched.

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

##### 4.6.1.3. Changing settings

Configuration script is invoked when application is being configured (this does not include installing and upgrading). The script is invoked with the following arguments:

```
configure
```

##### 4.6.1.4. Uninstalling Application From a Site

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

```
remove
```

#### 4.6.2. 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 is used as the encoding. 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.

##### 4.6.2.1. Full instance URL

Full URL specifying where the application is to be installed, 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`, as defined by [RFC 1738].

#### 4.6.2.2. Settings

For each application setting declared in package, the corresponding environment variable `SETTINGS_<id>` MUST be passed on to the installation script, where the `<id>` is the value of the `id` attribute of the setting description.

During the installation, all the settings provided in metadata MUST be passed to the configuration script.

During the reconfiguration, all the settings declared in metadata, except marked as installation-only, MUST be passed to the configuration script.

During the application instance upgrade and patch, all settings of the new version of package which exist in old version MUST be set to corresponding values from the old instance. If validator of setting 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.

During the uninstallation, all the settings declared in metadata, except marked as installation-only, MUST be passed to the configuration script.

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).

#### 4.6.2.3. 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 each aspect defines. See Points of extensibility - Environment variables.

### 5. 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 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 instantiation method.

- Aspect MUST declare rules of how Controllers are to process additional data supplied in package.

### 5.1. 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.

### 5.2. URL handlers types

Each aspect may declare several URL handlers types. Every requirement handler type declares how to declare rules 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.

### 5.3. Additional files

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

### 5.4. 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.

### 5.5. 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.

### 5.6. 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]

### 5.7. Rules

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

## 6. 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.

### 6.1. 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.

### 6.1.1. 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.

#### 6.1.1.1. Version requirement type

RELAX NG schema of PHP version requirement type:

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

element php:version {
  attribute min { text }?,
  attribute max-not-including { text }?
}
```

Here is an example of 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.

#### 6.1.1.2. Extension requirement type

RELAX NX schema of PHP extension requirement type:

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

element php:extension { text }
```

Here is an example of 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.

#### 6.1.1.3. Function requirement type

RELAX NG schema of PHP function requirement type:

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

element php:function { text }
```

Here is an example of PHP function requirement:

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

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

#### 6.1.1.4. PHP settings requirement types

RELAX NG schema of PHP allow\_url\_fopen requirement type:

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

```
element php:allow-url-fopen { xsd:boolean }
```

RELAX NG schema of PHP file\_uploads requirement type:

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

```
element php:file-uploads { xsd:boolean }
```

RELAX NG schema of PHP magic\_quotes\_gpc requirement type:

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

```
element php:magic-quotes-gpc { xsd:boolean }
```

RELAX NG schema of PHP register\_globals requirement type:

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

```
element php:register-globals { xsd:boolean }
```

RELAX NG schema of PHP safe\_mode requirement type:

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

```
element php:safe-mode { xsd:boolean }
```

RELAX NG schema of PHP short\_open\_tag requirement type:

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

```
element php:short-open-tag { xsd:boolean }
```

Here are examples of 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`, `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.

### 6.1.1.5. PHP limits requirement types

RELAX NG schema of PHP memory\_limit requirement type:

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

```
element php:memory-limit { xsd:integer }
```

RELAX NG schema of PHP max\_execution\_time requirement type:

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

```
element php:max-execution-time { xsd:integer }
```

RELAX NG schema of PHP post\_max\_size requirement type:

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

```
element php:post-max-size { xsd:integer }
```

Here are examples of 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 (in bytes and seconds).

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

### 6.1.2. URL handlers

RELAX NG schema of PHP URL handler type:

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

```
element php:handler {  
  (element php:disabled { empty }  
  |element php:extension { text }*)  
}
```

Here is an example of PHP URL handlers:

```
<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, the 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.

### 6.1.3. Permissions

RELAX NG schema of PHP permissions handler type:

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

```
element php:permissions {  
  attribute writable { "true" }?,  
  attribute readable { "false" }?  
}
```

Here is an example of PHP permission handlers:

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

The situation that 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 by default, so no explicit attribute for this is required.

The situation that 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 by default, so no explicit attribute for this is required.

### 6.1.4. Configuration Script Language

This aspect defines the `php` identifier for use 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.

### 6.2. ASP.NET aspect

This aspect is to be used by web applications which use ASP.NET.

This aspect uses the `http://apstandard.com/ns/1/aspnet` XML namespace.

#### 6.2.1. Requirement types

This aspect declares a single requirement type: ASP.NET version requirement.

RELAX NG schema of ASP.NET version requirement type:

```
namespace aspnet = "http://apstandard.com/ns/1/aspnet "  
  
element aspnet:version {  
  "1.0" | "2.0"  
}
```

Requirement of this type is satisfied when the application is installed in virtual directory with the specified version of ASP.NET enabled.

#### 6.2.2. URL handler type

RELAX NG schema of ASP.NET URL handler type:

```
namespace aspnet = "http://apstandard.com/ns/1/aspnet "  
  
element aspnet:handler {  
  (element aspnet:disabled { empty }  
  | element aspnet:extension { text }*)  
  )  
}
```

Here is an example of 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.

#### 6.2.3. Permissions

RELAX NG schema of ASP.NET file permissions handler type:

```
namespace aspnet = "http://apstandard.com/ns/1/aspnet "  
  
element aspnet:permissions {  
  attribute writable { "true" }?  
}
```

Here is an example of ASP.NET permission handler:

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

The situation that 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 by default, so no explicit attribute for this is required.

#### 6.2.4. Configuration script language

This aspect defines `javascript` and `vbscript` identifiers for use by configuration scripts (written in JScript and VBScript correspondingly).

### 6.3. Database Aspect

This aspect is to be used by web applications which need a database to operate.

This aspect uses the `http://apstandard.com/ns/1/db` namespace.

#### 6.3.1. Requirements Types

##### 6.3.1.1. Database requirement type

RELAX NG schema of database requirement type:

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

element d:db {
  element d:id { text },
  element d:default-name { text }?,
  element d:can-use-tables-prefix { xsd:boolean },
  element d:server-type { text },
  element d:server-min-version { text }
}
```

This requirement type is to be used when application needs a database. The following elements comprise the database requirement type:

<code>id</code>	Identifier of the database. This identifier will be used in environment variables passed to configuration scripts.
<code>default-name</code>	Optional. Proposed name of a database. It is not guaranteed that the allocated database will use this name.
<code>can-use-tables-prefix</code>	This element must have the <code>true</code> value only if an application is able to cope with sharing a database with another applications by using prefixed tables. Otherwise, is <code>false</code> .
<code>server-type</code>	This element must be one of the database server identifiers described below.
<code>server-min-version</code>	This element describes the minimal acceptable version of database server software.

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.

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 choices or in a choice and outside it simultaneously.

#### 6.3.1.1.1. 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 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.

#### 6.3.1.1.2. 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 driver are available. JDBC driver name (and a sub-name if the name specifies the company, as with 'microsoft:sqlserver') is used.

#### 6.3.1.1.3. 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.

### 6.4. 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.

### 6.4.1. Requirement types

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.

#### 6.4.1.1. Module requirement type

RELAX NG schema of Apache module requirement:

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

element apache:required-module {
  text
}
```

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

RELAX NG schema of Apache .htaccess requirement:

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

element apache:htaccess {
  empty
}
```

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

### 6.5. CGI Aspect

This aspect allows declaring CGI scripts in 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.

#### 6.5.1. URL handler

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"

handlerType = attribute h:handler-type {
  "executable" |
  "perl" |
  "php" |
  "python" |
  "ssi"
}

start = element cgi:handler {
  (element cgi:disabled { empty }
  |element cgi:extension {
    handlerType?,
    text
  }*)
  |element cgi:all-files {
    handlerType?
  }
)
```

```
}
```

Here are examples of CGI URL handler:

```
<cgi:handler>  
  <cgi:extension h:handler-type="perl">pl</cgi:extension>  
  <cgi:extension h:handler-type="perl">cgi</cgi:extension>  
</cgi:handler>
```

```
<cgi:handler>  
  <cgi:all-files/>  
</cgi:handler>
```

A handler of this type requires that files with the specified extensions (if no extensions are specified, the 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 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.

## References

### 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

[JDBCDRIVERS] *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 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)]