
Parallels

Application Packaging Standard 1.1

Developer's Guide

Revision 1.0.01



(c) 1999-2009

Contents

Introduction	3
About Application Packaging Standard.....	4
About This Guide.....	4
Useful Links.....	5
Typographical Conventions.....	5
Feedback.....	6
Packaging Overview	7
Before Creating Package	8
Defining Application Technologies.....	8
Creating Application Configuration Scripts.....	9
Creating Application Package	11
Creating Application Package Physical Structure.....	12
Creating Metadata File.....	13
Common Application Properties.....	13
Application Services.....	14
Validating Application Package	17
APSLint Utility.....	18
Managing APSLint for Windows.....	18
Managing APSLint for Linux/Unix.....	18
Controller Operations on Packages.....	20
Creating Application Instance.....	21
Updating Application Instance.....	21
Removing Application Instance.....	22
Sample Metadata File	23
Sample Environment Variables	29
Index	31

CHAPTER 1

Introduction

The Application Packaging Standard is an application packaging format designed to help implement a Software-as-a-Service (SaaS) business model.

This chapter provides general information about Application Packaging Standard and this guide.

In This Chapter

About Application Packaging Standard.....	4
About This Guide	4
Useful Links.....	5
Typographical Conventions.....	5
Feedback	6

About Application Packaging Standard

Because of the extensive development and integration requirements, hosting providers today offer a limited variety of hosted applications. As a result, independent software vendors have little incentive to create hosted applications that they cannot sell, and customers suffer from limited application choices.

In response to this, *Application Packaging Standard* (APS) was developed. It is an open format of packaging and managing web applications that will make it easier for the whole hosting industry to take advantage of the expanding *Software-as-a-Service* (SaaS) market.

By defining such open format, APS increases business opportunities for the entire hosting ecosystem bringing together application vendors and hosting service providers. By implementing APS, application vendors get access to a vast sales and marketing channel of APS-enabled hosting providers. In turn, hosting services providers, by implementing APS, gain access to a great variety of APS applications.

Application vendors implement APS by creating application packages in full compliance with the rules defined by the standard.

About This Guide

The purpose of this document is to provide web application developers and application distributors with recommendations for creating application packages in accordance with the *Application Packaging Standard*. The guide includes two appendixes with useful samples.

Abbreviations, Definitions and Conventions

- *APS* is used instead of Application Packaging Standard in some long sentences where using it will not change the meaning of the sentence.
- *Web site* is a collection of web pages, typically common to a particular domain name or sub-domain on the World Wide Web on the Internet.
- *Control panel* or *CP* is hosting provider software designed for managing user accounts, web sites and site applications.
- *Site application* or *application* is a Web application that can be hosted on a web site.
- *Application Package* or *Package* is a site application in its distribution format which includes all application and application-related files created, structured and packed according to the APS.
- *Application Instance* is a site application installed from application package on a particular web site and accessible via a unique URL.
- *Controller* is APS engine, a control panel part responsible for processing application packages and instances.

- *Aspect* is an additional specification that declares how to describe in the application package metadata: file technologies used by an application, application management scripts, and how these descriptions must be processed by Controllers. The technologies include any software or executable code.
- *Qualified technology* means any software or executable code described in an aspect included in the *APS: Package Format Specification*.
- *XPath* notation is used to describe the structure of XML documents, and to refer to XML elements and attributes in the text.
- *APS: Package Format Specification* is referred to as *the Specification*.

Useful Links

Official APS site. Latest news and relevant information on Application Packaging Standard development.

<http://www.apsstandard.com/>

APS: Package Format Specification

<http://www.apsstandard.com/r/doc/package-format-specification/index.html>

APS: Application Packaging Developer's Guide

<http://www.apsstandard.com/r/doc/aps-packaging-dev-guide/index.htm>

XML Technologies:

XPath Homepage

<http://www.w3schools.com/xpath/default.asp>

RELAX NG Compact Syntax Tutorial

<http://www.relaxng.org/compact-tutorial-20030326.html>

Typographical Conventions

Before you start using this guide, it is important to understand the documentation conventions used in it.

The following kinds of formatting in the text identify special information.

<u>Formatting convention</u>	<u>Type of Information</u>	<u>Example</u>
Special Bold	Items you must select, such as menu options, command buttons, or items in a list.	Go to the System tab.
	Titles of chapters, sections, and subsections.	Read the Basic Administration chapter.

<i>Italics</i>	Used to emphasize the importance of a point, to introduce a term or to designate a command line placeholder, which is to be replaced with a real name or value.	The system supports the so called <i>wildcard character</i> search.
Monospace	The names of commands, files, and directories.	The license file is located in the http://docs/common/licenses directory.
Preformatted	On-screen computer output in your command-line sessions; source code in XML, C++, or other programming languages.	<pre># ls -al /files total 14470</pre>
Preformatted Bold	What you type, contrasted with on-screen computer output.	<pre># cd /root/rpms/php</pre>
CAPITALS	Names of keys on the keyboard.	SHIFT, CTRL, ALT
KEY+KEY	Key combinations for which the user must press and hold down one key and then press another.	CTRL+P, ALT+F4

Feedback

If you have found a mistake in this guide, or if you have suggestions or ideas on how to improve this guide, please send your feedback using the online form at <http://www.parallels.com/en/support/usersdoc/>. Please include in your report the guide's title, chapter and section titles, and the fragment of text in which you have found an error.

Packaging Overview

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

To create and test an application package in compliance with APS, perform the following steps:

Note: it is supposed that you already have a working application.

- 1** Define technologies used by your application. Used technologies may be scripting languages, specific server modules, databases and the like. Application technologies form the list of requirements to be satisfied to provision application. Consider how these technologies will be recognized by Controllers. For details, refer to the Before Creating Application Package chapter (on page 8) further in this guide.
- 2** Define whether your application requires additional configuration. Some applications may require additional configuration on its installation, reconfiguration, upgrade or removal. Write scripts that will setup application instances. For details, refer to the Creating Application Configuration Scripts section (on page 9) further in this guide. This step is regulated by the Specification in sections 5.3.2 Configuration script and 6.5. Additional scripts.
- 3** Create an application package structure. Package structure must comply with the content organization requirements declared in the Specification. Check your package structure before archiving and update the application content if needed. For details on the requirements, refer to the Creating Application Package Physical Structure section (on page 12) further in this guide. This step is regulated by the Specification in sections 1. Introduction, 4. Basic Package Format and 6.5. Additional scripts.
- 4** Create a metadata file. Each package must contain a well-formed XML file that includes all the metadata required to manage the application. For details, refer to the Creating Metadata File section (on page 13) further in this guide. This step is regulated by the Specification in sections 4.3. Metadata File, 5. Metadata Descriptor and 6. Points of Extensibility.
- 5** Archive the application package files. The package should be a ZIP archive with the `.app.zip` extension (according to the 4.1. File format section of the Specification).
- 6** Validate the application package. The package validation may comprise two steps: checking whether package structure conforms to APS and then checking whether package is properly managed by Controllers. Package structure can be validated using *APSLint* command-line utility. If the application package is APS-compatible, then you can check how it is managed by Controllers. To do this, examine the Controller operations that require application package data. For details, refer to the Validating Application Package chapter (on page 17) further in this guide.

CHAPTER 3

Before Creating Package

Before you start creating an application package, consider the following:

- whether technologies used by your application (scripting languages, databases, server modules and the like) are qualified;
- whether your application requires additional configuration.

In This Chapter

Defining Application Technologies	8
Creating Application Configuration Scripts.....	9

Defining Application Technologies

Qualified technology is a software or executable code described in an aspect included in the Specification. For details on aspects that describe qualified technologies, refer to the 7. Common aspects section of the Specification.

If the application uses non-qualified technologies, write an aspect for each such technology and make it public. You can send them to the Specification maintainers for extending the Specification with your aspects (recommended scenario), or include them into the package release notes. This will help control panel developers to configure Controllers to provision the package properly. For details on what data should be included in an aspect, refer to the 6. Points of Extensibility section of the Specification.

Creating Application Configuration Scripts

Some applications may require additional configuration on its installation, reconfiguration, upgrade or removal. This can be performed with a help of scripts. Scripts can be written by application package developers to setup application instances. Script-writing language must be a qualified technology regulated by the Specification and must be declared in the metadata file.

Controller passes the data to scripts by means of environment variables. The variables contain all information about resources and settings of application instance. For details on what variables can be used by the scripts, refer to the 6.4. Environment Variables section of the Specification. For better understanding of what environment variables are passed to a particular application package configuration script, refer to the example provided in the Appendix B (on page 29).

Configuration script is invoked by Controller to configure application instances on installation, removal or upgrade. It is also called to reconfigure an application instance. The script is always invoked by Controller with one or more arguments that specify what task is performed by the script. For details on the script invocation and its arguments, refer to the 5.3.2.1. Configuration script actions section of the Specification.

For more information on configuration script, refer to the 5.3.2. Configuration script section of the Specification.

The following example presents a template of the `configure` script (written in PHP). The sample also shows how to operate with database environment variables passed to the script. When invoked with the `install` argument, the script connects to the database specified by environment variables and executes SQL queries stored in the `schema.sql` file.

```
<?php
if(count($_SERVER['argv']) < 2)
{
    print "Usage: configure (install | upgrade <version> | configure |
remove)\n";
    exit(1);
}

$command = $_SERVER['argv'][1];
//$command stores the argument with which the script was invoked.
if($command == "install")
{
    $db_id = 'main';
    //The database identifier value is to be substituted by the value //defined
in the application requirements section of the //metadata file.For details,
see the 6.3.1.1. Database requirement //type section of the Specification.
$query_file = 'schema.sql'; //File containing list of SQL queries.
//List of database-related variables that are passed to the configuration
//script. See the 6.3.1.1.1. Environment variables section of the
//Specification for details.
$db_address = getenv("DB_${db_id}_HOST");
if (fetch_env_var("DB_${db_id}_PORT") !== False)
    $db_address .= ':' . fetch_env_var("DB_${db_id}_PORT");
```

```
$dblogin = getenv("DB_${db_id}_LOGIN");
$dbpassword = getenv("DB_${db_id}_PASSWORD");
$dbname = getenv("DB_${db_id}_NAME");
//PHP functions for connecting to the mysql server and //executing SQL
queries.
mysql_connect($dbaddress, $dblogin, $dbpassword); mysql_select_db($dbname);
$sql_queries = file($query_file); foreach ($sql_queries as $query)
mysql_query($query);
//Other code to be executed on invoking configure with //the install
argument. exit(0);
}

if($command == "remove")
{
//Code to be executed on invoking configure with the remove argument
exit(0);
}

if($command == "upgrade")
{
//Code to be executed on invoking configure with the upgrade argument
exit(0);
}

if($command == "configure")
{
//Code to be executed on invoking configure with the configure argument
exit(0);
}

print "Error: unknown command $command.\n";
exit(1);

?>
```

CHAPTER 4

Creating Application Package

This chapter provides instructions on forming package physical structure and application metadata file.

In This Chapter

Creating Application Package Physical Structure	12
Creating Metadata File.....	13

Creating Application Package Physical Structure

Application package structure must conform to APS requirements. It must be comprised of the following files:

- *Application files.* The application files to be installed on a web site and additional files included in the package (application icon that is shown to control panel users, application screen shots, etc). The files must comply with requirements outlined in the 4.2. Files section in the Specification.
- *Metadata file.* The file that contains all information about the package and instructs Controller how to manage it. It must be called `APP-META.xml` and reside in the package root directory. For details on what data is stored in the file, refer to the Creating Application Package Metadata File section (on page 13) earlier in this guide.
- *Application Package scripts (optional).* The scripts which are invoked by a Controller to setup application instances. Application configuration scripts are placed in the `/scripts` directory in the package root. For details on what tasks can be performed by the scripts, refer to the Creating Application Configuration Scripts section (on page 9) earlier in this guide.

The following 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 to be setup
    ...
    ...
    ...

images/

    icon1.png        # Icon and screenshots of the
                    # application

screenshot2.jpg

    screenshot.jpg

    ...

htdocs/

    index.php        # Application files

    logo.php

    ...
```

Creating Metadata File

The application package metadata file (called `APP-META.xml`) contains all the metadata required to manage the application it describes. Metadata file includes application common properties like name, version, description as well as list of application services and resources required.

The XML presentation of the metadata is described by the RELAX NG schema (available at <http://apsstandard.com/r/doc/package-format-specification-1.0/basic.rnc>). Some schema elements are defined by aspects. It means that you must use aspects when adding the elements. For details on metadata elements, refer to the 5. Metadata Descriptor section of the Specification.

To create a metadata file, follow these steps:

- 1 Create file `APP-META.xml` in the package root directory. First of all, you should declare default XML namespace of metadata and version of APS specification the package complies with. For example,

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

- 2 Add elements that describe common application properties according to the 5.1. Common Application Properties section of the Specification. For details, refer to the Common Application Properties section (on page 13) further in this guide.

Important! `APP-META.xml` content is recommended to be in UTF-8 encoding.

- 3 Add services that your application provides. For each service you can specify its own settings, license agreement, required resources and provisioning methods. For details on the XML presentation of an application service, refer to the 5.2. Services and 5.3 Service Provision Methods sections of the Specification. For details, refer to the Application Services section (on page 14) further in this guide.

The example of metadata file is presented in Appendix A (on page 23). As an example we chose the *Open-Xchange* application package. It is an open-source application and has complex metadata file that allows demonstrating APS 1.1 new features, such as services and settings nesting.

Common Application Properties

Common application properties include general information about application origin, GUI-related properties and settings relevant for all application instances. Pre-defined elements are used to present application properties. These elements can be optional or required. Some elements represent only one property, for example, `version`, `name`, `release` elements. Others group related properties, for example `presentation` element. Group element can be optional, but contain required sub-element.

The following elements are most frequently used:

- name
- version
- release
- homepage (optional)
- vendor (optional) # *groups vendor properties*
- packager (optional)# *groups packager properties*
- presentation (optional)# *groups application GUI-related properties*
 - summary
 - description
 - icon (optional)
 - screenshot
 - changelog (optional)
 - categories (optional)
 - languages (optional)
 - patch (optional)
- global-settings (optional)
 - setting
 - patch (optional)
 - upgrade (optional)

For detailed list and description, refer to the 5.1. Common Application Properties section of the Specification.

Application Services

In general, an application purpose is to provide services to end-users. Application metadata file must contain, at least, one `service` element. Required resources, specific provisioning methods and settings are specified per application service. Also, application service may define child services.

The following is a set of elements used to describe application service:

- service
 - license (optional)
 - presentation (optional)
 - name (optional)
 - summary (optional)
 - infolinks (optional)
 - entry points (optional)
 - settings (optional)

- `requirements` (optional)
- `provision` (optional)
 - `url-mapping` (optional)
 - `configuration-script` (optional)
- `service` # *child service*
 - `requirements` (optional)
 - `provision` (optional)

Elements that represent service general properties are much the same as those of application. Service specific elements like `requirements` and `provision` we consider in details.

Service Requirements

The `requirements` element groups conditions to be satisfied to provision a service. It may include resources to be allocated, scripting languages, specific server modules. Requirements descriptions should be added according to technology-specific aspects. The aspects describing qualified technologies are found in the 7.0 Common Aspects section of the Specification.

For example, your application uses the PHP scripting language, and PHP version must be 5.0 or later. First, you should refer to the PHP aspect (see the 7.1. PHP aspect section of the Specification), read how the version requirement should be described in the metadata file (see the 7.1.1. Requirement types of the Specification), and add corresponding child element to the `requirements` element: `<php:version min="5.0" />`. The metadata fragment describing the requirement looks as follows:

```
<requirements xmlns:php="http://apstandard.com/ns/1/php">
  <php:version min="5.0" />
</requirements>
```

Additionally, the code of your application is encrypted, for example, with ionCube, you need to declare the necessity of using ionCube Loader. To do so, add the following child element to the `requirements` element: `<php:extension>ionCube Loader</php:extension>`. The metadata fragment describing the requirements will look in this case as follows:

```
<requirements xmlns:php="http://apstandard.com/ns/1/php">
  <php:version min="5.0" />
  <php:extension>ionCube Loader</php:extension>
</requirements>
```

Service Provisioning Methods

Provision methods are declared with `provision` element. Two types of provisioning methods are available: URL mapping and configuration script. Provisioning method choice depends on environment and application nature. Application may support both provisioning methods, depending on requirements satisfied. For example, you define to sets of requirements that represents different environments. Then, you specify what provisioning method should be used for each set. If Controller satisfies one of requirements sets, it uses assigned provisioning method. For details on service provisioning, refer to the 5.2.7. Service Provision section of the Specification.

URL Mapping

URL Mapping defines how to handle a request for a specific URL pertaining to the application. For details, refer to the 5.3.1 URL Mapping section of the Specification.

Some technologies require permission mappings for files or directories to be set. For example, your application service uses a PHP script located in the `htdocs/core` directory. The script is expected to write some data to a file located in the `htdocs/tmp` directory. Then the `provision` element should contain the following info:

- The `htdocs/core` directory must be handled by the PHP interpreter.
- PHP scripts must have write access to the `htdocs/tmp` directory.

PHP is a qualified technology. Instructions on how to describe PHP handlers and permissions are found in the 7.1.2. URL Handlers section of the Specification. Basing on these instructions and the XML presentation of the `mapping` element, the following fragment is to be added:

```
<mapping url="/core" path="htdocs/core"
xmlns:php="http://apstandard.com/ns/1/php">
  <php:handler>
    <php:extension>php</php:extension>
  </php:handler>
</mapping>
<mapping url="/tmp" path="htdocs/tmp"
xmlns:php="http://apstandard.com/ns/1/php">
  <php:permissions writable="true"/>
</mapping>
```

Note: Structure defined by URL mapping can differ from physical structure of an application instance.

For example, the package with the mentioned above mapping declarations is instantiated to the <http://www.example.com>. The requests to <http://www.example.com/core> are handled by PHP scripts in the `htdocs/core` directory of the site. If the first mapping element had `url="/php"` argument, then the requests to <http://www.example.com/php> were handled by PHP scripts in the `htdocs/core` directory.

Configuration Script

Configuration script can be used for service provisioning as well as for configuration, upgrade or removal. In case of provisioning, it is invoked with the `install` argument. It is allowed using different configuration scripts for different services provisioning. For details, refer to the 5.3.2 Configuration Script section of the Specification.

A programming language must be declared for each configuration script used. For example, your script `configure-box` is written in PHP, it is qualified technology. Instructions on what identifier should be used are found in the 7.3.1 Configuration Script Language of the Specification. Basing on these instructions and the XML presentation of the `configuration-script` element, the following fragment is to be added:

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

CHAPTER 5

Validating Application Package

This chapter provides instructions on validating application package. The package validation may comprise two steps:

- Checking package physical structure using APSLint utility.
- Checking whether package is properly managed by Controllers.

In This Chapter

APSLint Utility	18
Controller Operations on Packages.....	20

APSLint Utility

After you created an application package, you can determine whether the package conforms to the APS using a command-line utility called `APSLint`. The utility also enables you to check whether the application package can attain a certain APS certification level. For information on certification levels and requirements a package must satisfy to attain a specific level, refer to the *APS: Application Certification Criteria* document located on the official APS site.

`APSLint` utility validates the following:

- Application package file format;
- Application package physical structure;
- Application package metadata file.

As a result, the utility outputs the following data:

- Errors indicating why a package fails to conform to the APS;
- Errors indicating requirements that a package must satisfy to attain a certain certification level;
- Total number of errors. If 0 errors occur, the package is successfully validated.

The `APSLint` utility can be downloaded from the APS Standard web site (Windows and Linux/Unix versions are available).

Managing APSLint for Windows

To install the `APSLint` utility:

- 1 Download the utility using the following URL:
<http://apsstandard.com/doc/apslint.zip>.
- 2 Install Microsoft .NET Framework version 2.0/3.0/3.5. In case you already have it, just skip this step.
- 3 Extract the utility to your preferred location. We recommend it to be a folder where your application package file is located.

To validate an application package:

- 1 Run `APSLint` from the directory where the application package is located using the following command:

```
> apslint.exe <application-package-name>.app.zip
```

Note: if you run `APSLint` from another directory, you must provide the absolute or relative path to the application package that you want to validate.

Managing APSLint for Linux/Unix

To install the `APSLint` utility:

- 1 Download the utility using the following URL:
<http://apsstandard.com/doc/apslint.tar.gz>.
- 2 Install Mono version 1.2.x (2.0 runtime) and libgdiplus. In case you already have them, just skip this step.
- 3 Extract the utility to your preferred location.

To validate an application package:

- 1 Run `APSlint` from the directory where the application package is located using the following command:

- If the `binfmt` module is installed in your system:

```
# ./apslint.exe <application-package-name>.app.zip
```

- If the `binfmt` module is not installed in your system:

```
# mono ./apslint.exe <application-package-name>.app.zip
```

Note: if you run `APSlint` from another directory, you must provide the absolute or relative path to the application package that you want to validate.

Controller Operations on Packages

This section explains how Controller performs management operations that require application package data. Examine your package, considering Controller operations execution. These operations are as follows:

- Creating application instance from application package;
- Updating application instances using application package;
- Removing application instance created from application package.

Creating Application Instance

Controller performs the following actions when it is instantiating an application package:

- 1 Prepares all the data and conditions necessary for instantiating:
 - Presents an application-usage license text (if any is provided in the package) to a user and provides tools necessary for accepting the license if it is required. For details, refer to the 5.2.2 License Agreement section in the Specification.
 - Determines and satisfies the application requirements. For details, refer to the 5.2.6. Requirements, 5.3.2.2.3. Aspect-defined Environment Variables sections in the Specification.
 - Prompts the user to configure the application installation settings and processes the submitted settings. For details, refer to the 5.3.2.2. Environment Variables, 5.2.4. Entry points, 5.2.5. Service Settings sections in the Specification.
 - Retrieves the application's global configuration and prepares to apply it to the instance. For details, refer to 5.2.5. Service Settings, 5.3.2.2.2. Settings sections in the Specification.

Note: The order of these preparatory sub-stages depends on the Controller implementation.

- 2 Deploys the application files to a site. For details, refer to the 5.3.1. URL mapping, 5.3.2.2.3. Aspect-defined Environment Variables sections in the Specification.
- 3 Runs the application configuration script passing to it all the available information about the application in the form of environment variables created on the previous stages. For details, refer to the 6.6. Configuration Script Language, 5.3.2. Configuration script, 6.4. Environment Variables sections in the Specification.

Updating Application Instance

Controller can update an application instance in case a suitable update package is available. Updating application instance supposes that the application of older version is replaced with the one of newer version, and the user settings and files of the old application are picked up by the new one.

APS supports two types of application updates: *patch* and *upgrade*. Generally speaking, the difference is that patching supposes unattended instance update, while upgrading brings more serious changes to the instance and may require the user attendance. Application patch and upgrade are defined in detail by the Specification in section 5.1.13. Updates. Depending on what kind of update is required - patch or upgrade - a Controller behavior slightly differs:

- When patching, the Controller skips reading new requirements, and preserves all resources allocated to the old version so that the new version could painlessly pick them all up, as defined by the Specification in section 5.2.6. Requirements.
- When upgrading, the Controller analyzes new requirements and provides a mechanism of satisfying them together with migrating the application vitally-important data. (For example, when a new application version requires using a database of another version, the Controller provides mechanism of allocating the new database and migrating to it the old data.)

Removing Application Instance

When removing an application instance, a Controller invokes the application configuration script with `remove` argument. It removes the application files and deallocates resources used by it, as defined by the Specification in section 5.3.2.1. Configuration script actions. The Controller passes to the configuration script all application settings declared in metadata, except marked as `installation-only`, with the `SETTINGS_<id>` environment variables, as defined by the Specification in section 5.3.2.2.2. Settings.

Sample Metadata File

This appendix demonstrates the metadata file of the *Open-Xchange* application package. The *Open-Xchange* application is chosen because it is an open source application with a complex metadata file that allows us to demonstrate APS new features, such as services and settings nesting. Its application package can be downloaded at <http://www.apsstandard.com/app/>. The following metadata file is extended with delimiters that separate file fragments added on different steps of the metadata file creation described in the Creating Metadata File section (on page 13) earlier in this guide.

```
<!-- Application namespaces and APS version (step 1) -->
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://apstandard.com/ns/1" version="1.1">
<!-- Application common properties (step 2) -->
  <name>Open-Xchange</name>
  <version>6.7</version>
  <release>17</release>
  <homepage>http://www.open-xchange.com/en/products/open-xchange-
    hosting-edition-en</homepage>
  <vendor>
    <name>Open-Xchange Inc.</name>
    <homepage>http://www.open-xchange.com</homepage>
    <icon path="images/ox_logo.jpg" />
  </vendor>
  <packager>
    <name>Parallels</name>
    <homepage>http://parallels.com</homepage>
    <uri>uuid:c9fa49b2-ba47-11dd-9fed-00155882361a</uri>
  </packager>
  <presentation>
    <summary>
      Open-Xchange Hosting Edition is a highly advanced
      business-class email and collaboration solution
      for competitive success.
    </summary>
    <description>
      Open-Xchange Hosting Edition is a highly advanced
      business-class email and collaboration solution
      for competitive success. It provides a highly
      advanced, improved and intuitive browser based
      interface that meets all their email, calendaring,
      tasking, contacts and document sharing requirements.
      Each of the functions simplifies the daily work and
      it is the unique linking and integration of the
      different modules that makes organizing and managing
      within the company even smoother, faster, and more
      transparent.
    </description>
    <icon path="images/ox_logo.jpg" />
    <screenshot path="images/ox_portal.jpg">
      <description>Open-Xchange portal page</description>
    </screenshot>
```

```

<screenshot path="images/ox_email.jpg">
  <description>Open-Xchange E-Mail page</description>
</screenshot>
<screenshot path="images/ox_calendar.jpg">
  <description>Open-Xchange calendar page
  </description>
</screenshot>
<screenshot path="images/ox_contact.jpg">
  <description>Open-Xchange contact page</description>
</screenshot>
<screenshot path="images/ox_task.jpg">
  <description>Open-Xchange task page</description>
</screenshot>
<screenshot path="images/ox_infostore.jpg">
  <description>Open-Xchange infostore page
  </description>
</screenshot>
<changelog>
  <version version="6.7" release="1">
    <entry>Initial package version</entry>
  </version>
</changelog>
<categories>
  <category>Collaboration/Email</category>
</categories>
<languages>
  <language>en</language>
  <language>de</language>
</languages>
</presentation>
<global-settings>
  <setting id="ox_host" class="title" type="string"
  default-value="" min-length="1">
    <name>Open-Xchange installation host</name>
    <description>This is DNS name or IP address of
      Open-Xchange installation, used for
      provisioning access.
    </description>
  </setting>
  <setting id="ox_site" class="title" type="string"
  default-value="" min-length="1">
    <name>Open-Xchange public site address</name>
    <description>This is DNS name or IP address of
      Open-Xchange public site address.
    </description>
  </setting>
  <setting id="ox_master_admin" class="title" type="string">
    <name>Master Administrator Login</name>
    <error-message>Please make sure the text you entered
      starts with a letter and continues
      with either numbers, letters,
      underscores or hyphens.
    </error-message>
  </setting>
  <setting id="ox_master_password" class="title"
  type="password">
    <name>Master Administrator Password</name>
  </setting>
</global-settings>
<!-- Application services (step 3) -->
<service id="context" class="service">

```

```

<license must-accept="true">
  <text>
    <name>End User License Agreement</name>
    <url>http://software.open-xchange.com/
      OX6/doc/OXHE-Provisioning/ar02.html
    </url>
  </text>
</license>
<presentation>
  <entry-points>
    <entry dst="http://{ox_site}/mail/elogin.php"
      method="POST">
      <label>Open-Xchange context
        administration</label>
      <variable name="ox_site"
        value-of-setting="ox_site" />
      <variable name="username"
        value-of-setting="admin_login" />
      <variable name="password"
        value-of-setting="admin_password" />
    </entry>
  </entry-points>
</presentation>
<settings>
  <group class="authn">
    <setting id="admin_login" class="login"
      type="email" installation-only="true"
      default-value="admin">
      <name>Administrator login</name>
    </setting>
    <setting id="admin_password" class="password"
      type="password" track-old-value="true"
      min-length="1">
      <name>Administrator password</name>
    </setting>
  </group>
  <group class="vcard">
    <group class="email">
      <setting id="admin_email" class="value"
        type="email">
      <name>Administrator primary email
        address</name>
    </setting>
  </group>
  <group class="fn n">
    <setting id="admin_given_name"
      class="given-name" type="string"
      min-length="1">
      <name>Administrator given name
        </name>
    </setting>
    <setting id="admin_surname"
      class="family-name" type="string"
      min-length="1">
      <name>Administrator surname</name>
    </setting>
  </group>
  <setting id="organization_name"
    class="organization-name"
    default-value="" type="hidden">
    <name>Organization</name>

```

```

        </setting>
    </group>
    <setting id="filestore_quota" type="hidden"
        default-value="1024">
        <name>Open-Xchange context wide filestore
            quota (in MB)
        </name>
    </setting>
</settings>
<requirements xmlns:php="http://apstandard.com/ns/1/php">
    <php:version min="5.0" />
    <php:extension>soap</php:extension>
</requirements>
<provision>
    <configuration-script name="configure.php">
        <configuration-script-language>php
        </configuration-script-language>
        <status-control/>
    </configuration-script>
</provision>
<service id="account" class="account">
    <presentation>
        <entry-points>
            <entry dst="http://{ox_site}/mail/
                elogin.php" method="POST">
                <label>Open-Xchange account
                </label>
                <variable name="ox_site"
                    value-of-setting="ox_site" />
                <variable name="username"
                    value-of-setting="user_login" />
                <variable name="password"
                    value-of-setting="user_password"
                    />
            </entry>
        </entry-points>
    </presentation>
    <settings>
        <group class="authn">
            <setting id="user_login" class="login"
                type="string" installation-only="true"
                min-length="1">
                <name>Login</name>
            </setting>
            <setting id="user_password"
                class="password"
                type="password" min-length="1">
                <name>Password</name>
            </setting>
        </group>
        <group class="vcard">
            <group class="email">
                <setting id="user_email"
                    class="value"
                    type="email">
                    <name>Primary email address
                    </name>
                </setting>
            </group>
            <group class="fn n">
                <setting id="user_given_name"

```

```

        class="given-name" type="string"
        min-length="1">
            <name>Given name</name>
        </setting>
    </setting id="user_surname"
    class="family-name" type="string"
    min-length="1">
        <name>Surname</name>
    </setting>
</group>
</group>
<setting id="admin_login" type="hidden"
    value-of-setting="admin_login"/>
<setting id="admin_password" type="hidden"
    value-of-setting="admin_password"/>
</settings>
<requirements
xmlns:mail="http://apstandard.com/ns/1/mail">
    <mail:mailbox>
        <mail:id>account</mail:id>
        <mail:access>
            <mail:imap/>
        </mail:access>
        <mail:outgoing>
            <mail:smtp/>
        </mail:outgoing>
    </mail:mailbox>
</requirements>
<provision>
    <configuration-script name="configure-mbox.php">
        <configuration-script-language>php
        </configuration-script-language>
        <status-control/>
    </configuration-script>
</provision>
</service>
</service>
</application>

```

After Controller has finished to handle metadata file and services provisioning, it provides application configuration scripts with environment variables. For details what variables are in the list, refer to the Appendix B (on page 29).

Sample Environment Variables

For the purpose of illustrating what environment variables are passed to the application configuration script on its invocation, we chose the *Open-Xchange* application package. It is an open-source application and has complex metadata file that allows demonstrating APS new features, such as services and settings nesting. The application package can be downloaded at <http://www.apsstandard.com/app/>.

The types of environment variables that are passed to the *Open-Xchange* configuration scripts by a Controller are as follows:

- Variables are defined by application settings (see the 5.3.2.2.2. Settings section of the Specification).
- Aspects-defined (see the 6.4. Environment Variables section of the Specification).

The full list of *Open-Xchange* environment variables list is as follows:

- *Defined by application settings declarations:*

```

SETTINGS_ox_host
SETTINGS_ox_site
SETTINGS_ox_master_admin
SETTINGS_ox_master_password
SETTINGS_admin_login
SETTINGS_admin_password
SETTINGS_admin_email
SETTING_admin_given_name
SETTING_admin_surname
SETTING_organization_name
SETTINGS_filestore_quota
OLDSETTINGS_admin_password
SETTINGS_user_login
SETTINGS_user_password
SETTING_user_email
SETTING_user_given_name
SETTING_user_surname
    
```

- *Aspects-defined environment variables.*

- **PHP aspect**

```
PHP_VERSION
```

- **Mail aspect**

MAIL_account_EMAIL
MAIL_account_IMAP_HOST
MAIL_account_IMAP_MAILBOX
MAIL_account_IMAP_PORT
MAIL_account_IMAP_PORT_SSL
MAIL_account_PASSWORD
MAIL_account_POP3_HOST
MAIL_account_POP3_PORT
MAIL_account_POP3_PORT_SSL
MAIL_account_SMTP_HOST
MAIL_account_SMTP_PORT
MAIL_account_SMTP_PORT_SSL
MAIL_account_USER

Index

A

About Application Packaging Standard • 3
About This Guide • 4
Application Services • 16
APSLint Utility • 20

B

Before Creating Package • 8

C

Common Application Properties • 15
Controller Operations on Packages • 22
Creating Application Configuration Scripts • 9
Creating Application Instance • 23
Creating Application Package • 11
Creating Application Package Physical
Structure • 12
Creating Metadata File • 14

D

Defining Application Technologies • 8

F

Feedback • 6

I

Introduction • 3

M

Managing APSLint for Linux/Unix • 21
Managing APSLint for Windows • 21

P

Packaging Overview • 7

R

Removing Application Instance • 24

S

Sample Environment Variables • 31
Sample Metadata File • 25
Service Provisioning Methods • 18
Service Requirements • 17

T

Typographical Conventions • 5

U

Updating Application Instance • 23
Useful Links • 5

V

Validating Application Package • 20