
Parallels

Application Packaging Standard 1.2

Developer's Guide

Revision 1.0.14



(c) 1999-2009

Contents

Introduction	4
About Application Packaging Standard.....	4
About This Guide.....	5
Useful Links.....	6
Typographical Conventions.....	6
Feedback.....	7
Packaging Instruction	8
1. Define type of an environment where the application is to be provisioned.....	8
2. Define a model of service provisioning.	9
3. Define application structure.	9
4. Define application services hierarchy.	9
5. Define an application licensing procedure.	10
6. Define technologies used by your application.	10
7. Define presentation details and settings for application and its services.	11
8. Define application content delivery method.	11
9. Create a draft of metadata file.	12
10. Write configuration scripts.....	12
11. Prepare package contents listing.....	13
12. Create an application package structure.	13
13. Archive the application package files.....	14
14. Validate the application package.	14
Packaging Scenarios	15
Packaging WordPress Application.....	15
Packaging Sugar CRM Application.....	19
Packaging Open-Xchange Application.....	24
Validating Application Package	28
APSLint Utility.....	29
Managing APSLint for Windows.....	29
Managing APSLint for Linux/Unix.....	29
Controller Operations on Packages.....	31
Creating Application Instance.....	31
Updating Application Instance.....	32
Removing Application Instance.....	32
Appendix A. WordPress Sample Metadata File	33
Sample Environment Variables.....	36
Appendix B. Sugar CRM Sample Metadata File	37
Sample Environment Variables.....	44

Sample Metadata File	47
----------------------	----

Sample Environment Variables	52
------------------------------------	----

Index	54
-------	----

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.

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:

Packaging Instruction section provides step-by-step packaging instruction.

Packaging Scenarios section (on page 15) contains three examples of packaging real applications.

Validating Application Package section (on page 28) provides information on validation tool (APSLint utility) and possible operations performed over application package by Controller.

Appendixes (on page 47) with metadata files of application described in Packaging Scenarios section.

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.
<i>Italics</i>	Titles of chapters, sections, and subsections. 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.	Read the Basic Administration chapter. 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.

CHAPTER 2

Packaging Instruction

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 intended that you already have a working application.

1. Define type of an environment where the application is to be provisioned.

Application may dictate type of environment the Controller should provision it to in clear. On the other hand, the Controller may select an environment basing on application services hardware and software requirements. One may single out the following types of hosting environment:

- *Shared environment* - hosting environment that is shared by instances of an application (multi-instance application);
- *Dedicated environment* - application states that only one instance can be ran in a hosting environment (single-instance application);
- *External environment* - hosting environment that is not managed by Controller, and application provisioning, in this case, is delegated to configuration script.

2. Define a model of service provisioning.

Selecting the provisioning model application vendor should consider the level of application management by Controller and application services hierarchy. Namely, one should correctly map application services on application. One may single out the following models of service provisioning:

- *Multi-tenant* - this model implies that a single instance of the application runs on hosting vendor servers, serving multiple client organizations (tenants). Tenants data and application configuration are stored separately. When a tenant logs in, an application instance respective configuration (customization) is applied and each tenant see and manages only his data as if he is an application single user. So, in this model, application context is already exists in hosting environment and customers' data is stored within it separately.
- *Single-tenant* - this model implies that an application instance run on hosting vendor servers, serving only one tenant. In other words, a separate application instance is provisioned for each customer. Application files are simply copied to location dedicated to customer and data of all customer applications is stored within customer.

Note: both models can be implemented for shared environment and for dedicated one as well.

3. Define application structure.

An application can be divided on main module and additional ones. The main module is responsible for application actual functioning and providing basic services. Additional modules usually provide some extra functionality. For example changing interface appearance with a help of skins is a basic service, but a number of additional skins are add-ons.

If your application have such additional modules, consider packaging them individually like a general package with dependency on their main module. In other words, metadata of packaged add-ons must reference to their master package, so the Controller could check whether the package is really an additional module to an application installed in hosting environment and its version meets condition (if any) in add-on's metadata file.

For details on packaging add-ons, refer to the 5.1.5. Master package reference section of the Specification.

4. Define application services hierarchy.

In general, an application purpose is to provide services to end-users. Application services hierarchy should reflect order of application services provisioning and level of application services management by Controller.

For example, let's consider multi-tenant application. The top-level application service represents a tenant. Under top-level service an application context is declared (second-level service) that is provisioned inside of the tenant. Then one may declare a number of application features (third-level services). Another way is to declare users and specify features for a user.

5. Define an application licensing procedure.

APS supports licensing applications by means of license keys. Each application instance or service may require license keys. To get keys, a well-formed license request should be send to respective authority. After the license is issued it is installed to the application.

If your application requires license to be installed, you should specify this in its metadata file. A special script should perform all involved operations, like querying application, installing and removing license.

For details on license requirement declaration, refer to the APS Licensing Aspect document.

6. Define technologies used by your application.

Used technologies may be scripting languages, specific server modules, databases, DNS zone specific records and the like. Application technologies form the list of requirements are to be satisfied to provision application. In general, requirements usually request some resource to be allocated or specific configuration to be performed.

Keep in mind that Controller can recognize only qualified 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.

Tip: write down an application requirements and used technologies in terms of the Specification. Then it will be easier to form metadata file.

7. Define presentation details and settings for application and its services.

Presentation details and settings makes application interface user-friendly. If an application is multi-tenant, for example, it should meet requirements of a number of tenants that uses this application for absolutely different purposes. Namely, the more settings you declare for an application, the more flexible it becomes. The Specification offers you a great variety of ways to provide end-users with user-friendly and configurable interface. For details, refer to the 5.1 Common Application Properties and 5.2 Services sections of the Specification.

Tip: write down an application settings and presentation details in terms of the Specification. Then it will be easier to form metadata file.

8. Define application content delivery method.

Content delivery method defines the way application files are deployed to hosting environment. For now APS allows using the following methods:

- *Inside package* - the method implies that package content does not require any special processing prior its deployment. The Controller simply copying the very application files and directories from the package to a hosting environment specific location, according to URL mapping rules (if any specified).
- *Inside PVC template* - the method implies that application files are packed in PVC template that regulates application files deployment. For details on PVC templates, refer to the Parallels Virtuozzo Containers: Templates Management guide <http://download.swsoft.com/virtuozzo/virtuozzo4.0/docs/en/lin/VzLinuxTmplMgmt/index.htm>.
- *Delivery is not required* - the method implies that hosting environment already contains all necessary content and do not require any configuration. This method suites for provisioning external service.

9. Create a draft of metadata file.

Each package must contain a well-formed XML file (called `APP-META.xml`) that includes all the metadata required to manage the application. 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 <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 file and its elements, refer to the 4.3. Metadata File and 5. Metadata Descriptor sections of the Specification.

10. Write configuration scripts.

The Specification defines the following provisioning methods: URL mapping and configuration script. These methods are not mutually exclusive and do not depend on each other. Thus, both of them can be declared for a service provisioning. URL mapping configures access to service through web and configuration script performs service installation. Method of service provisioning depends on hosting environment or application nature. Application may declare different sets of provisioning methods depending on which requirements (set of requirements) were satisfied. For details on provisioning methods, refer to the 5.3. Service Provision Methods section of the Specification.

You may develop a number of scripts that serve different purposes:

- *Configuration script.* It serves as method of service provisioning. Also, it is invoked by Controller on application instances configuration, removal, upgrade and enabling or disabling service. If an application has more than one service, separate configuration scripts may be implemented for each service. All resources required by a service must be allocated, instance files unpacked and placed to the file system before configuration script invocation. For details on configuration script invocation, its arguments and environment variables, refer to the 5.3.2. Configuration Script section of the Specification.
- *Verification script.* It serves to verify service settings for consistency, but it should never change state of application or its instances. Verification script has the same calling conventions as a configuration script, but does not support `enable` and `disable` arguments. The script can be declared and called in global context, as well, to check application global settings. For details on verification script invocation, its arguments and environment variables, refer to the 5.3.3. Verification Script section of the Specification.
- *Resource script.* It serves to report application current resources usage to a Controller. The script utilizes the same calling conventions as configuration one, but its output stream should match XML scheme defined on the Specification. For details on resource script invocation, its arguments, output and environment variables, refer to the 5.3.4. Resource Script section of the Specification.

- *Backup script.* It serves to preserve and restore users' data. It is invoked by Controller on application instance backing up and restoration. Also, an operation of migration is performed with this script as sequential execution of backup and restore operations. For details on backup script invocation, its arguments and environment variables, refer to the 5.3.5. Backup/Restore Script section of the Specification.

The script should be able to back up all sensitive application data to file specified and all involved resources, if required; restore backed up data to location specified.

- *License script.* It serves to install application license, if any. It is invoked when new license is issued for the application, old license is updated or removed. The script should be able to get instance-specific license information, install issued license and remove installed one. For details on license script invocation, its arguments and environment variables, refer to the APS Licensing Aspect document.

Script-writing language must be a qualified technology regulated by the Specification and must be declared in the metadata file. Application data is passed to scripts by means of environment variables. The variables contain all information about resources and settings of application instance you defined on the 7th step of the instruction.

Scripts execution is not always going without a hitch. Consider making script output structure matching XML schema defined in the Specification. A Controller catches such structured output, reads instructions and quickly reacts, in case some errors occur. These instructions may include identifiers of settings with erroneous values accompanied by localized error messages or problems detailed descriptions as well as warnings and errors related to the requirements. For details on forming structure of script output, refer to the 5.3.2.3 Configuration script output section of the Specification.

After necessary scripts are implemented, update metadata file with scripts declarations, according to the Specification. For details, refer to the 5.3. Service Provision Methods and 8. Common Aspects sections of the Specification.

11. Prepare package contents listing.

Each package must contain a well-formed XML file named APP-LIST.xml in the package root directory. The file must contain the list of all files in the package, but itself. Size in bytes and SHA256 digest value must be specified for each file in the list. This file may be digitally signed by a packager or certification authority.

Presence of packager signature assures package integrity and helps to prevent counterfeit. If a package is signed by certification authority, for example APS organization, the package contains a certificate as well.

For details on listing structure, refer to the 4.4 Package contents listing and 6. Package contents listing sections of the Specification

12. 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. Make sure, that *APP-META.xml* and *APP-LIST.xml* files are well-formed and contain all necessary information about an application.

For details on the requirements, refer to the 1. Introduction, 4. Basic Package Format and 6.5. Additional scripts sections of the Specification.

13. 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.

14. 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 28) further in this guide.

CHAPTER 3

Packaging Scenarios

This chapter provides you with packaging scenarios of real applications. Each scenario is step-by-step explanation of creating metadata file. The whole metadata files are provided in respective Appendixes.

Packaging WordPress Application

WordPress is a personal publishing platform designed to ease web blog management. The application is built on PHP and MySQL and licensed under the GPL. It is designed to be installed on dedicated web server or shared hosting account and desktop. Its size is about 6,4 MB. The latest stable release of WordPress is version 2.7.1. Web blog can be localized and delivered in a language at customer choice. Every installation of WordPress comes with a file editor one can use to edit templates and other WordPress related files, right in a browser without having to worry about downloading and uploading the files in order to edit them.

Note: above information is taken from application vendor home page.

Packaging procedure:

- 1 Type of environment. The application vendor states that WordPress can be installed on shared hosting. So, we can draw a conclusion that WordPress is a multi-instance application.
- 2 Model of service provisioning. The application is managed by one user 'admin' with all privileges. It is not possible to create more than one user 'admin' within one application instance. A number of users with restricted set of privileges can be added under the 'admin'. All data is shared between users and web blog appearance depends only on 'admin' user preferences. So, WordPress is a single-tenant or simple application and application files are simply copied to location dedicated to customer.
- 3 Services hierarchy. WordPress provides only one service - web blog management. Let's write down it in the Specification terms:
- 4 Used technologies. WordPress requires PHP version 4.3. or greater and MySQL version 4.0 or greater. Let's write down it in the Specification terms:

```
<requirements xmlns:php="http://apstandard.com/ns/1/php"
  xmlns:db="http://apstandard.com/ns/1/db">
  <php:version min="4.2.0"/>
  <php:extension>mysql</php:extension>
  <php:safe-mode>>false</php:safe-mode>
  <db:db>
    <db:id>main</db:id>
    <db:default-name>wordpress</db:default-name>
```

```

    <db:can-use-tables-prefix>>false</db:can-use-tables-prefix>
    <db:server-type>mysql</db:server-type>
    <db:server-min-version>4.0.0</db:server-min-version>
  </db:db>
</requirements>

```

- 5** Presentation details and settings. We defined earlier that WordPress provides one service - web blog management. Application requires the following settings on installation: 'admin' user login, password and e-mail address, web blog title and language choice. Let's group these settings for better presentation, and write down it in the Specification terms:

```

<settings>
  <group>
    <name>Administrator's preferences</name>
    <setting id="admin_name" type="string"
      default-value="admin" min-length="1"
      max-length="32" regex="^[a-zA-Z][0-9a-zA-Z_\-]*">
      <name>Administrator's 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="admin_password" type="password" min-length="1" >
      <name>Password</name>
    </setting>
    <setting id="admin_email" type="email">
      <name>Administrator's email</name>
    </setting>
  </group>
  <group>
    <name>Weblog's preferences</name>
    <setting id="title" type="string" min-length="1">
      <name>Weblog title</name>
    </setting>
  </group>
  <group>
    <name>Other preferences</name>
    <setting id="locale" type="enum" default-value="en-US">
      <name>Interface language</name>
      <choice id="en-US" >
        <name>English</name>
      </choice>
      <choice id="de-DE" >
        <name>German</name>
      </choice>
    </setting>
  </group>
</settings>

```

- 6** Content delivery method. WordPress is a simple application and should be accessible via Internet. So, delivery method is *Inside package* and URL mapping rules should be declared. Let's write down URL mapping rules and specify required amount of disk space in the Specification terms:

```

<provision>
  <url-mapping>
    <default-prefix>wordpress</default-prefix>
    <installed-size>6696960</installed-size>
    <mapping url="/" path="htdocs"
      xmlns:php="http://apstandard.com/ns/1/php">
      <php:handler>

```

```
<php:extension>php</php:extension>
</php:handler>
<mapping url="blogs/media">
  <php:permissions writable="true"/>
</mapping>
<mapping url="wp-content">
  <php:permissions writable="true"/>
</mapping>
<mapping url="tmp">
  <php:permissions writable="true"/>
</mapping>
</url-mapping>
</provision>
```

Draft of metadata file. Now, it is time to make a draft of the application metadata file. Describe application general information in the Specification terms. For details, refer to the 5.1 Common Application Properties section of the Specification. Add information written down on previous steps and replenish it with details.

Configuration script. Write configuration script that performs application installation, configuration and removal, using environment variable defined on the previous steps. Configuration script should perform all actions required for application configuration. For the list of environment variables, refer to the Sample Environment Variables section (on page 36) further in this guide.

Add information about script to the `provision` section of the metadata file. For example,

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

Archiving application. Form application package according URL mapping rules defined and requirements in the 4. Basic Package Format section of the Specification and archive it. In our case, the structure is the following:

```
APP-META.xml      # Metadata container. XML file.

scripts/

    configure     # This script will be invoked when
                  # application instance is to be setup.

images/

    icon.png      # Icon and screenshots of the
                  # application
    admin_page.jpg

htdocs/

    blogs/        # Application files
    ...
    index.php
    ...
```

Packaging Sugar CRM Application

Sugar CRM Community Edition enables organizations to efficiently organize, populate, and maintain information on all aspects of their customer relationships. It is a bit more complex application than WordPress one. It consists of modules, each of which represents a specific functional aspect of CRM such as Accounts, Activities, Leads and Opportunities. These modules are designed to help managing customer accounts through each step of their lifecycle, starting with generating and qualifying leads to customer support and resolving reported bugs. The application is managed by administrator, who has power to regulate access for the application modules, customize the look and feel of the application across an organization. The application is accessed by users through Internet, so it requires a network access to be configured to a server that is running the Sugar software. SugarCRM uses GNU General Public License version 3 for the Sugar Community Edition. It requires the following components to be installed on server: PHP with memory limit 40Mb or higher, Apache and MySQL.

Note: above information is taken from application vendor home page.

Packaging procedure:

- 1 Type of environment. The application vendor states that Sugar CRM can be installed on shared hosting. So, we can draw a conclusion that the application is a multi-instance one.
- 2 Model of service provisioning. The application is managed by administrator (top-level user) with access to each application module. It is possible to create more than one such user within one application instance. Administrator enables users and regulate their access rights. Still, all data is shared between users and stored within the application instance. So, Sugar Pro is a single-tenant or simple application and application files are simply copied to location dedicated to customer.
- 3 Services hierarchy. Sugar CRM allows creating more than one top-level user within the application instance. Let's write down it in the Specification terms:

```
<service id="instance">
  <service id="account">
  </service>
</service>
```

- 4 Used technologies. Sugar CRM requires PHP 5.1. or higher with memory limit 40Mb or higher, and MySQL. Let's write down it in the Specification terms:

```
<requirements xmlns:php="http://apstandard.com/ns/1/php"
  xmlns:db="http://apstandard.com/ns/1/db"
  xmlns:apache="http://apstandard.com/ns/1/apache">
  <php:version min="5.1.0"/>
  <php:memory-limit>41943040</php:memory-limit>
  <php:extension>mysql</php:extension>
  <php:extension>mbstring</php:extension>
  <db:db>
    <db:id>main</db:id>
    <db:default-name>sugarce</db:default-name>
    <db:can-use-tables-prefix>true
  </db:can-use-tables-prefix>
  <db:server-type>mysql</db:server-type>
```

```

        <db:server-min-version>4.1.2
        </db:server-min-version>
    </db:db>
</requirements>

```

- 5** Presentation details and settings. We defined earlier that SugarCRM provides two services - instance and account. Application requires the following settings on installation: 'admin' user login and password, system name to be displayed in the browser title bar, whether application sends usage statistics and how checks for upgrade are performed. The second-level service - account - requires user login and password and profile information. APS allows assigning the `class` attribute to setting or setting group. This attribute inform Controller about the setting meaning. The Controller may decide whether to prompt customer or use some pre-defined values, basing on attribute meaning. Let's group these settings for better presentation, and write down it in the Specification terms. Here, we demonstrate only instance settings, to view the whole set, refer to the Appendix B. Sugar CRM Sample Metadata section (on page 37).

```

<service id="instance">
  <settings>
    <group class="authn">
      <name>Administrator's Account</name>
      <setting id="admin_name" type="string"
        default-value="admin"
        min-length="1" max-length="32"
        regex="^[a-zA-Z][0-9a-zA-Z_\-]*"
        class="login">
        <name>Administrator's 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="admin_password" type="password"
        class="password">
        <name>Administrator's Password</name>
      </setting>
    </group>
    <group class="web">
      <setting id="title" type="string"
        default-value="SugarCRM" class="title">
        <name>System Name</name>
        <description>This name will be displayed
          in the browser title bar when users
          visit the Sugar application.
        </description>
      </setting>
      <setting id="send_usage_statistics"
        type="enum" default-value="true"
        installation-only="true">
        <name>Send Anonymous Usage Statistics</name>
        <description>If selected 'Yes', Sugar will
          send anonymous statistics about your
          installation to SugarCRM Inc. every
          your system checks for new versions.
        </description>
        <choice id="true">
          <name>Yes</name>
        </choice>
        <choice id="false">

```

```

        <name>No</name>
    </choice>
</setting>
<setting id="check_for_updates" type="enum"
    default-value="automatic">
    <name>Check For Updates</name>
    <description>How the system will check
        for updated versions of the
        application.</description>
    <choice id="automatic">
        <name>Automatic</name>
    </choice>
    <choice id="manual">
        <name>Manual</name>
    </choice>
</setting>
</group>
</settings>
</service>

```

- 6** Content delivery method. Sugar CRM is a simple application and should be accessible via Internet. So, delivery method is *Inside package* and URL mapping rules should be declared. URL mapping rules declaration is needed only for instance service. Let's write down URL mapping rules and specify required amount of disk space in the Specification terms:

```

<provision>
  <url-mapping>
    <default-prefix>sugarcrm</default-prefix>
    <installed-size>53547008</installed-size>
    <mapping url="/" path="htdocs"
      xmlns:php="http://apstandard.com/ns/1/php">
      <php:handler>
        <php:extension>php</php:extension>
      </php:handler>
      <mapping url="cache">
        <php:permissions writable="true"/>
      </mapping>
      <mapping url="custom">
        <php:permissions writable="true"/>
      </mapping>
      <mapping url="data">
        <php:permissions writable="true"/>
      </mapping>
      <mapping url="modules">
        <php:permissions writable="true"/>
      </mapping>
      <mapping url="tmp">
        <php:permissions writable="true"/>
      </mapping>
      <mapping url="config.php" virtual="virtual">
        <php:permissions writable="true"/>
      </mapping>
    </mapping>
  </url-mapping>
</provision>

```

- 7** Draft of metadata file. Now, it is time to make a draft of the application metadata file. Describe application general information in the Specification terms. For details, refer to the 5.1 Common Application Properties section of the Specification. Add information written down on previous steps and replenish it with details.

- 8 Configuration script.** Sugar CRM provides two services, consider possibility to write separate configuration scripts for each service as in our example. Configuration script should perform all actions required for application configuration using environment variables defined on previous steps. For the list of environment variables, refer to the Sample Environment Variables section (on page 44) further in this guide.

The application vendor states that any user could not be deleted, the only way to restrict access at all is to disable user. Considering this information, we added `<status-control/>` element to the "usermanager" script.

Add information about scripts to the respective `provision` sections of the metadata file. For example,

```
<service id="instance">
  <provision>
    <configuration-script name="configure">
      <configuration-script-language>php
    </configuration-script-language>
    </configuration-script>
  </provision>
  <service id="account">
    <provision>
      <configuration-script name="usermanager">
        <configuration-script-language>php
      </configuration-script-language>
      <status-control/>
    </configuration-script>
    </provision>
  </service>
</service>
```

- 9 Archiving application.** Form application package according URL mapping rules defined and requirements in the 4. Basic Package Format section of the Specification and archive it. In our case, the structure is the following:

```
APP-META.xml      # Metadata container. XML file.

scripts/

  configure      # This script will be invoked when
  usermanager    application instance is to be setup.

images/

  icon.png      # Icon and screenshots of the
  ...           application

screen_admin.png

htdocs/
```

```
cache/          # Application files
...
modules/
index.php
...
```

Packaging Open-Xchange Application

The Open-Xchange Hosting Edition is built in a completely modular way that allows service providers to offer a variety of Smart Collaboration solutions - from business e-mail over personal information management (PIM) to a comprehensive collaboration solution. The architecture of the Open-Xchange Hosting Edition enables seamless integration into existing infrastructures with existing tools for authentication, user setup, system administration, accounting and e-mail storage. The Open-Xchange Hosting Edition offers a complete client capability that makes it possible to operate many thousands of customers simultaneously in a virtual server environment. This guarantees customers consistently good efficiency and performance and utilizes hardware resources in the best possible way.

Note: above information is taken from application vendor home page.

Packaging procedure:

- 1 Type of environment. The application vendor states that Open-Xchange can seamlessly integrate to any environment. So, we can draw a conclusion that the application is a multi-instance one and may be provisioned to external environment that not managed by Controller.
- 2 Model of service provisioning. The application may serve to some number of customers simultaneously in a virtual server environment. Each customer can work in his own environment within this instance – completely separated from each other customer.
- 3 Services hierarchy. Open-Xchange supports multi-tenancy and this means that customers (tenants) are enabled within one application instance. Within each tenant (organization) a number of end-users (staff members) is created. In order to inform the Controller about service destination we assign `class` attribute to each service. Let's write down it in the Specification terms:

```
<service id="context" class="service">
  <service id="account" class="account">
  </service>
</service>
```

- 4 Used technologies. There is no need to provision application itself, so the only requirements we should declare are script language support and mail box for end-users. Let's write down it in the Specification terms:

```
<service id="context" class="service">
  <requirements xmlns:php="http://apstandard.com/ns/1/php">
    <php:version min="5.0" />
    <php:extension>soap</php:extension>
  </requirements>
  <service id="account" class="account">
    <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>
</service>
</service>

```

- 5** Presentation details and settings. We defined earlier that there are two services - tenant environment and end-user account. Tenant environment requires the following settings on installation: 'administrator login and password, and profile information. The second-level service - account - requires user login, password and profile information. When end-user logs in application it is necessary to provide tenant credentials along with user ones. So, we declare to hidden settings for a user, that inherits tenant credentials. APS allows assigning the `class` attribute to setting or setting group. This attribute inform Controller about the setting meaning. The Controller may decide whether to prompt customer or use some pre-defined values, basing on attribute meaning. Let's group these settings for better presentation, and write down it in the Specification terms. Here, we demonstrate only tenant settings, to view the whole set, refer to the Appendix C. Open-Xchange Sample Metadata section (on page 47).

```

<service id="context" class="service">
  <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>
    </group>
  </settings>
</service>

```

```

        </setting>
    </group>
    <setting id="filestore_quota" type="hidden"
        default-value="1024">
        <name>Open-Xchange context wide filestore
            quota (in MB)
        </name>
    </setting>
</settings>
</service>

```

- 6** Content delivery method. Open-Xchange integrates into existing infrastructures by itself. So, it is provisioned like external service. When Controller provisions external services, it need to know service installation host, DNS name or IP address and login credentials. These settings affects for all instances of services and should be set prior to any service provisioning. Such settings are application global settings. Let's write down it in the Specification terms:

```

<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>

```

- 7** Draft of metadata file. Now, it is time to make a draft of the application metadata file. Describe application general information in the Specification terms. For details, refer to the 5.1 Common Application Properties section of the Specification. Add information written down on previous steps and replenish it with details.
- 8** Configuration script. Consider possibility to write separate configuration scripts for each service as in our example. Configuration script should perform all actions required for application configuration using environment variables defined on previous steps. Script for second-level service should configure user mailbox additionally. For the list of environment variables, refer to the Sample Environment Variables section (on page 52) further in this guide.

The application vendor states that any tenant/user could not be deleted, the only way to restrict access at all is to disable tenant/user. Considering this information, we added `<status-control/>` element to both scripts.

Add information about scripts to the respective `provision` sections of the metadata file. For example,

```
<service id="context" class="service">
  <provision>
    <configuration-script name="configure.php">
      <configuration-script-language>php
    </configuration-script-language>
    <status-control/>
    </configuration-script>
  </provision>
  <service id="account" class="account">
    <provision>
      <configuration-script name="configure-mbox.php">
        <configuration-script-language>php
      </configuration-script-language>
      <status-control/>
      </configuration-script>
    </provision>
  </service>
</service>
```

- 9 Archiving application.** Form application package requirements in the 4. Basic Package Format section of the Specification and archive it. The application is provisioned as external service, so application files themselves are not required. In our case, the structure is the following:

```
APP-META.xml          # Metadata container. XML file.

scripts/

configure             # This scripts will be invoked when
configure-            service instance is to be setup.
mbox.php

elogin.php

images/

                        # Screenshots of the application
ox_calendar.jpg

...

ox_task.jpg
```

CHAPTER 4

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.

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://www.apsstandard.org/r/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

`APSLint` requires *mono 2.0* emulator and *libgdiplus* package to be installed on server before running `APSLint`.

To install the APSlint utility:

- 1 Download the utility using the following URL:
<http://www.apsstandard.org/r/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.

Appendix A. WordPress Sample Metadata File

This appendix demonstrates the metadata file of the *WordPress* application package. The following metadata file is extended with delimiters that separate file fragments added on different steps of the metadata file creation described in the Packaging WordPress Application section (on page 15) earlier in this guide.

```
<!-- Application namespaces and APS version (step 9) -->
<application xmlns="http://apstandard.com/ns/1" version="1.2">
<!-- Application common properties (step 9) -->
  <id>http://wordpress.org/</id>
  <name>WordPress</name>
  <version>2.7.1</version>
  <release>1</release>
  <homepage>http://wordpress.org/</homepage>
  <vendor>
    <name>WordPress.org</name>
    <homepage>http://wordpress.org/</homepage>
    <icon path="images/icon.png"/>
  </vendor>
  <packager>
    <name>Parallels</name>
    <homepage>http://parallels.com</homepage>
    <uri>uuid:714f0a7b-85d6-4eb8-b68e-40f9acbb3103</uri>
  </packager>
  <presentation>
    <summary>WordPress is a state-of-the-art semantic personal
      publishing platform with a focus on aesthetics, web
      standards, and usability.</summary>
    <description>
      WordPress is a state-of-the-art semantic personal
      publishing platform with a focus on aesthetics,
      web standards, and usability. What a mouthful.
      WordPress is both free and priceless at the same time.
      More simply, WordPress is what you use when you want
      to work with your blogging software, not fight it.
    </description>
    <icon path="images/icon.png"/>
    <screenshot path="images/admin_page.jpg">
      <description>
        Admin page.
      </description>
    </screenshot>
    <changelog>
      <version version="2.7.0" release="5">
        <entry>'PHP safe mode off' requirement is added
          </entry>
      </version>
    </changelog>
    <categories>
      <category>Web/Blog</category>
```

```

    <category>Personal/Blog</category>
</categories>
<languages>
  <language>en</language>
</languages>
</presentation>
<patch match="/application/version = '2.5.1'
      and /application/release = '4'"/>
<upgrade match="/application/version = '2.0'"/>
<!-- Application service (step 4) -->
<service id="blog">
<!-- Service presentation properties (step 7) -->
  <license must-accept="true">
    <text>
      <name>GPLv2</name>
      <file>htdocs/license.txt</file>
    </text>
  </license>
  <presentation>
    <name>Blog</name>
    <entry-points>
      <entry class="control-panel" dst="/wp-admin/"
        method="POST">
        Administrative Interface
      </entry>
      <entry class="frontpage" dst="/">
        Application entry point
      </entry>
    </entry-points>
  </presentation>
  <!-- Service settings (step 7) -->
  <settings>
    <group>
      <name>Administrator's preferences</name>
      <setting id="admin_name" type="string"
        default-value="admin" min-length="1"
        max-length="32" regex="^[a-zA-Z][0-9a-zA-Z_\-]*">
      <name>Administrator's 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="admin_password"
        type="password" min-length="1" >
      <name>Password</name>
      </setting>
      <setting id="admin_email" type="email">
      <name>Administrator's email</name>
      </setting>
    </group>
    <group>
      <name>Weblog's preferences</name>
      <setting id="title" type="string" min-length="1">
      <name>Weblog title</name>
      </setting>
    </group>
    <group>
      <name>Other preferences</name>
      <setting id="locale" type="enum" default-value="en-US">
      <name>Interface language</name>
      <choice id="en-US" >

```

```

        <name>English</name>
    </choice>
    <choice id="de-DE" >
        <name>German</name>
    </choice>
</setting>
</group>
</settings>
<!-- Service used technologies (step 6) -->
<requirements xmlns:php="http://apstandard.com/ns/1/php"
    xmlns:db="http://apstandard.com/ns/1/db">
    <php:version min="4.2.0"/>
    <php:extension>mysql</php:extension>
    <php:safe-mode>>false</php:safe-mode>

    <db:db>
        <db:id>main</db:id>
        <db:default-name>wordpress</db:default-name>
        <db:can-use-tables-prefix>>false
            </db:can-use-tables-prefix>
        <db:server-type>mysql</db:server-type>
        <db:server-min-version>4.0.0</db:server-min-version>
    </db:db>
</requirements>
<!-- Content delivery settings (step 8) -->
<provision>
    <url-mapping>
        <default-prefix>wordpress</default-prefix>
        <installed-size>6696960</installed-size>
        <mapping url="/" path="htdocs"
            xmlns:php="http://apstandard.com/ns/1/php">
            <php:handler>
                <php:extension>php</php:extension>
            </php:handler>
            <mapping url="blogs/media">
                <php:permissions writable="true"/>
            </mapping>
            <mapping url="wp-content">
                <php:permissions writable="true"/>
            </mapping>
            <mapping url="tmp">
                <php:permissions writable="true"/>
            </mapping>
        </url-mapping>
    <!-- Service configuration script declaration (step 10) -->
    <configuration-script name="configure">
        <script-language>php
            </script-language>
        </configuration-script>
    </provision>
</service>
</application>

```

In This Appendix

Sample Environment Variables..... 36

Sample Environment Variables

The types of environment variables that are passed to the *WordPress* configuration script 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 *WordPress* environment variables list is as follows:

- *Defined by URL mapping rules:*

BASE_URL_SCHEME

BASE_URL_HOST

BASE_URL_PORT

BASE_URL_PATH

WEB__DIR

WEB__blogs_media_DIR

WEB__wp-content_DIR

WEB__tmp_DIR

- *Defined by application settings declarations:*

SETTINGS_admin_name

SETTINGS_admin_password

SETTINGS_admin_email

SETTINGS_title

SETTINGS_locale

- *Aspects-defined environment variables.*

- **PHP aspect**

PHP_VERSION

- **Database aspect**

DB_main_TYPE

DB_main_NAME

DB_main_LOGIN

DB_main_PASSWORD

DB_main_HOST

DB_main_PORT

DB_main_VERSION

DB_main_PREFIX

APPENDIX B

Appendix B. Sugar CRM Sample Metadata File

This appendix demonstrates the metadata file of the *Sugar CRM* application package. The following metadata file is extended with delimiters that separate file fragments added on different steps of the metadata file creation described in the Packaging Sugar CRM Application section (on page 19) earlier in this guide.

```
<!-- Application namespaces and APS version (step 9) -->
<application xmlns="http://apstandard.com/ns/1" version="1.2">
<!-- Application common properties (step 9) -->
  <id>http://www.sugarcrm.com/crm/</id>
  <name>SugarCRM</name>
  <version>5.2.0a</version>
  <release>1</release>
  <homepage>http://www.sugarcrm.com/crm/</homepage>
  <vendor>
    <name>SugarCRM Inc.</name>
    <homepage>
      http://www.sugarcrm.com/crm/about/about-sugarcrm.html
    </homepage>
    <icon path="images/icon.png"/>
  </vendor>
  <packager>
    <name>Parallels</name>
    <homepage>http://parallels.com</homepage>
    <uri>uuid:714f0a7b-85d6-4eb8-b68e-40f9acbb3103</uri>
  </packager>
  <presentation>
    <summary>Sugar CRM Community Edition</summary>
    <description>
      Sugar CRM Community Edition enables organizations to
      efficiently organize, populate, and maintain
      information on all aspects of their customer
      relationships. It provides integrated management of
      corporate information on customer accounts and contacts,
      sales leads and opportunities, plus activities such as
      calls, meetings, and assigned tasks. The system
      seamlessly blends all of the functionality required to
      manage information on many aspects of your business
      into an intuitive and user-friendly graphical interface.
    </description>
    <icon path="images/icon.png"/>
    <screenshot path="images/screen_home.png">
      <description>Home Screen</description></screenshot>
    <screenshot path="images/screen_admin.png">
      <description>Admin Screen</description></screenshot>
    <screenshot path="images/screen_dashboard.png">
      <description>Dashboard View</description></screenshot>
    <screenshot path="images/screen_accounts.png">
      <description>Account Details</description></screenshot>
    <screenshot path="images/screen_campaigns.png">
```

```

<description>Marketing Campaigns</description></screenshot>
  <screenshot path="images/screen_bug_tracker.png">
<description>Bug Tracker</description></screenshot>
  <screenshot path="images/screen_calendar.png">
<description>Calendar</description></screenshot>
  <screenshot path="images/screen_documents.png">
<description>Documents</description></screenshot>
  <changelog>
    <version version="5.2.0a" release="1">
      <entry>Packaged as APS 1.1</entry>
    </version>
  </changelog>
  <categories>
    <category>
      Back office/Customer Relationship Management
    </category>
  </categories>
  <languages>
    <language>en</language>
  </languages>
</presentation>
<patch match="/application/version = '5.0.0'
  or /application/version = '5.1.0a'"/>
<!-- Application services (step 4) -->
  <service id="instance">
<!-- Service presentation properties (step 7) -->
    <license must-accept="true">
      <free/>
      <text>
        <name>GPLv3</name>
        <file>htdocs/LICENSE.txt</file>
      </text>
    </license>
  </presentation>
    <name>SugarCRM Instance</name>
    <summary>Basic services</summary>
    <entry-points>
      <entry class="control-panel" dst="/index.php"
        method="POST">
        <label>Application entry point</label>
        <variable name="module">Users</variable>
        <variable name="action">Authenticate
      </variable>
        <variable name="return_module">Users
      </variable>
        <variable name="return_action">Login
      </variable>
        <variable name="cant_login"/>
        <variable name="login_module"/>
        <variable name="login_action"/>
        <variable name="login_record"/>
        <variable name="user_name" class="login"
          value-of-setting="admin_name"/>
        <variable name="user_password"
          class="password"
          value-of-setting="admin_password"/>
        <variable name="login_theme">Sugar
      </variable>
        <variable name="login_language">en_us
      </variable>
        <variable name="Login">++Login++
      </variable>
    </entry-points>
  </service>
</!-- Application services (step 4) -->
</!-- Service presentation properties (step 7) -->

```

```

        </variable>
    </entry>
</entry-points>
</presentation>
<!-- Service settings (step 7) -->
<settings>
    <group class="authn">
        <name>Administrator's Account</name>
        <setting id="admin_name" type="string"
        default-value="admin" min-length="1"
        max-length="32"
        regex="^[a-zA-Z][0-9a-zA-Z_\-]*"
        class="login">
            <name>Administrator's 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="admin_password" type="password"
        class="password">
            <name>Administrator's Password</name>
        </setting>
    </group>
    <group class="web">
        <setting id="title" type="string"
        default-value="SugarCRM" class="title">
            <name>System Name</name>
            <description>This name will be displayed
            in the browser title bar when users visit
            the Sugar application.</description>
        </setting>
        <setting id="send_usage_statistics"
        type="enum" default-value="true"
        installation-only="true">
            <name>Send Anonymous Usage Statistics
            </name>
            <description>If selected 'Yes', Sugar
            will send anonymous statistics about
            your installation to SugarCRM Inc.
            every time your system checks for
            new versions.
            </description>
            <choice id="true">
                <name>Yes</name>
            </choice>
            <choice id="false">
                <name>No</name>
            </choice>
        </setting>
        <setting id="check_for_updates" type="enum"
        default-value="automatic">
            <name>Check For Updates</name>
            <description>How the system will check
            for updated versions of the application.
            </description>
            <choice id="automatic">
                <name>Automatic</name>
            </choice>
            <choice id="manual">

```

```

                <name>Manual</name>
            </choice>
        </setting>
    </group>
</settings>
<!-- Service used technologies (step 6) -->
<requirements xmlns:php="http://apstandard.com/ns/1/php"
    xmlns:db="http://apstandard.com/ns/1/db"
    xmlns:apache="http://apstandard.com/ns/1/apache" >
    <php:version min="5.1.0"/>
        <php:extension>mysql</php:extension>
        <php:extension>mbstring</php:extension>
        <db:db>
            <db:id>main</db:id>
            <db:default-name>sugarcrm</db:default-name>
            <db:can-use-tables-prefix>true
        </db:can-use-tables-prefix>
        <db:server-type>mysql</db:server-type>
        <db:server-min-version>4.1.2
    </db:server-min-version>
    </db:db>
</requirements>
<!-- Content delivery settings (step 8) -->
<provision>
    <url-mapping>
        <default-prefix>sugarcrm</default-prefix>
        <installed-size>53547008</installed-size>
        <mapping url="/" path="htdocs"
    xmlns:php="http://apstandard.com/ns/1/php" >
            <php:handler>
                <php:extension>php</php:extension>
            </php:handler>
            <mapping url="cache">
                <php:permissions writable="true"/>
            </mapping>
            <mapping url="custom">
                <php:permissions writable="true"/>
            </mapping>
            <mapping url="data">
                <php:permissions writable="true"/>
            </mapping>
            <mapping url="modules">
                <php:permissions writable="true"/>
            </mapping>
            <mapping url="tmp">
                <php:permissions writable="true"/>
            </mapping>
            <mapping url="config.php"
    virtual="virtual">
                <php:permissions writable="true"/>
            </mapping>
        </url-mapping>
    </provision>
<!-- Service configuration script declaration (step 10) -->
    <configuration-script name="configure">
        <script-language>php
    </script-language>
    </configuration-script>
</provision>
<!-- Application services (step 4) -->

```

```

<service id="account">
  <!-- Service presentation properties (step 7) -->
  <presentation>
    <name>SugarCRM Account</name>
    <entry-points>
      <entry class="control-panel"
        dst="/index.php"
        method="POST">
        <label>Account entry point</label>
        <variable name="module">
          Users
        </variable>
        <variable name="action">
          Authenticate
        </variable>
        <variable name="return_module">
          Users
        </variable>
        <variable name="return_action">
          Login
        </variable>
        <variable name="cant_login"/>
        <variable name="login_module"/>
        <variable name="login_action"/>
        <variable name="login_record"/>
        <variable name="user_name"
          class="login"
          value-of-setting="user_login"/>
        <variable name="user_password"
          class="password"
          value-of-setting="user_password"/>
        <variable name="login_theme">Sugar
        </variable>
        <variable name="login_language">
          en_us
        </variable>
        <variable name="Login">++Login++
        </variable>
      </entry>
    </entry-points>
  </presentation>
  <!-- Service settings (step 7) -->
  <settings>
    <group class="authn">
      <name>Account Preferences</name>
      <setting id="user_login" class="login"
        track-old-value="true"
        type="string" min-length="3"
        max-length="60"
        regex="^[a-zA-Z][0-9a-zA-Z_\-]*">
        <name>Account's Login</name>
      </setting>
      <setting id="user_password"
        class="password" type="password"
        min-length="4">
        <name>Account's Password</name>
      </setting>
    </group>
    <group class="vcard">
      <group class="fn n">
        <setting id="first_name"

```

```
        class="given-name" type="string"
        max-length="30">
            <name>First Name</name>
        </setting>
        <setting id="last_name"
        class="family-name" type="string"
        max-length="30">
            <name>Last Name</name>
        </setting>
    </group>
    <group class="email">
        <setting id="user_email"
        class="value"
        type="email">
            <name>Email</name>
        </setting>
    </group>
    <setting id="title" class="title"
    type="string">
        <name>Title</name>
    </setting>
    <setting id="department"
    class="organization-unit"
    type="string">
        <name>Department</name>
    </setting>
    <group class="tel">
        <name class="type">work</name>
        <setting id="phone_work"
        class="value"
        type="string">
            <name>Work Phone Number
            </name>
        </setting>
    </group>
    <group class="tel">
        <name class="type">cell</name>
        <setting id="phone_mobile"
        class="value"
        type="string">
            <name>Mobile Phone Number
            </name>
        </setting>
    </group>
    <group class="tel">
        <name class="type">fax</name>
        <setting id="phone_fax"
        class="value"
        type="string">
            <name>Fax Number</name>
        </setting>
    </group>
    <group class="tel">
        <name class="type">home</name>
        <setting id="phone_home"
        class="value"
        type="string">
            <name>Home Phone Number
            </name>
        </setting>
    </group>
```

```

        <setting id="address_street"
        class="street-address" type="string">
            <name>Street</name>
        </setting>
        <setting id="address_city"
        class="locality"
        type="string">
            <name>City</name>
        </setting>
        <setting id="address_state"
        class="region"
        type="string">
            <name>Region</name>
        </setting>
        <setting id="address_country"
        class="country-name" type="string">
            <name>Country</name>
        </setting>
        <setting id="address_postalcode"
        class="postal-code" type="string">
            <name>Postal Code</name>
        </setting>
        <setting id="description" class="note"
        type="string">
            <name>Description</name>
        </setting>
    </group>
</settings>
<!-- Service configuration script declaration (step 10) --
>
    <provision>
        <configuration-script name="usermanager">
            <script-language>php
            </script-language>
            <status-control/>
        </configuration-script>
    </provision>
</service>
</service>
</application>

```

In This Appendix

Sample Environment Variables..... 44

Sample Environment Variables

The types of environment variables that are passed to the *Sugar CRM* 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 *Sugar CRM* environment variables list is as follows:

'Root' service variables:

- Pre-defined variables representing the instance: URL:
BASE_URL_SCHEME
BASE_URL_HOST
BASE_URL_PORT
BASE_URL_PATH
- Variables defined by service settings declarations:
SETTINGS_admin_name
SETTINGS_admin_password
SETTINGS_title
SETTINGS_send_usage_statistics
SETTINGS_check_for_updates
- Variables defined by service requirements declarations:
PHP_VERSION
DB_main_NAME
DB_main_LOGIN
DB_main_PASSWORD
DB_main_HOST
DB_main_PORT
DB_main_VERSION
DB_main_PREFIX
- Variables defined by service URL mapping declarations:
WEB__DIR
WEB__cache_DIR
WEB__custom_DIR
WEB__data_DIR
WEB__modules_DIR
WEB__tmp_DIR
WEB__config.php_DIR

Child service variables:

- Pre-defined variables representing the instance URL:
BASE_URL_SCHEME
BASE_URL_HOST
BASE_URL_PORT
BASE_URL_PATH
- Variables defined by service settings declarations:
SETTINGS_user_login
OLDSETTINGS_user_login
SETTINGS_user_password
SETTINGS_first_name
SETTINGS_last_name
SETTINGS_user_email
SETTINGS_title
SETTINGS_department
SETTINGS_phone_work
SETTINGS_phone_mobile
SETTINGS_phone_fax
SETTINGS_phone_home
SETTINGS_address_street
SETTINGS_address_city
SETTINGS_address_state
SETTINGS_address_country
SETTINGS_address_postalcode
SETTINGS_description
- Variables defined by parent service requirements declarations:
PHP_VERSION
DB_main_NAME
DB_main_LOGIN
DB_main_PASSWORD
DB_main_HOST
DB_main_PORT
DB_main_VERSION
DB_main_PREFIX

Sample Metadata File

This appendix demonstrates the metadata file of the *Open-Xchange* application package. The following metadata file is extended with delimiters that separate file fragments added on different steps of the metadata file creation described in the Packaging Open-Xchange Application section (on page 24) earlier in this guide.

```
<!-- Application namespaces and APS version (step 9) -->
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://apstandard.com/ns/1" version="1.2">
<!-- Application common properties (step 9) -->
  <id>http://www.open-xchange.com/</id>
  <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>
<!-- Content delivery settings (step 8) -->
<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 4) -->
<service id="context" class="service">
    <!-- Service presentation properties (step 7) -->
    <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>
<!-- Service settings (step 7) -->
<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>

```



```

        </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>
<!-- Service used technologies (step 6) -->

<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>
<!-- Service configuration script declaration (step 10) --
>
    <provision>
        <configuration-script name="configure-mbox.php">
            <script-language>php
            </script-language>
            <status-control/>
        </configuration-script>
    </provision>
</service>
</service>
</application>

```

In This Appendix

Sample Environment Variables..... 52

Sample Environment Variables

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
SETTINGS_admin_given_name
SETTINGS_admin_surname
SETTINGS_organization_name
SETTINGS_filestore_quota
OLDSETTINGS_admin_password
SETTINGS_user_login
SETTINGS_user_password
SETTINGS_user_email
SETTINGS_user_given_name
SETTINGS_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

1

1. Define type of an environment where the application is to be provisioned. • 8
10. Write configuration scripts. • 12
11. Prepare package contents listing. • 13
12. Create an application package structure. • 13
13. Archive the application package files. • 13
14. Validate the application package. • 14

2

2. Define a model of service provisioning. • 9

3

3. Define application structure. • 9

4

4. Define application services hierarchy. • 10

5

5. Define an application licensing procedure. • 10

6

6. Define technologies used by your application. • 10

7

7. Define presentation details and settings for application and its services. • 11

8

8. Define application content delivery method. • 11

9

9. Create a draft of metadata file. • 11

A

- About Application Packaging Standard • 4
- About This Guide • 5
- Appendix A. WordPress Sample Metadata File • 33
- Appendix B. Sugar CRM Sample Metadata File • 37

APSLint Utility • 28

C

- Controller Operations on Packages • 30
- Creating Application Instance • 31

F

Feedback • 7

I

Introduction • 4

M

- Managing APSLint for Linux/Unix • 29
- Managing APSLint for Windows • 29

P

- Packaging Instruction • 8
- Packaging Open-Xchange Application • 24
- Packaging Scenarios • 15
- Packaging Sugar CRM Application • 19
- Packaging WordPress Application • 16

R

Removing Application Instance • 32

S

- Sample Environment Variables • 36, 44, 53
- Sample Metadata File • 47

T

Typographical Conventions • 6

U

- Updating Application Instance • 31
- Useful Links • 6

V

Validating Application Package • 28