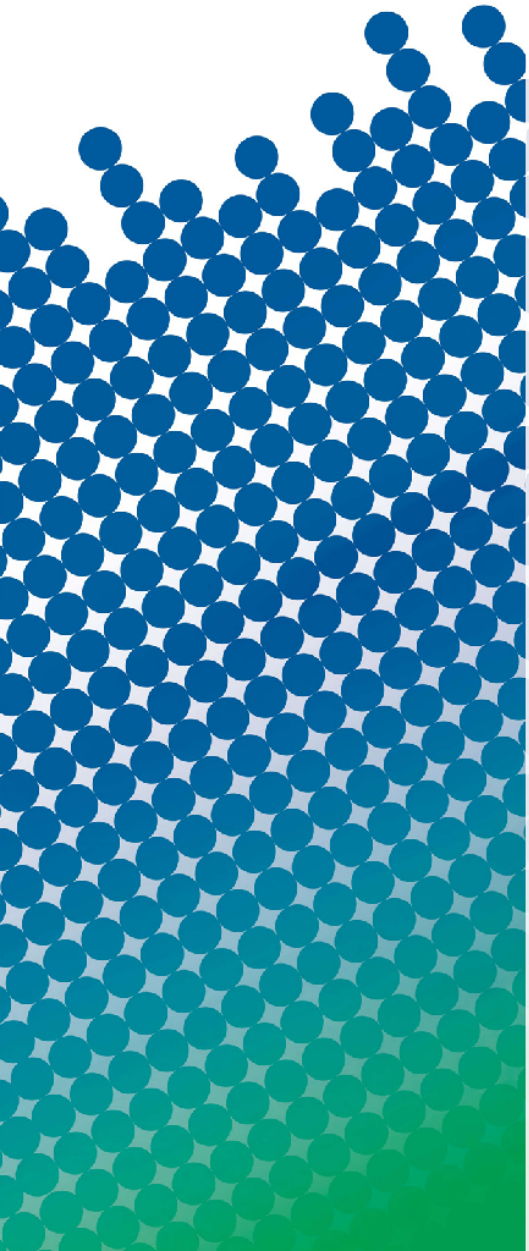

# OpenClovis ASP Runtime Director User Guide

OpenClovis, Inc.

www.openclovis.com

Doc. No. OCDOC-000107000200-02

2009-July-26 • Release 1.1

**For OpenClovis Software Development Kit (SDK) Release 3.0**

**Copyright © 2009 OpenClovis Inc.**

**All rights reserved**

This document contains proprietary and confidential information of OpenClovis Inc., and may not be used, modified, copied, reproduced, disclosed or distributed in whole or in part except as authorized by OpenClovis Inc. This document is intended for informational use and planning purposes only. All planned features, specifications, and content are subject to change without notice.

**Third-Party Trademarks**

Sun, Sun Microsystems, and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. UNIX is a registered trademark of The Open Group. Windows is a registered trademark of Microsoft Corporation in the United States and/or other countries. CLEI is a trademark of Telcordia Technologies, Inc. Adobe, Acrobat, and Acrobat Reader are registered trademarks of Adobe Systems, Inc. All other trademarks, service marks, product names, or brand names mentioned in this document are the property of their respective owners.

**Government Use**

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in FAR 12.212 (Commercial Computer Software-Restricted Rights) and DFAR 227.7202 (Rights in Technical Data and Computer Software), as applicable.

**Note**: This document is not subject of the GPL license, even if you have obtained this document as a part of the GPL-ed version of OpenClovis SDK.

# Contents

# Preface

*OpenClovis ASP Web Director (AWD) User Guide* provides information about the usage of OpenClovis Web Director, a web-based cluster and chassis management system from OpenClovis Inc. that complements the OpenClovis ASP (Application Service Platform). This guide helps you to utilise the OpenClovis Web Director in your own environment.

OpenClovis ASP Web Director is designed to simplify and accelerate the development of Telecom application software over OpenClovis platform. It provides a simple and powerful mechanism to examine and modify an online ASP cluster, to share critical chassis resources, install and manage new applications, and dynamically modify the ASP Information Model.

The OpenClovis ASP Web Director is also a "living" case study for customers to use when creating a web-based EMS (element management system) or network management glue layer. It can be easily rebranded to provide essentially an off-the-shelf EMS for ASP deployments, customized to provide enhanced EMS functionality, or used as a reference when integrating OpenClovis ASP into an existing EMS framework.

## Audience

OpenClovis ASP Web Director (AWD) User Guide is designed for system integrators, designers, and system architects. To use this OpenClovis product, you must be aware of the fundamentals of operation, management, and configuration of telecommunication and networking domains. You must also be familiar with Python programming, the OpenClovis ASP product, and have a basic knowledge of Linux.

# Documentation Conventions

This guide uses different fonts and symbols to differentiate between document elements and types of information. These conventions are summarized in the following table:

| Notation | Description |
|---|---|
| `Code` | This font denotes the source code provided in various examples. |
| Cross reference | This font denotes a hyperlink. You can click on the hyperlink text to access the reference location, which can be either a section within the User Guide or a URL link. A cross reference refers to a section name accesses the first page of that section. |
| **Bold Text** | Menu items and button names. |
| *Italic Text* | Variables for which you enter values. |

 This indicates the presence of notes or annotations, related to the context of the document.

 This indicates that certain precautionary measures must be taken before performing a particular task.

 This indicates that additional information is provided to you.

# Related Documentation

For additional information about OpenClovis products, refer to the following guides:

- **OpenClovis Release Notes** provides information about the software and the hardware required to install OpenClovis Application Service Platform (ASP) and Integrated Development Environment (IDE). It contains the additional features and enhancements of the product since the previous release. It also summarizes the issues and limitations of the product and provides workarounds wherever applicable.
- **OpenClovis SA Forum Compliance** describes the level of compliance of OpenClovis ASP and its Application Programming Interface (API) with the relevant Service Availability Forum Specifications.
- **OpenClovis Installation Guide** provides the system requirements, installation procedure for OpenClovis ASP, IDE, and the Evaluation System.
- **OpenClovis Sample Application Tutorial** provides the steps to create and build a sample model using OpenClovis IDE and OpenClovis ASP. It also provides troubleshooting information for this process. This provides the logical first step in understanding the OpenClovis offering.
- **OpenClovis Evaluation System User Guide** provides all the required information to configure and run the sample models packaged within the Evaluation System. This document also provides good understanding of OpenClovis ASPs functionality. This is the natural follow on to the *OpenClovis Sample Application Tutorial* as it builds on the example created in that document.
- **OpenClovis SDK User Guide** provides information about OpenClovis Application Service Platform (ASP) architecture, various OpenClovis ASP components, and their interactions. This guide helps you to configure the OpenClovis ASP components, compile, and execute the OpenClovis ASP code to build your custom application.
- **OpenClovis Log Tool User Guide** provides information about the usage of OpenClovis Log Tool. OpenClovis Log Tool is an interactive utility that allows you to view binary log files in a readable format and hence monitor system errors, warnings, and other log information. Log Tool allows you to format the `.log` files and filter them to view the required entries.
- **OpenClovis API Reference Guide** is provided for each component. It describes the Application Programming Interface (API), Service Model, and Management Information Model of the various OpenClovis Application Service Platform (ASP) services. It helps the developer to understand the capabilities of the ASP services and the APIs provided by these services.
- **OpenClovis ASP Console Reference Guide** provides details about managing applications built on OpenClovis ASP using the ASP runtime debug console commands. ASP Console commands can be used to manage, monitor, and test your application.

For additional information about TurboGears, the web application mega-framework used by the OpenClovis ASP Web Director please refer to:

- Turbogears web site at http://www.turbogears.org.

# OpenClovis Online Technical Support

OpenClovis customers and partners can register on our Web site and receive personalized services and information. If you need any support or assistance while working on OpenClovis products, you can access the following: URL: http://www.openclovis.com. Send your queries to: support@openclovis.com. Open source community support is available at: http://www.openclovis.org/forum.

# Documentation Feedback

We are interested in hearing from our customers on the documentation provided with the product. Let us know your comments and suggestions on improving the documentation at OpenClovis Inc. Send your comments to: support@openclovis.com. Post your feedback on documentation at: http://www.openclovis.org/forum. {{AwdDocHeader}}

CHAPTER 1

# Overview

The following diagram show the major functional blocks provided by the ASP Web Director.

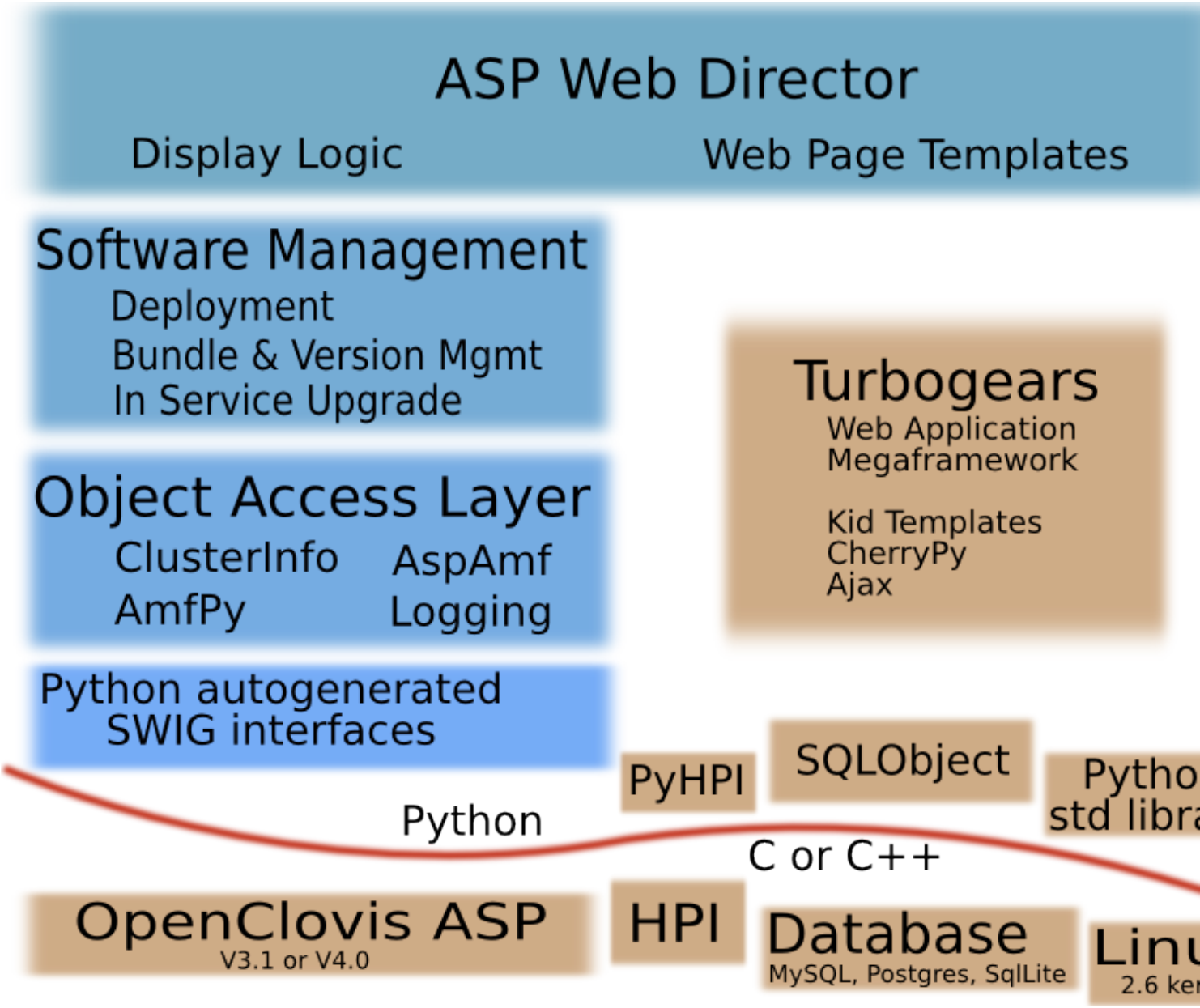


Figure 1.1: ASP Web Director High Level Architecture Diagram

The complete AWD package includes everything above the red curve that denotes the Python/C boundary. Above the red curve, blue boxes denote software created by OpenClovis while brown boxes show the free open source software that the AWD utilizes.

For explanatory purposes, the fundamental components of your cluster are shown below the red curve. These include the OpenClovis ASP (required), HPI (optional and it is bundled with ASP), a SQL database, and Linux (or other OS supported

by ASP).

## 1.1 Architecture Block Descriptions

CHAPTER 2

# ASP Web Director

The ASP Web Director itself is a thin layer that simply implements the GUI look and feel and calls the underlying APIs. It uses the TurboGears web application framework and templating system to serve particular web pages. To facilitate rebranding, customization, and extensibility, this layer is carefully constructed to contain only the web logic, and not any ASP functionality.

All subsequent sections describe the APIs available to any application. These APIs are described in detail in the ASP Web Director API document.

CHAPTER 3

# Software Management

The software management layer consists of APIs that handle software deployment onto any node, software bundle and version management, and in-service software upgrade (ISSU). This layer uses the Object Access Layer to access the ASP middleware and Linux services to deploy software.

CHAPTER 4

# Object Access Layer

This set of APIs presents the cluster Information Model as a set of objects and methods, instead of presenting it functionally as occurs in the lower layers.

## 4.1 ClusterInfo

The ClusterInfo library presents the AMF Information Model (both configuration and current state) as a set of objects. Since the Information Model naturally consists of objects (Service Groups, Service Units, and Components for example) this abstraction greatly simplifies the effort required to access the data.

Also included are routines that create internally consistent groups of Information Model entities. For example, a single function call will return a complete SAF Service Group hierarchy (Service Group, Service Unit, Components, Service Instances, and Component Service Instances) for a new application that is both internally consistent and consistent with the existing Information Model. This set of objects is not yet installed into the system, allowing you to tweak and modify their configuration before "committing" the group.

## 4.2 AspAmf

The AspAmf library contains all functions that modify the AMF Information Model. These functions are tightly integrated with the ClusterInfo entities and encapsulate a tremendous amount of functionality as compared to the underlying functional interface. For example, modifying the Information Model to implement common operations such as adding or removing a Service Group can require

hundreds of function calls at the AMF layer. However this can be done in a single AspAmf API call called "InstallApp". This single call takes as a parameter a list of AMF entities, and (from the user's perspective) it simply adds them into the AMF Information Model. But actually doing so is not so simple! The function ensures that the entities are created if they do not exist, or modified to match a new configuration if they already exist. It also ensures that the entities are created in the proper order and that they correctly reference each other and existing objects within the Information Model. The end result is a very simple API to accomplish a complex task.

The AspAmf library also contains APIs to query entity state and start and stop them.

## 4.3 AmfPy

The amfpy library allows a Python program to register itself as an AMF component and therefore receive AMF component control callbacks (start, stop, CSI assignment, etc). Both extending and embedding is supported – that is Python can either be embedded as a small interpreter into an ASP-aware C program, or it can be run standalone and the amfpy functionality can be "import"ed. Python programs can either be spawned automatically by the AMF's component manager, or can be run from the command line (using the asp_run tool provided by the ASP) and register with the AMF dynamically.

## 4.4 SWIG Interfaces

Most OpenClovis ASP functions are directly available at the Python interpreter using an automatically generated Python-to-C translation layer. However, it is often inconvenient to use these functions since they correspond one-to-one with equivalent C functions. Additionally if the function uses complex pointer or object relationships (such as forced casting of pointers) it is sometimes very difficult or even impossible to express the parameters in a type-safe language like Python. Therefore it is recommended that this layer only be used if the functionality does not exist in the higher layers. {{AwdDocHeader}}

CHAPTER 5

# Deployment and Packaging

The ASP Web Director GUI is provided in the form of a gzipped tar file (.tgz). You should open this file (tar xvfz {filename}) in the ASP user's (generally "root") home directory or the ASP base directory. It will create a directory called aspwebdirector-{version}-{OS}-{arch}. The rest of this document will refer to this directory using the symbol {AWDBASE}.

## 5.1 Integration within your IDE model

The ASP Web Director GUI can be integrated within your IDE-based model by creating a 1+1 Service group with 1 component on 1 or 2 ASP nodes. The component executable should reference a script that simply executes the AWD binary ({AWDBASE}/bin/aspwebdirector). Before running your model, you must run the {AWDBASE}/install script to ensure that all of the AWD requirements are installed.

## 5.2 Dynamic Installation

Additionally, the ASP Web Director GUI can be dynamically installed into a running ASP chassis. To do so, detar the package and run the ./install and then the ./start scripts. These scripts do not need to be run again (even after an cluster reboot) unless you remove the ASP configuration (located in the "var" directory of your ASP installation). If the ASP configuration is deleted, you will need to run the "./start" script again to insert the AWD service group into the ASP Application Management Framework (AMF).

## 5.3 Library Integration

The ASP Web Director Libraries can be deployed within your management application by running your management application (or agent) on an ASP node and running a Python interpreter within it. Embedding a Python interpreter in this manner is very simple. You may use the AWD GUI as an example of how to do so, or you may read the "Extending and Embedding Python" chapter of the python documentation located at http://www.python.org.

## 5.4 Standalone Python

Additionally, you may run the ASP Web Director Libraries stand-alone in its own Python shell. To do so, first start ASP and then add the AWD "bin" directory to your "PYTHONPATH" environment variable (see Python documentation for more details) so you can "import" the AWD libraries. Next, from the ASP base directory run "bin/asp_run python". You should see the python prompt. Next run:

```
>>> import asppy
>>> asppy.initializeAmf()
```

At this point, you should be able to access the API. For example:

```
>>> import clusterinfo
>>> print [x.name for x in clusterinfo.ci.nodes]
```

Additionally, if you choose to not run the GUI, you do not need most of the applications that are installed as part of the {AWDBASE}/install script (although it does not hurt to install them). All that is needed is "pexpect" which is used for deployment (to automate the "scp" command). You may see how to install this by hand by looking at the {AWDBASE}/3rdparty/Makefile file. If you are not using the GUI or the deployment features, you do not need to install any prerequisites. {{AwdDocHeader}}

CHAPTER 6

# Software Management

Software Management consists of the creation and management of software bundles, software version tracking, deployment, removal, and upgrade. The software management functionality is available as a set of Python APIs delivered as part of the AWD API. Individual function documentation is available in the AWD API reference guide.

## 6.1 Software Lifecycle

This section describes the application software lifecycle from the perspective of the AWD GUI for clarity. However, understand that AWD APIs exist to access every stage of this process independently of the GUI and of other stages. Therefore an independently written EMS can selectively modify or override any or all of these stages.

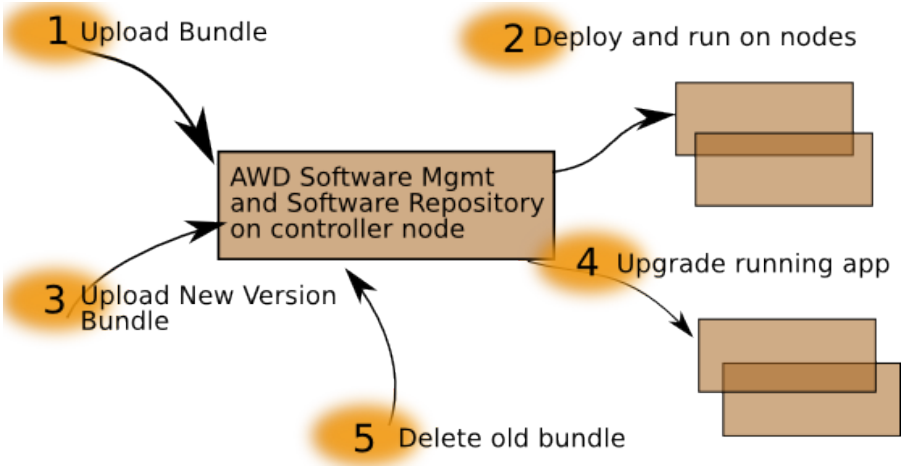


Figure 6.1: Software Lifecycle

1. Upload: Application Software is delivered in the form of a "bundle", which is a tarred, gzipped (.tgz) archive file containing the application. The application author generates this bundle as part of the application "build" process. This bundle file is uploaded to the ARD server (controller nodes) and is stored in the AWD's software repository.

2. Deployment: Once the application software is in the AWD bundle manager, it can be deployed to nodes. In this step the software is automatically copied to each node and the appropriate SAF AMF Entities are created to model the application

3. New Version: A new version (bugfix or major change) of the application is delivered as another bundle in the same manner as the original upload.

4. Upgrade: An upgrade can occur when a newer version of application software exists in the bundle manager then is running on the nodes. The user initiates an upgrade, and can do so selectively – that is, for particular deployments (service groups) and not others. Multiple upgrades can be ongoing at the same time.

5. Removal: When no deployments are currently using a particular version of application software, that software can be deleted from the bundle manager.

## 6.2 Software Library Overview

CHAPTER 7

# AspApp

Software can be provided to the AWD in a single bundle file, which is essentially a Linux .tgz (gzipped tar archive) that contains specific files and a well-defined directory layout. Most importantly, at the top level there exists an XML file called "appcfg.xml". This file describes the application contained in the bundle file, and includes details such as its current version and required redundancy model (see Application Bundle Format ).

The "aspApp" module manages bundles in the system. It is notified of a bundle file via and API call that simply takes the name of the file as a parameter (AppDb.NewAppFile()). The module then opens the file, parses the appcfg.xml file and creates an "AppFile" object that corresponds to this software bundle. This object contains data members and methods that make it easy for an application to access all information about the bundle. The module also places this "AppFile" object into a global database of all application bundles, so that it is easy to understand its relationship to other bundles that are also in the system (such as to prior versions of the same application). It also connects this application bundle to the existing ASP state, so you can easily determine whether this application bundle has already been deployed on the system and exactly which Service Groups are running it.

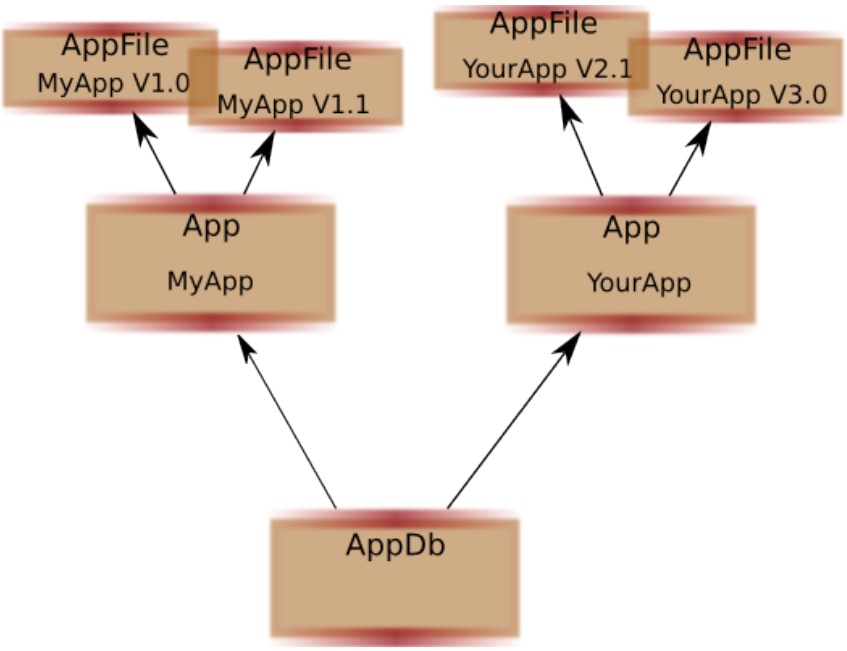


Figure 7.1: OpenClovis ASP Web Director Application Management Object Hierarchy

CHAPTER 8

# AppDeploy

The "appDeploy" module handles software deployment. It contains a single API called "deploy" that takes some data members from an AppFile object (basically the bundle extraction location and the deployment configuration), a list of nodes to deploy to, and some flags controlling various aspects of the operation. The function does not take the AppFile directly for two reasons:

- It is usable on software that is not part of the software bundle management.
- You may want to modify the software's deployment configuration.

Using this function, you may either copy the software onto all specified nodes, create the required SAF AMF entities in the information model, or do both simultaneously (depending on the flags passed).

You may also select whether you want an exception raised upon error (such as inability to deploy the software to a node), or whether you want to continue, returning all issues at the end of the operation.

All together, this results in an incredibly powerful but simple deployment facility!

CHAPTER 9

# Application Removal

Applicaton removal is extremely simple. You simple choose the SAF entities to be deleted, and call the "DeleteEntities" member of the aspAmf module. The system makes sure that applications are shut down before deletion, etc. And to make the process even simpler, there is a helper function in the clusterinfo module called "getDependentEntities" that will return a list of all entities that are wholly dependent on the entity passed in. These two functions can be used in conjunction to create a very powerful application or entity removal system. For example, to delete a Service Group named "mySg" the following code could be used:

```
entities = ci.getDependentEntities(ci.entities["mySg"])
amf.DeleteEntities(entities)
```

[In the code above, "ci" is the global clusterinfo database located at "clusterinfo.ci" and "amf" is an instance of an aspAmf session (aspAmf.Session())].

This will delete the Service Group and all Service Units, Components, Service Instances, and Component Service Instances within that Service Group.

This idiom works for all SAF entities. For example, to delete a node named "myNode" the exact same code could be used (other then the entity name):

```
entities = ci.getDependentEntities(ci.entities["myNode"])
amf.DeleteEntities(entities)
```

This will delete the node and all Service Units and Components running on that node. If a Service Group is running on several nodes including the deleted node, only the Service Unit and Components on the node will be affected – the Service Group does not need to be taken out of service, and work (SAF SIs) will automatically be transferred to Service Units that are not being deleted!

CHAPTER 10

# Application Upgrade

Application Upgrade is implemented in the "upgrade" module. This module contains an upgrade manager (UpgradeMgr) that is essentially a container for objects that describe the upgrade state of individual Service Groups (UpgradeSg).

To initiate an upgrade one first must provide the software bundle file containing the new version to the bundle manager (aspApp, described above). Next, one simply adds the service group to the upgrade manager by calling the UpgradeMgr.add() member function with the service group entity (accessed via clusterinfo). At this point an UpgradeSg object is created but not yest started. It is now possible to configure the specifics of the upgrade operation by calling UpgradeSg member functions and modifying member variables. For the upgrade algorithm (single step or rolling) can be selected by modifying the "upMethod" member variable.

Finally, an upgrade can be initiated by calling the UpgradeSg.Upgrade() member function. This function accepts as a parameter an AppFile (see above for a description of AppFile) object that describes exactly what target software to upgrade to. The upgrade system takes care of automatically deploying the software to the required nodes, shutting down Service Units, and deleting and recreating SAF AMF entities.

Its THAT simple!

## 10.1 Additional Upgrade APIs

There are also additional APIs that allow a programmer to have more control over the upgrade process. The "Upgrade" member function is actually a thin wrapper around a "generator" function (http://docs.python.org/tutorial/classes.html#generators)

that "returns" each time a single step in the upgrade is complete. So you can directly call the generator function if you need to execute some code at each step in the upgrade process. The AWD GUI uses this feature to implement step-by-step guided upgrade.

The upgrade objects are also derived from a single class called ChangeTracker. The ChangeTracker class has an API that allows threads to block until a state change has occurred (ChangeTracker.changeWait). As the upgrade proceeds, waiting threads are unblocked each time the upgrade changes the state of the cluster, allowing application code to be written to observe these state changes. For example, the AWD GUI uses this technique to report the upgrade state changes to the web browser as they occur, allowing the user to watch the upgrade's progress. {{AwdDocHeader}}

## 10.2 Software Management

Software Management consists of the creation and management of software bundles, software version tracking, deployment, removal, and upgrade. The software management functionality is available as a set of Python APIs delivered as part of the AWD API. Individual function documentation is available in the AWD API reference guide.

CHAPTER 11

# Software Lifecycle

This section describes the application software lifecycle from the perspective of the AWD GUI for clarity. However, understand that AWD APIs exist to access every stage of this process independently of the GUI and of other stages. Therefore an independently written EMS can selectively modify or override any or all of these stages.

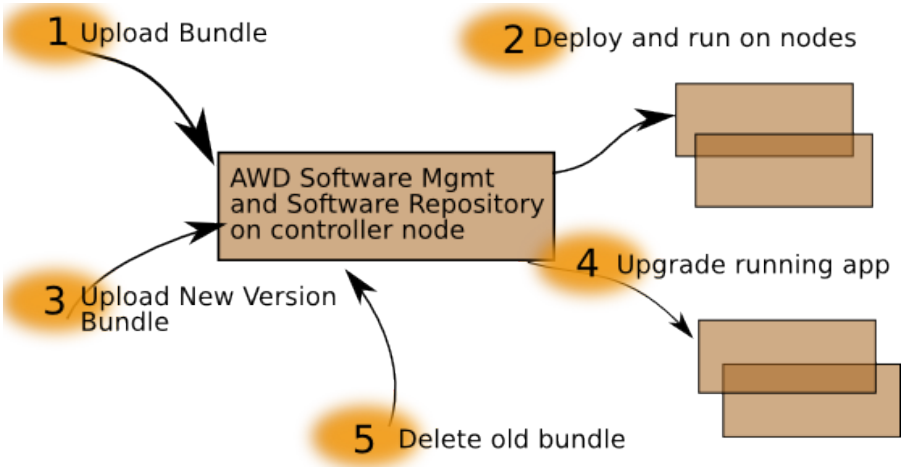


Figure 11.1: Software Lifecycle

1. Upload: Application Software is delivered in the form of a "bundle", which is a tarred, gzipped (.tgz) archive file containing the application. The application author generates this bundle as part of the application "build" process. This bundle file is uploaded to the ARD server (controller nodes) and is stored in the AWD's software repository.

2. Deployment: Once the application software is in the AWD bundle manager, it can be deployed to nodes. In this step the software is automatically copied to each node and the appropriate SAF AMF Entities are created to model the application

3. New Version: A new version (bugfix or major change) of the application is delivered as another bundle in the same manner as the original upload.

4. Upgrade: An upgrade can occur when a newer version of application software exists in the bundle manager then is running on the nodes. The user initiates an upgrade, and can do so selectively – that is, for particular deployments (service groups) and not others. Multiple upgrades can be ongoing at the same time.

5. Removal: When no deployments are currently using a particular version of application software, that software can be deleted from the bundle manager.

CHAPTER 12

# Software Library Overview

## 12.1 AspApp

Software can be provided to the AWD in a single bundle file, which is essentially a Linux .tgz (gzipped tar archive) that contains specific files and a well-defined directory layout. Most importantly, at the top level there exists an XML file called "appcfg.xml". This file describes the application contained in the bundle file, and includes details such as its current version and required redundancy model (see Application Bundle Format ).

The "aspApp" module manages bundles in the system. It is notified of a bundle file via and API call that simply takes the name of the file as a parameter (AppDb.NewAppFile()). The module then opens the file, parses the appcfg.xml file and creates an "AppFile" object that corresponds to this software bundle. This object contains data members and methods that make it easy for an application to access all information about the bundle. The module also places this "AppFile" object into a global database of all application bundles, so that it is easy to understand its relationship to other bundles that are also in the system (such as to prior versions of the same application). It also connects this application bundle to the existing ASP state, so you can easily determine whether this application bundle has already been deployed on the system and exactly which Service Groups are running it.

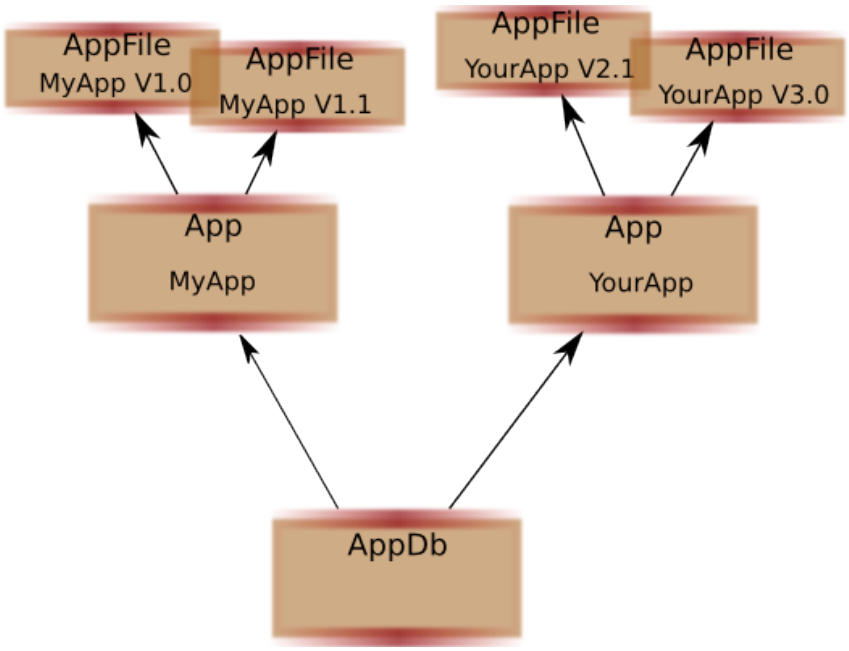


Figure 12.1: OpenClovis ASP Web Director Application Management Object Hierarchy

## 12.2 AppDeploy

The "appDeploy" module handles software deployment. It contains a single API called "deploy" that takes some data members from an AppFile object (basically the bundle extraction location and the deployment configuration), a list of nodes to deploy to, and some flags controlling various aspects of the operation. The function does not take the AppFile directly for two reasons:

- It is usable on software that is not part of the software bundle management.
- You may want to modify the software's deployment configuration.

Using this function, you may either copy the software onto all specified nodes, create the required SAF AMF entities in the information model, or do both simultaneously (depending on the flags passed).

You may also select whether you want an exception raised upon error (such as inability to deploy the software to a node), or whether you want to continue, returning all issues at the end of the operation.

All together, this results in an incredibly powerful but simple deployment facility!

## 12.3 Application Removal

Applicaton removal is extremely simple. You simple choose the SAF entities to be deleted, and call the "DeleteEntities" member of the aspAmf module. The system makes sure that applications are shut down before deletion, etc. And to make the process even simpler, there is a helper function in the clusterinfo module called "getDependentEntities" that will return a list of all entities that are wholly dependent on the entity passed in. These two functions can be used in conjunction to create a very powerful application or entity removal system. For example, to delete a Service Group named "mySg" the following code could be used:

```
entities = ci.getDependentEntities(ci.entities["mySg"])
amf.DeleteEntities(entities)
```

[In the code above, "ci" is the global clusterinfo database located at "clusterinfo.ci" and "amf" is an instance of an aspAmf session (aspAmf.Session())].

This will delete the Service Group and all Service Units, Components, Service Instances, and Component Service Instances within that Service Group.

This idiom works for all SAF entities. For example, to delete a node named "myNode" the exact same code could be used (other then the entity name):

```
entities = ci.getDependentEntities(ci.entities["myNode"])
amf.DeleteEntities(entities)
```

This will delete the node and all Service Units and Components running on that node. If a Service Group is running on several nodes including the deleted node, only the Service Unit and Components on the node will be affected – the Service Group does not need to be taken out of service, and work (SAF SIs) will automatically be transferred to Service Units that are not being deleted!

## 12.4 Application Upgrade

Application Upgrade is implemented in the "upgrade" module. This module contains an upgrade manager (UpgradeMgr) that is essentially a container for objects that describe the upgrade state of individual Service Groups (UpgradeSg).

To initiate an upgrade one first must provide the software bundle file containing the new version to the bundle manager (aspApp, described above). Next,

one simply adds the service group to the upgrade manager by calling the UpgradeMgr.add() member function with the service group entity (accessed via clusterinfo). At this point an UpgradeSg object is created but not yest started. It is now possible to configure the specifics of the upgrade operation by calling UpgradeSg member functions and modifying member variables. For the upgrade algorithm (single step or rolling) can be selected by modifying the "upMethod" member variable.

Finally, an upgrade can be initiated by calling the UpgradeSg.Upgrade() member function. This function accepts as a parameter an AppFile (see above for a description of AppFile) object that describes exactly what target software to upgrade to. The upgrade system takes care of automatically deploying the software to the required nodes, shutting down Service Units, and deleting and recreating SAF AMF entities.

Its THAT simple!

**Additional Upgrade APIs**

There are also additional APIs that allow a programmer to have more control over the upgrade process. The "Upgrade" member function is actually a thin wrapper around a "generator" function (http://docs.python.org/tutorial/classes.html#generators) that "returns" each time a single step in the upgrade is complete. So you can directly call the generator function if you need to execute some code at each step in the upgrade process. The AWD GUI uses this feature to implement step-by-step guided upgrade.

The upgrade objects are also derived from a single class called ChangeTracker. The ChangeTracker class has an API that allows threads to block until a state change has occurred (ChangeTracker.changeWait). As the upgrade proceeds, waiting threads are unblocked each time the upgrade changes the state of the cluster, allowing application code to be written to observe these state changes. For example, the AWD GUI uses this technique to report the upgrade state changes to the web browser as they occur, allowing the user to watch the upgrade's progress. {{AwdDocHeader}}

## 12.5 Application Bundle Format

An application bundle is a gzipped tar file (.tgz, open via the "tar xvfz <file>" command) that contains at a minimum the following layout:

```
appcfg.xml
deploy (a directory)
```

```
bin  (a directory)
  <application binary files>
etc
  <application configuration files>
```

The appcfg.xml file describes the contents of the application bundle. It is an xml V1.1 document with a simple layout. As a simplification to standard XML, attributes and child tags are considered equivalent and the special attribute "value" is equivalent to child content. The purpose is to allow the xml author to use whichever form is more convenient. This concept is most easily understood by noting that all of the following are equivalent config files: