Cloud Application Management for Platforms Version 1.1

Committee Specification Draft 05

27 August 2014

Specification URIs

This version:

http://docs.oasis-open.org/camp/camp-spec/v1.1/csd05/camp-spec-v1.1-csd05.pdf (Authoritative)
http://docs.oasis-open.org/camp/camp-spec/v1.1/csd05/camp-spec-v1.1-csd05.html
http://docs.oasis-open.org/camp/camp-spec/v1.1/csd05/camp-spec-v1.1-csd05.doc

Previous version:

http://docs.oasis-open.org/camp/camp-spec/v1.1/csprd02/camp-spec-v1.1-csprd02.pdf (Authoritative)
http://docs.oasis-open.org/camp/camp-spec/v1.1/csprd02/camp-spec-v1.1-csprd02.html
http://docs.oasis-open.org/camp/camp-spec/v1.1/csprd02/camp-spec-v1.1-csprd02.doc

Latest version:

http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.pdf (Authoritative)
http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.html
http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.doc

Technical Committee:

OASIS Cloud Application Management for Platforms (CAMP) TC

Chair:

Martin Chapman (martin.chapman@oracle.com), Oracle

Editors:

Jacques Durand (jdurand@us.fujitsu.com), Fujitsu Limited
Adrian Otto (adrian.otto@rackspace.com), Rackspace Hosting, Inc.
Gilbert Pilz (gilbert.pilz@oracle.com), Oracle
Tom Rutt (trutt@us.fujitsu.com), Fujitsu Limited

Additional artifacts:

This prose specification is one component of a Work Product that also includes:

- Non-normative auxiliary files: http://docs.oasis-open.org/camp/camp-spec/v1.1/csd05/camp-type-definitions/

Related work:

This specification is related to:

- *Cloud Application Management for Platforms (CAMP) Test Assertions Version 1.1*. Edited by Jacques Durand, Gilbert Pilz, Adrian Otto, and Tom Rutt. Latest version: http://docs.oasis-open.org/camp/camp-ta/v1.1/camp-ta-v1.1.html.

Abstract:

This document defines the artifacts and APIs that need to be offered by a Platform as a Service (PaaS) cloud to manage the building, running, administration, monitoring and patching of applications in the cloud. Its purpose is to enable interoperability among self-service interfaces to PaaS clouds by defining artifacts and formats that can be used with any conforming cloud and

enable independent vendors to create tools and services that interact with any conforming cloud using the defined interfaces. Cloud vendors can use these interfaces to develop new PaaS offerings that will interact with independently developed tools and components.

Status:

This document was last revised or approved by the OASIS Cloud Application Management for Platforms (CAMP) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=camp#technical.

TC members should send comments on this specification to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at https://www.oasis-open.org/committees/camp/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (https://www.oasis-open.org/committees/camp/ipr.php).

Citation format:

When referencing this specification the following citation format should be used:

[CAMP-v1.1]

Cloud Application Management for Platforms Version 1.1. Edited by Jacques Durand, Adrian Otto, Gilbert Pilz, and Tom Rutt. 27 August 2014. OASIS Committee Specification Draft 05. http://docs.oasis-open.org/camp/camp-spec/v1.1/csd05/camp-spec-v1.1-csd05.html. Latest version: http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.html.

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see https://www.oasis-open.org/policies-guidelines/trademark for above guidance.

Table of Contents

1 Introduction

1.1 Overview

Platform as a Service (PaaS) is a term that refers to a type of cloud computing in which the service provider offers customers/consumers access to one or more instances of a running application computing platform or application service stack. NIST defines PaaS [SP800-145] as a "service model" with the following characteristics:

> The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.

There are multiple commercial PaaS offerings in existence using languages such as Java, Python and Ruby and frameworks such as Spring, Django and Rails. Although these offerings differ in such aspects as programming languages, application frameworks, etc., there are inherent similarities in the way they manage the lifecycle of the applications that are targeted for, and deployed upon them. *The core proposition of this specification is that these similarities can be leveraged to produce a generic application and platform management API that is language, framework, and platform neutral.*

For PaaS consumers this management API would have the following benefits:

- "Portability between clouds" is emerging as one of the primary concerns of cloud computing. By standardizing the management API for the use cases around deploying, stopping, starting, and updating applications, this specification increases consumers' ability to port their applications between PaaS offerings.
- It is likely that implementations of this specification will appear as plugins for application development environments (ADEs) and application management systems. Past experience has shown that, over time, such generic implementations are likely to receive more attention and be of higher quality than the implementations written for solitary, proprietary application management interfaces.

For PaaS providers this management API would have the following benefits:

- Because the strength and features of a PaaS offering's application management API are unlikely to be perceived as key differentiators from other PaaS offerings, the existence of a consensus management API allows providers to leverage the experience and insight of the specification's contributors and invest their design resources in other, more valuable areas.
- By increasing the portability of applications between PaaS offerings, this management API helps "grow the pie" of the PaaS marketplace by addressing one of the key pain points for PaaS consumers.

1.2 Purpose

This document defines the artifacts and APIs that need to be offered by a Platform as a Service (PaaS) cloud to manage the building, running, administration, monitoring and patching of applications in the cloud. Its purpose is to enable interoperability among self-service interfaces to PaaS clouds by defining artifacts and formats that can be used with any conforming cloud and enable independent vendors to create tools and services that interact with any conforming cloud using the defined interfaces. Cloud vendors can use these interfaces to develop new PaaS offerings that will interact with independently developed tools and components.

The following is a non-exhaustive list of the use cases which are supported by this specification.

- Building and packaging an application in a local Application Development Environment (ADE)
- Building an application in an ADE running in the cloud

- Importing a Platform Deployment Package into the cloud
- Uploading application artifacts into the cloud
- Run, stop, suspend, snapshot, and patch an application

1.3 Example (non-normative)

This example illustrates a scenario in which the application administrator wants to run and monitor an application.

The administrator does this by deploying the application, in the form of a Platform Deployment Package, to the platform. This is done by sending an HTTP POST request to the URL of the *assemblies resource* as shown below, where "/my_paas/assemblies" is this URL and "/my_paas/pkgs/1" is the location of the application package.

```
POST /my_paas/assemblies HTTP/1.1
Host: example.org
Content-Type: application/json
Content-Length: …

{
  "pdp_uri": "/my_paas/pkgs/1"
}
```

On receiving such a request the platform unpacks the package, parses and validates the Plan file, resolves the service dependencies described by that Plan, and starts the application. On successful start the platform creates a new resource representing the running application and provides the URL of that resource "/my_paas/apps/1" in the response as shown below.

```
HTTP/1.1 201 Created
Location: http://example.org/my_paas/apps/1
Content-Type: …
Content-Length: …

…
```

The administrator can now monitor the running application by sending an HTTP GET request to the resource that represents the running application, which was obtained in the previous step ("/my_paas/apps/1").

```
GET /my_paas/apps/1 HTTP/1.1
Host: example.org
```

The response contains the JSON representation of the running application as shown below.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: …

{
  "uri": "http://example.org/my_paas/apps/1",
  "name": "Hello Cloud App",
  "type": "assembly",
  "description": "Hello Cloud Application Running in a PaaS Env",
  "components": [
    {
      "href": "/my_paas/apps/1/acs/1", "target_name": "appComp1"
    },
    {
      "href": "my_paas/apps/1/acs/2", "target_name:: "appComp2"
    },
    {
      "href": "/my_paas/pcs/1", "target_name": "dbPlatComp"
    },
    {
      "href": "my_paas/pcs/2", "target_name": "msgBusPlatComp"
    }
  ]
}
```

1.4 Non-Goals

The interfaces exposed by the components and services in a PaaS system can be broadly split into two categories; functional interfaces and management interfaces. Functional interfaces are those that involve the specific utility provided by that component. For example, the interface used to submit a message to a message queuing service is as a functional interface. Management interfaces are those that deal with the administration of components. For example, the interface used to deploy and start an application on the platform is a management interface.

The specification of functional interfaces is out of scope for this document.

1.5 Actors

There are many actors for a PaaS environment. For the purposes of this specification we identify the following actors:

Application Developer: The person that builds and tests an application and presents the developed artifacts for deployment.

Application Administrator: The person that deploys applications and manages the application throughout its life-cycle.

Together these two actors make up the consumers of the management API described in this specification. This specification is intended mainly for Application Administrators, though it does constraint the artifacts that an Application Developer presents for deployment.

Platform Administrator: The person that manages the platform. This specification describes some of the functions of a Platform Administrator, though most of the functions of this actor are outside its scope.

Application End-User: A user of an application deployed on the platform. The interactions of the Application end-user and the application are outside the scope of this specification.

Extension Developer: The person who creates new Extensions for Platforms.

1.6 Terminology

1.6.1 Term Definitions

1.6.1.1 CAMP Provider (Provider)

A CAMP Provider (Provider) is an implementation of the service aspects of this specification.

1.6.1.2 CAMP Consumer (Consumer)

A CAMP Consumer (Consumer) is an implementation of the client aspects of this specification.

1.6.2 Keywords, Conventions, and Normative Text

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 [RFC2119].

Upper case versions of the RFC 2119 keywords are used in this document for emphasis. All text except examples, unless otherwise labeled, is normative. Normative statements that use these keywords have been highlighted as per this sentence. Each such statement has been given a unique tag in the following manner: **[EX-00]**. For convenience these statements have been tabulated and cross-indexed by their tags and appear in Appendix C. All examples, figures and notes in this document are informative. Unless marked otherwise text in this specification is normative.

See Section 8, “Conformance”, for details on Conformance to this specification.

1.7 Notational Conventions

The JSON and YAML descriptions that depict the representation of resources and the structure of Plans use a pseudo-schema notation with the following conventions:

- Characters are appended to items to indicate cardinality:
  - “?” (0 or 1)
  - “*” (0 or more)
  - “+” (1 or more)

  Absent any indication of cardinality, the default cardinality of an element is “exactly 1”. The scope of these operators is the entire line on which they appear.
- Vertical bars, “|”, denote choice. For example, “a | b” means a choice between “a” and “b”.
- Parentheses, “(“ and “)”, are used to indicate the scope of the “|” operator.
- An expression in italics indicates a value whose type is indicated by the italicized expression. For example, “foo: *String*” indicates that the value of “foo” will be a String.
- Square brackets, “[]”, indicate an array of the type indicated by the expression preceding it. For example, “foo: *String[]*” indicates that the value of “foo” will be an array of Strings.
- An expression surrounded in angle brackets, “<” and “>”, indicates a value whose type is indicated either by some of field in the object or by other context information. For example, “foo: <*sensor_type*>” indicates that the type of the value of “foo” is provided by the value of the “sensor_type” attribute.

Note that the information presented in pseudo-schema is intended as a condensed guide and is subordinate to the textual descriptions of the nodes and objects that appear in those descriptions. In the event of a conflict (due to a typo or other editorial error) the text takes precedence over the pseudo-schema.

1.8 Specification Version

Each version of a CAMP specification is identified by a unique string termed the “Specification Version String”. The Specification Version String for this specification is “CAMP 1.1”.

1.8.1 Backwards Compatibility

This version of the CAMP specification is not backwards compatible with any previous version of the CAMP specification.

1.9 Normative References

[ISO 8601:2004] International Organization for Standardization, Geneva, Switzerland, “Data elements and interchange formats -- Information interchange - - Representation of dates and times”, March 2008. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=40874

[OVF] DMTF DSP0243, “Open Virtualization Format Specification 2.0.1”, http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.0.1.pdf

[RFC1952] Deutsch, P., "GZIP file format specification version 4.3", RFC 1952, May 1996. http://www.ietf.org/rfc/rfc1952.txt

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels”, BCP 14, RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999. http://www.ietf.org/rfc/rfc2616.txt

[RFC2818] E. Rescorla, “HTTP Over TLS”, RFC 2818, May 2000. http://www.ietf.org/rfc/rfc2818.txt

[RFC2388] Masinter, L., "Returning Values from Forms: multipart/form-data", RFC 2388, August 1998. http://www.ietf.org/rfc/rfc2388.txt

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005. http://www.ietf.org/rfc/rfc3986.txt

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006. http://www.ietf.org/rfc/rfc4346.txt

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006. http://www.ietf.org/rfc/rfc4627.txt

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008. http://www.ietf.org/rfc/rfc5246.txt

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, March 2010. http://www.ietf.org/rfc/rfc5789.txt

[RFC6902] Bryan, P., Ed., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Patch", RFC 6902, April 2013. http://www.ietf.org/rfc/rfc6902.txt

[SHA256] FIPS PUB 180-4, Federal Information Processing Standards Publication, “Secure Hash Standard (SHS) (FIPS PUB 180-4)”, 6.2, SHA-256. http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

[TAR] IEEE Std 1003.1, 2004 Edition, Standard for Information Technology - Portable Operating System Interface (POSIX) http://ieeexplore.ieee.org/servlet/opac?punumber=9158

[YAML 1.1] Oren Ben-Kiki, Clark Evans, Brian Ingerson, “YAML Ain’t Markup Language (YAML) Version 1.1, 2005-01-18”. http://yaml.org/spec/1.1/. Also archived at http://xml.coverpages.org/yaml-spec-v1.1-archive-copy.html.

[ZIP] ZIP File Format Specification, http://www.pkware.com/documents/APPNOTE/APPNOTE-6.3.0.TXT

1.10 Non-Normative References

[Git] The Software Freedom Conservancy, “Git, the fast version control system”, March 2012. http://git-scm.com/

[Keystone] OpenStack Foundation, “OpenStack Identity Service API v2.0 Reference”, July 2013. http://docs.openstack.org/api/openstack-identity-service/2.0/content/

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999. http://www.ietf.org/rfc/rfc2617.txt

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012. http://www.ietf.org/rfc/rfc6749.txt

[SP800-145] Peter Mell, Timothy Grance, “The NIST Definition of Cloud Computing”, Special Publication 800-145, September 2011. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[POM-Syntax] Tim O'Brien, Manfred Moser, John Casey, Brian Fox, Jason Van Zyl, Eric Redmond Larry Shatzer, “Maven: The Complete Reference”, 2008-2014, Section 3.3 POM Syntax. http://books.sonatype.com/mvnref-book/reference/

2 Concepts and Types

This section is informative.

2.1 Introduction

This specification defines the self-service management API that a Platform as a Service offering presents to the consumer of the platform. The API is the interface into a platform implementation layer that controls the deployment of applications and their use of the platform.

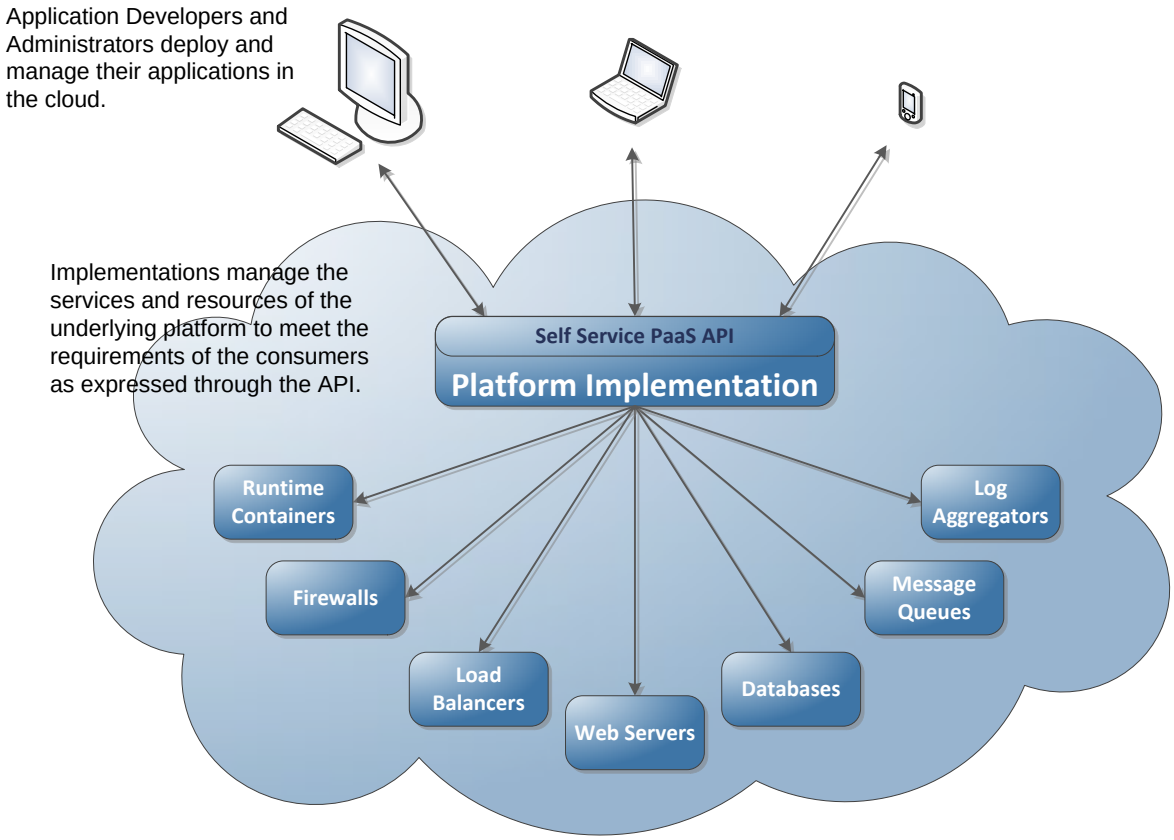

Figure 2-1: Typical PaaS Architecture

The figure above shows a typical architecture of a Platform as a Service cloud. The platform implementation is a management client of the underlying resources that transforms (through policies) the application requirements expressed by the Application Administrator into provisioning and other operations on those resources. The Platform Administrator manages the underlying hardware, storage, networks, and software services that make up the platform through existing administrative interfaces. Thus the Application Administrator is able to concentrate on their application and its deployment environment rather than having to be a systems administrator, database administrator and middleware administrator as well (as is the case with IaaS).

The goal of the management interface is to provide the PaaS consumer with a model that is as simple as possible, and yet still provides the essential elements that give them control over the deployment, execution, administration and metering of their application and its deployment environment.

2.2 Resources

The CAMP API is made up of resources in a REST protocol. The resources represent elements of the underlying system. The protocol enables interaction with the resources. The following are the main resources in the API:

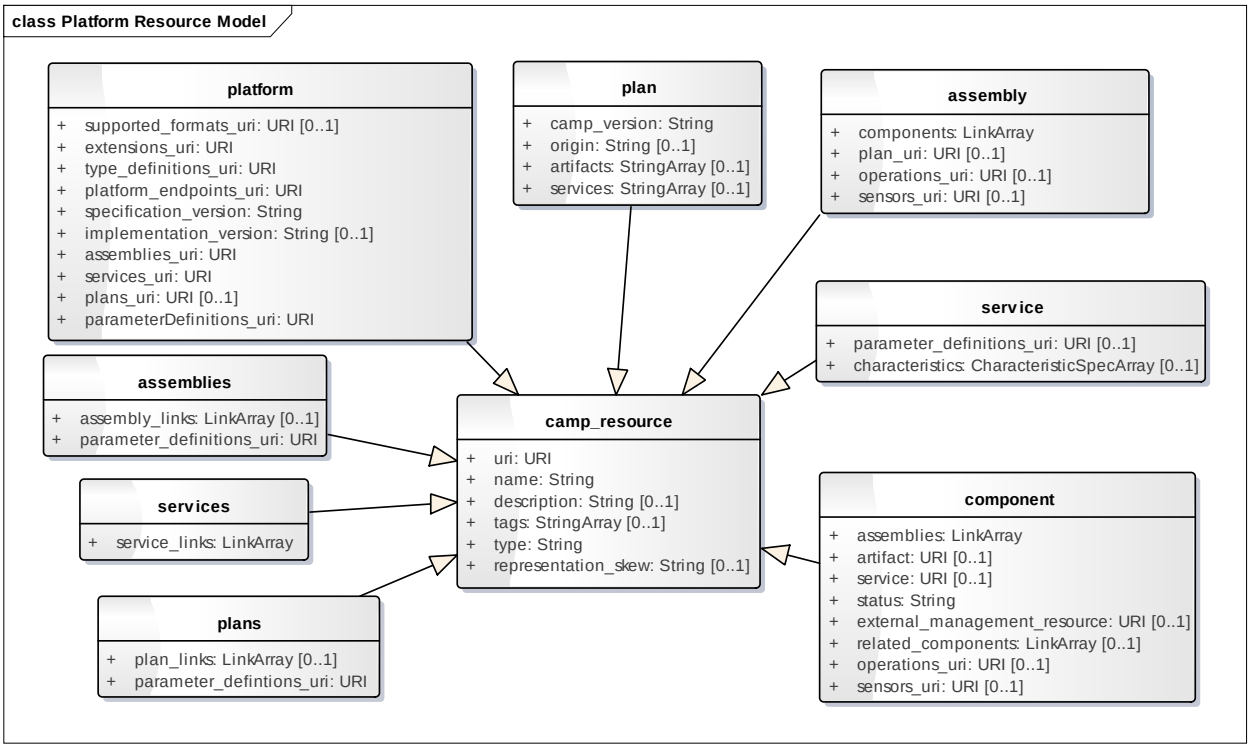

Figure 2-2: CAMP Resources as UML Classes

Figure 2-2 is a UML Class Diagram showing the CAMP resources as UML classes. All CAMP resources share a set of common attributes which they inherit from the *camp_resource* parent class.

Each attribute shown in these UML class diagrams has a CAMP common attribute type. The '+' symbol before each attribute name in the boxes indicates that the attribute access is public (i.e. available through the API). Non-mandatory resource attributes are indicated using the [0..1] UML multiplicity tag.

2.2.1 Platform

The *platform resource* is the primary view of the platform and what is running on it. The *platform resource* references collections of resources that represent the services provided by the platform (as Services), the applications running on this platform (as assembly Resources), as well as collections of metadata resources that describe the resources supported by the platform as well as any extensions that the Provider has implemented. The *platform resource* also determines the scope of access for sharing amongst multiple applications.

2.2.2 Assemblies

An *assembly resource* represents running applications. Operations on an *assembly resource* affect the components and elements of that application.

2.2.3 Components

An *assembly resource* is comprised of one or more *component resources*. A *component resource* represents a discrete and, in most cases, dynamic element of an application such as such as a deployed Ruby gem, a database instance, or a set of entries in a LDAP directory. A *component resource* can be related to other *component resources* through producer/consumer or other kinds of relationships.

2.2.4 Plans

A Plan is meta-data that provides a description of the artifacts that make up an application, the services that are required to execute or utilize those artifacts, and the relationship of the artifacts to those services. Plans can be expressed in two forms; either as a YAML file or, optionally, as a CAMP resource. The Artifacts described in a Plan represent discrete, static elements of an application such as a Ruby gem file, an SQL script, or a PKCS #12 file.

2.2.5 Services

A *service resource* represents a blueprint for creating *component resources* that utilize or embody a platform-provided service in some way. For example, a Service may represent the platform's ability to create a message queue for use by an application.

2.2.6 Operations and Sensors

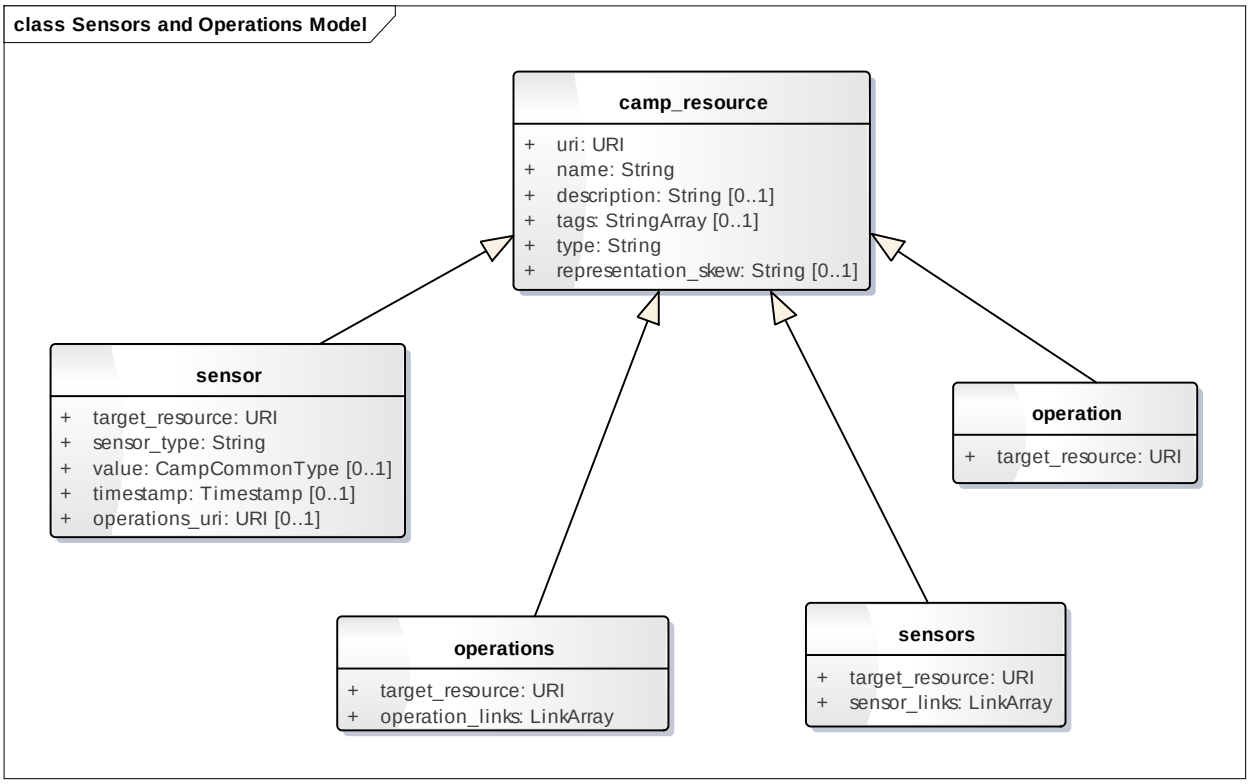

Figure 2-3: Operations and Sensors

Figure 2-3 is a UML class diagram showing the attributes of the *operation resources* and *sensor resources*.

Operations and Sensors provide a way of interacting with an application through the CAMP API. An *operation resource* represents actions that can be taken on a resource, and *sensor resources* represent dynamic data about resources, such as metrics or state. A *sensor resource* is useful for exposing data that changes rapidly, or that might need to be fetched from a secondary system. A *sensor resource* can also offer Operations to allow resetting metrics, or adjusting frequency collection settings.

Multiple *operation resources* and *sensor resources* can be exposed both on *assembly resources* and *component resources*. Operations are also known as effectors. The combination of Operations and Sensors enables ongoing management. This can include automation techniques such as using policies, event-condition-action paradigms, or autonomic control. A Consumer can use the REST API to perform such management. A Provider can also use them. For example, a *component resource* could be offered that allows for "autoscaling" capacity based on the volume of work an application performs.

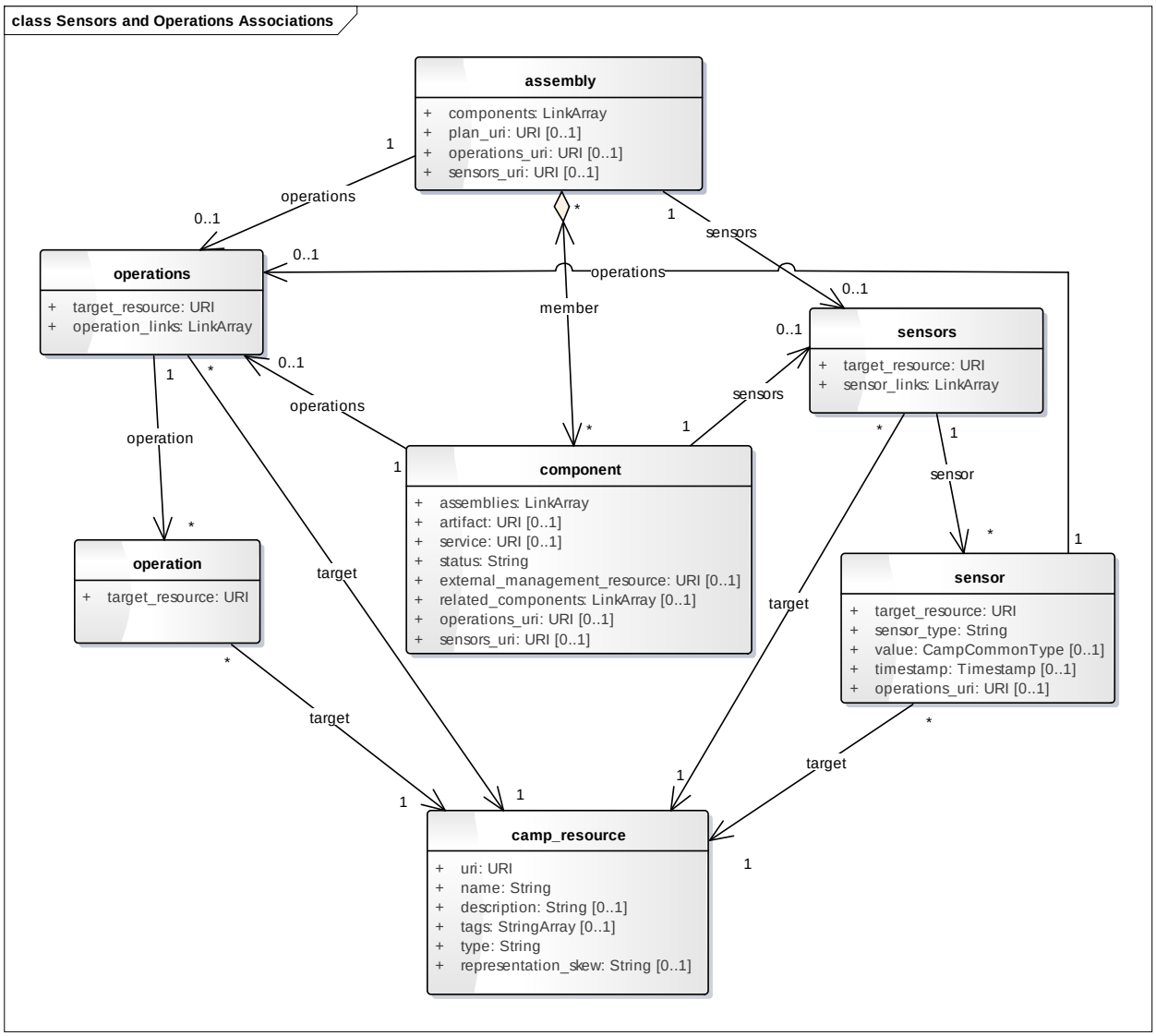

Figure 2-4: Operations and Sensors Associations

Figure 2-4 is a UML class diagram showing *operation resources* and *sensors resources*, and the other CAMP resources that they are associated with.

2.2.7 Resource Relationships

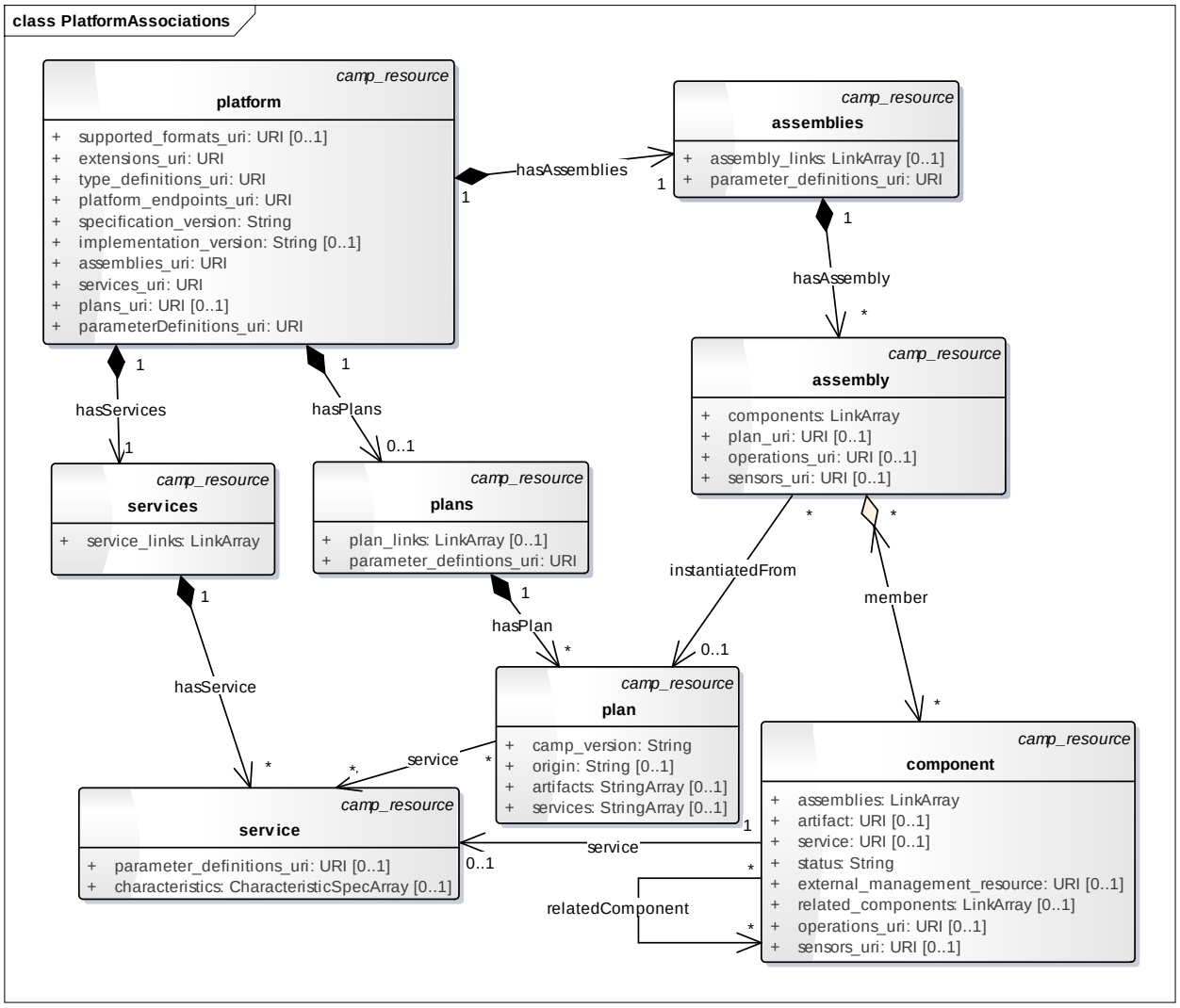

Figure 2-5: Platform Resource Relationships

Figure 2-5 shows the relationships between Platform Resources using a UML class diagram.

Associations which are visible through pointer attributes in resources (i.e. URI, Link, or LinkArray attribute types) are shown using UML named associations with navigation arrows.

Associations which model implementation specific relationships, not visible through the API, are represented using the UML association end notation, without navigation arrows. The '–' symbol on these association ends expresses that access is private (i.e. navigation using resource links is not available through the API).

Strict aggregation (i.e. "has" relationship) is indicated using a solid diamond on the association end attached to the owning resource. This implies that the owned resource cannot exist independent of its owner.

2.3 Deployment

A Platform Deployment Package (PDP) is an archive containing a Plan file together with application content files such as web archives, database schemas, scripts, source code, localization bundles, and icons; and metadata files such as manifests, checksums, signatures, and certificates. It can be used to move an Application and its Components from Platform to Platform, or between an Application Development Environment and a Platform.

In the simplest case (an example of which is provided Section 1.3, “Example”), a PDP or a Plan file can be used to create an *assembly resource* by transmitting an HTTP POST request containing either the PDP or the Plan file to the *assemblies resource*.

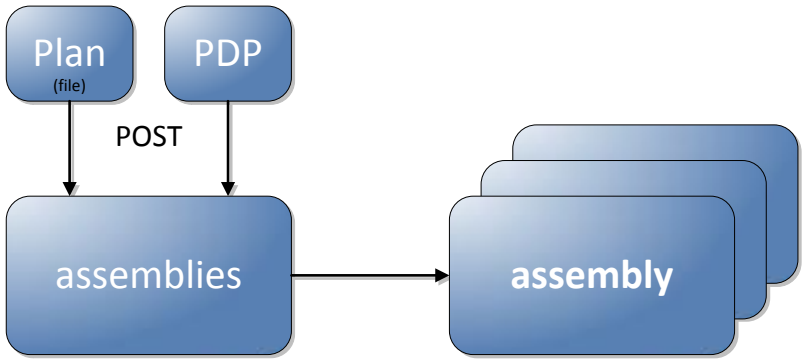

Figure 2-6: Deploying an application

On platforms that choose to support Plans, a CAMP Consumer can create a *plan resource* by uploading either a PDP or a Plan file to the *plans resource* URI using an HTTP POST request. An *assembly resource* can then be created from the *plan resource* by including a reference to the *plan resource* in an HTTP POST request to the *assemblies resource*.

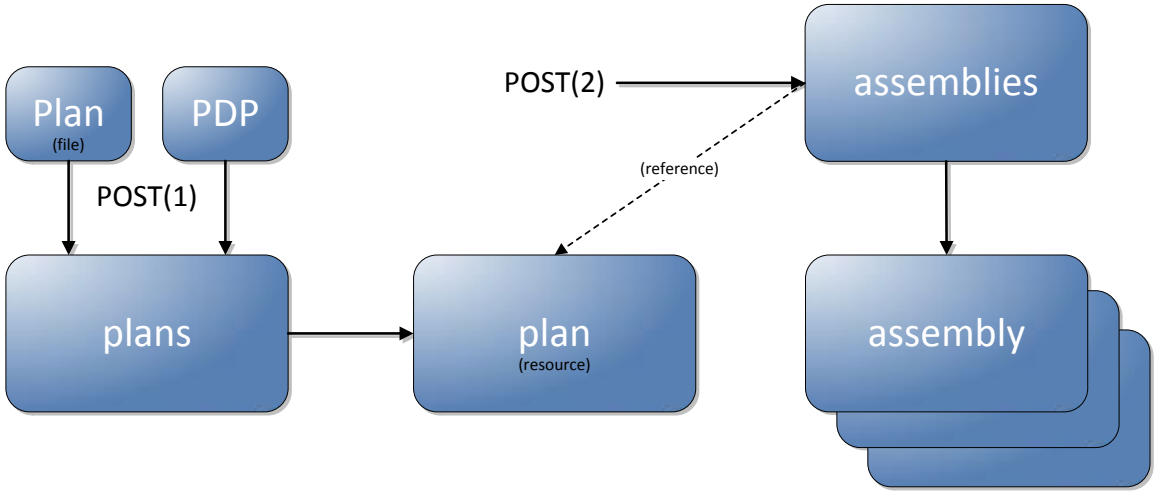

Figure 2-7: Instantiating an application from a plan resource

In Figure 2-7 the POST(1) request creates a *plan resource* by uploading either a PDP or a Plan file to the *plans resource*. The POST(2) request to the *assemblies resource* creates an *assembly resource*. Multiple *assembly resources* can be created from a single *plan resource* by submitting multiple HTTP POST requests.

2.4 Versions and Extensions

This specification supports multiple endpoints and versions, and extensions. All of these are represented in the resource model so they can be discovered by CAMP Consumers. The resources enabling discovery are shown in Figure 2-8, and their relationships are shown in Figure 2-9.

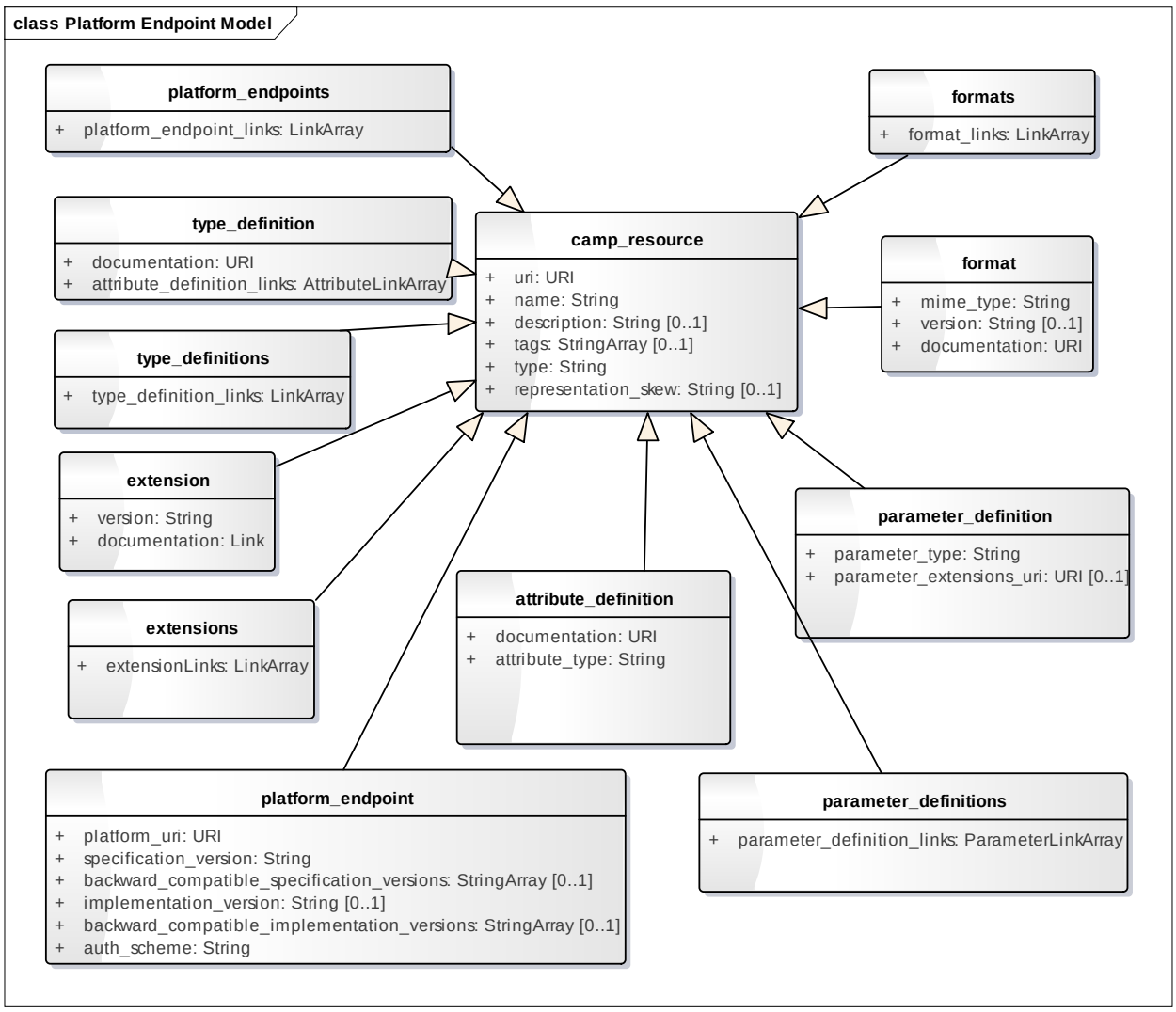

Figure 2-8: Platform Endpoint and Meta Data Resources

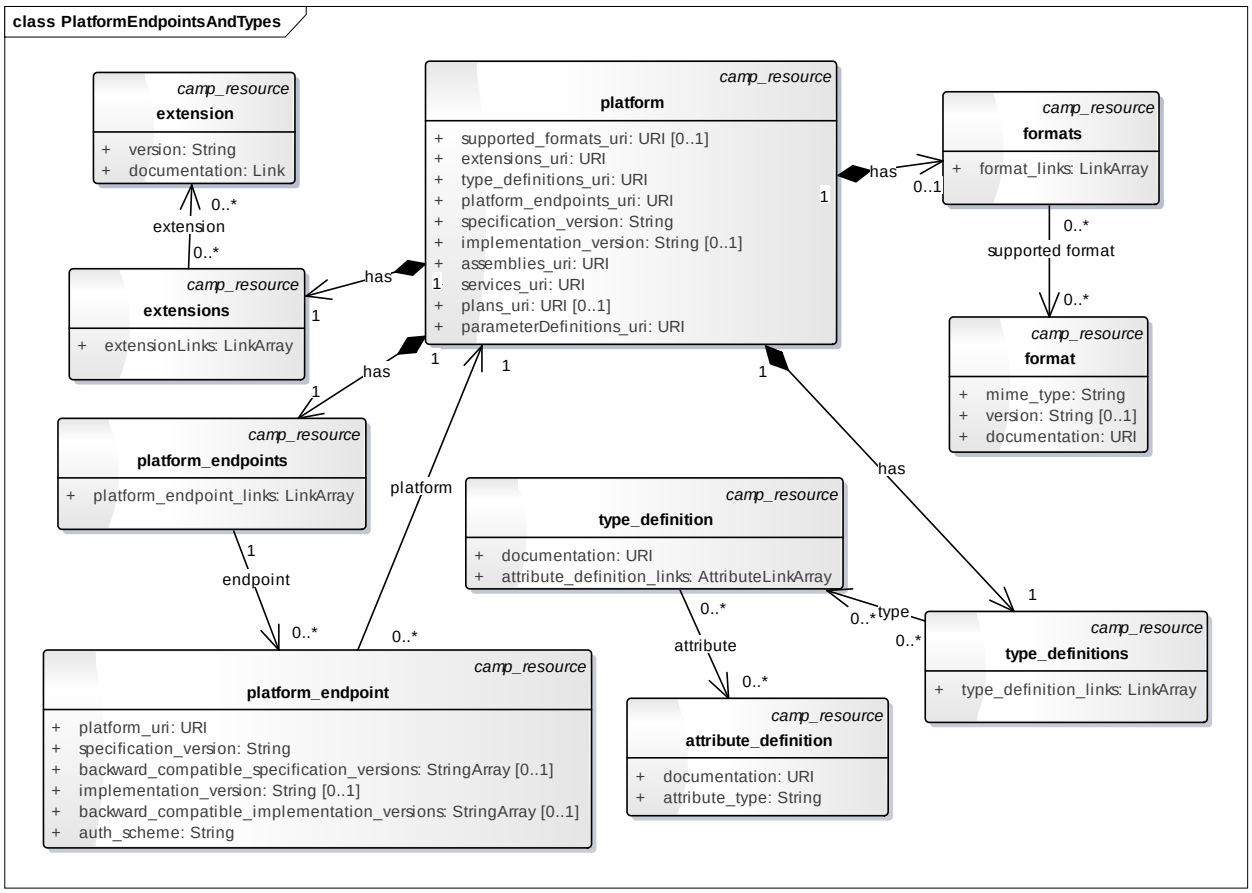

Figure 2-9: Platform Endpoint and Extension Resource Relationships

2.5 Parameters

Parameters can be defined on the *assemblies resource*, *services resource*, and, if supported, *plans resource*. Parameters affect the resources that are generated from these resources. Figure 2-10 illustrates the relationships between these resources and the resources used to represent Parameters.

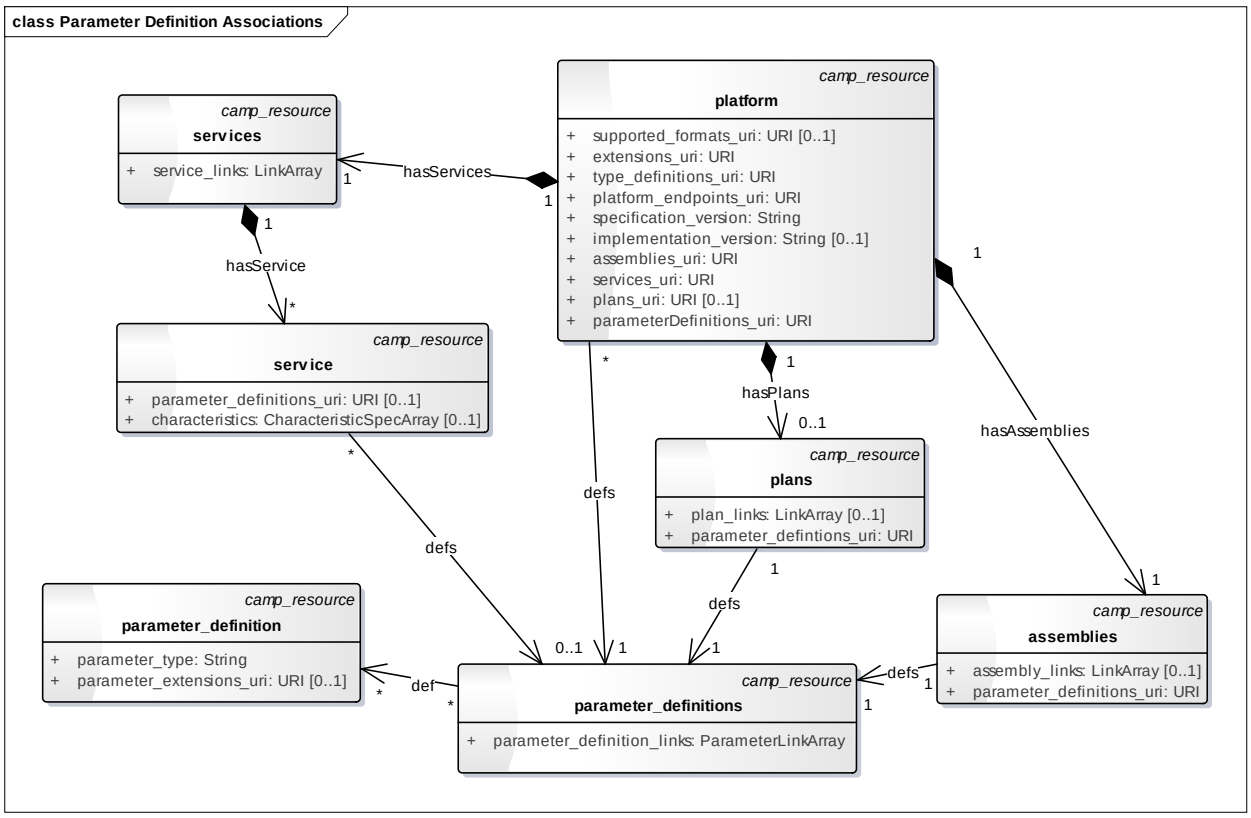

Figure 2-10: Parameter Definition Relationships

2.6 CAMP Common Attribute Types

Many of the attributes in the UML class diagrams have one of the CAMP common attribute types, specified in Section 5.2, "Attribute Types". Figure 2-11 is a UML diagram showing the common data types as UML Data Types, which are used for the Types of these Resource Attribute definitions.

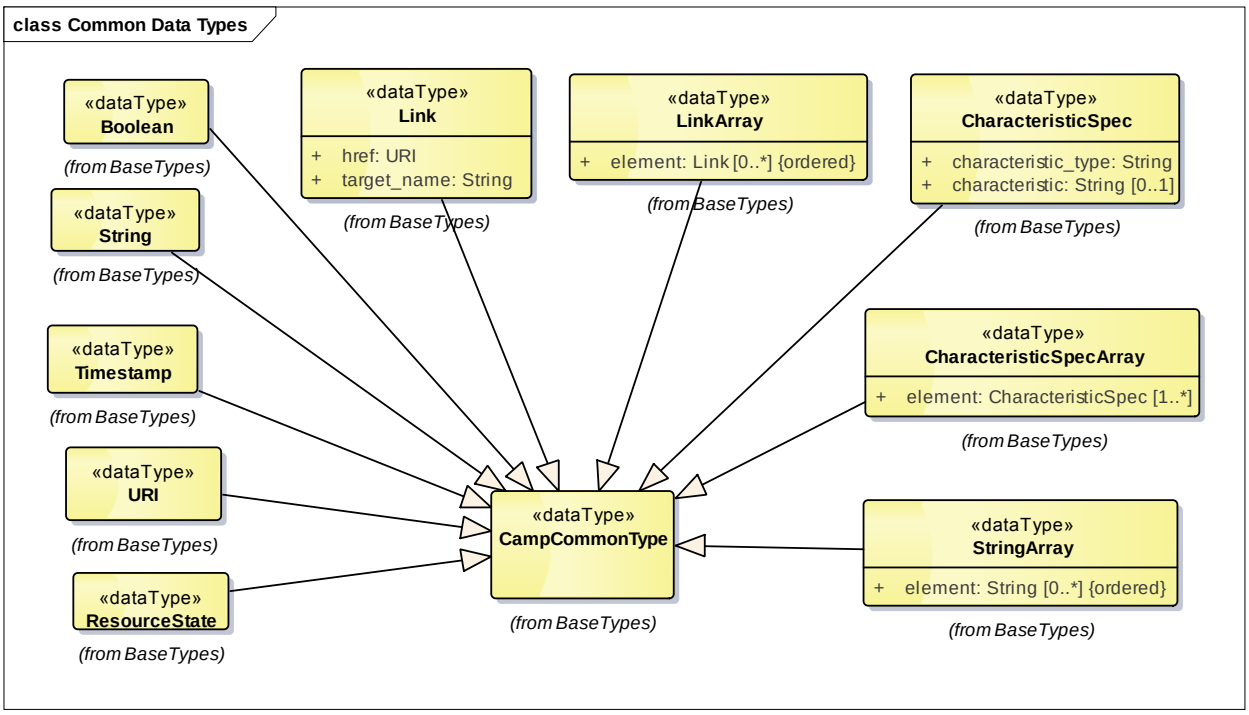

Figure 2-11: CAMP Common Base Types for Resource Attribute Definitions

Multi-valued member attributes are used to model the elements of the LinkArray and StringArray. This is done for modeling purposes only; the attribute name "element" does not appear in the JSON serialization for these common types.

The array types have their elements tagged as ordered.

2.7 Representation Skew

There can be situations in which the information in the resources provided by the CAMP API is not a complete or accurate representation of the state of the underlying implementation. For example, while generating a new instance of an application, a CAMP server might be asked to provide a representation of a Component that corresponds to a dataset that is in the process of being loaded onto a database. While the CAMP server might not be able to provide all of the information about this Component, it would be inaccurate to say that the Component does not exist; it exists but in an intermediate state. It is expected that these sorts of situations will be the exception and that, during the majority of its existence, a CAMP resource will be in synch with the state of its underlying platform implementation.

The significance of this skew is the manner in which it affects the Consumer's interactions with, and expectations about, the resource. In the above example, while the Consumer cannot make any changes to the Component until it has reached a steady state, the Consumer can expect that the resource will reach this state in the near future. There are other situations in which, through some sort of error, the CAMP API cannot tell when or if the information in the resource will ever be synchronized with the underlying implementation.

Details on how this skew is exposed in the CAMP API are provided in Section 5.4.6, "representation_skew".

3 Application Management Lifecycle

This section is informative. The figures in this section are UML object instance diagrams, which represent related Resources at various stages of Platform Resource lifecycle. For simplification, attributes for these resources are not shown. For a comprehensive list of attributes for resources see Section 5, "Resources".

Instances in these diagrams are indicated by boxes, with an underlined "*object-name: Class"* label. Relationships visible through the API are shown using associations with navigation arrows. Implementation specific relationships are indicated using the association end notation, without navigation arrows.

3.1 Initial Platform Resources

The CAMP model includes the resources below when no *assembly resources* or *plan resources* have been created. Note that the support of the *plans resource* and *plan resources* is optional.

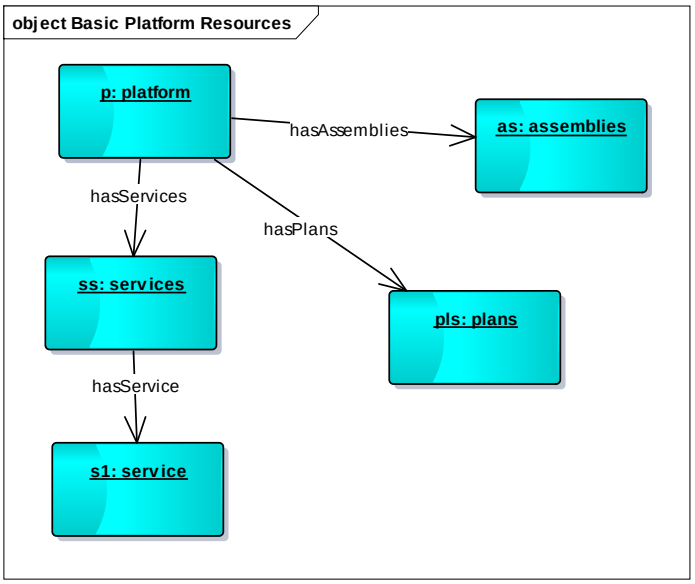

Figure 3-1: Initial Platform Resources

When the Application Administrator first accesses a new account a Platform will have a number of resources visible through the API. The *platform resource* is used to find the other resources in this diagram. The various *service resources* allow for discovery of all the platform services that are available along with value ranges for each service's attributes.

3.2 Creating an Assembly from a PDP or Plan File

A CAMP Consumer can create a new *assembly resource* by uploading either a PDP or a Plan file to the *assemblies resource* URI using an HTTP POST request (see Section 2.3, "Deployment"). The loaded assembly model might then appear as follows (for simplification, the instantiated component resources are not shown in Figure 3-2):

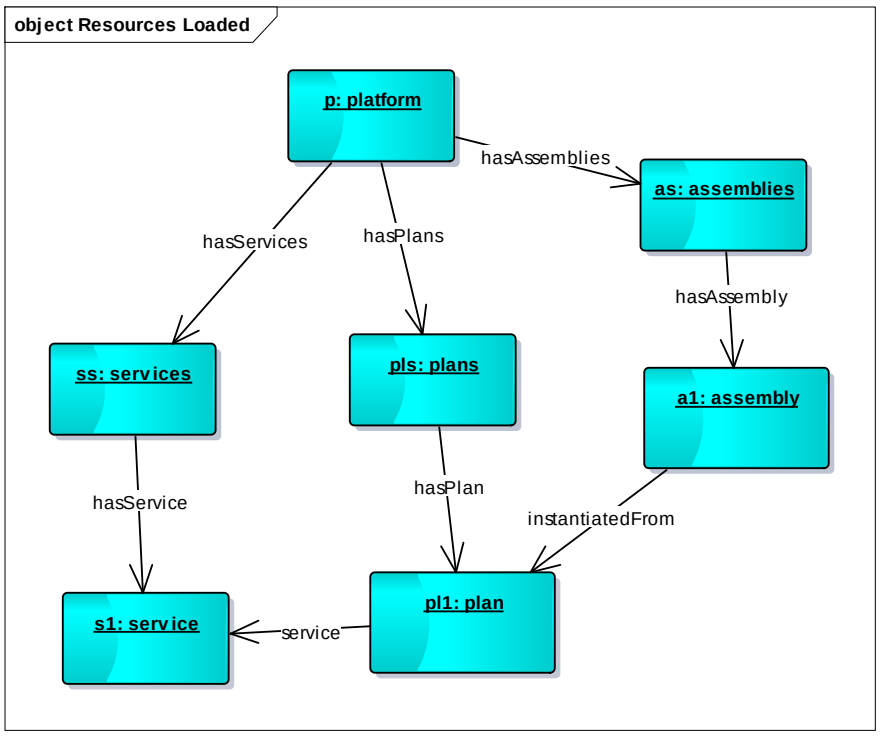

Figure 3-2: Loaded Assembly Resource

If any of its requirements are not resolved, a PDP or Plan file could require modification before it can be used to create *assembly resources*.

3.3 Creating an Assembly from a plan resource

If a Provider supports the *plans resource*, a CAMP Consumer can create a new *plan resource* without creating an *assembly resource* by supplying the contents of, or a reference to, either a PDP or a Plan file to the *plans resource* URI in an HTTP POST request (see Section 6.12, "Registering a Plan"). The loaded *plan resource* model might then appear as follows:

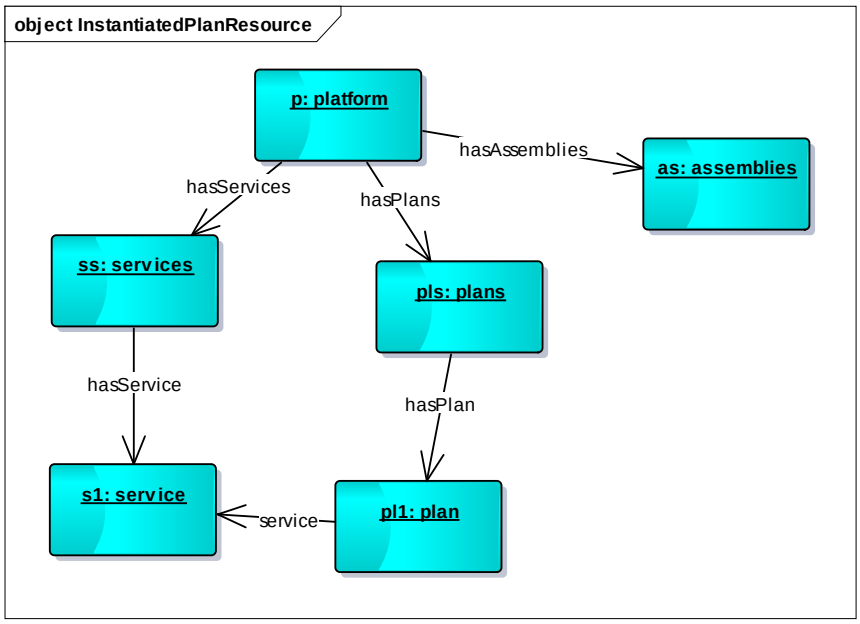

Figure 3-3: Loaded Plan Resource

A CAMP Consumer can create a new *assembly resource* from an existing *plan resource* by providing the reference to that *plan resource* to the *assemblies resource* URI in a HTTP POST request (see Section 6.11.1, “Deploying an Application by Reference”).

Using this two-step process, the loaded *assembly resource* model would appear the same as when using the one-step process, as shown in Figure 3-2.

3.4 Managing an Application Assembly

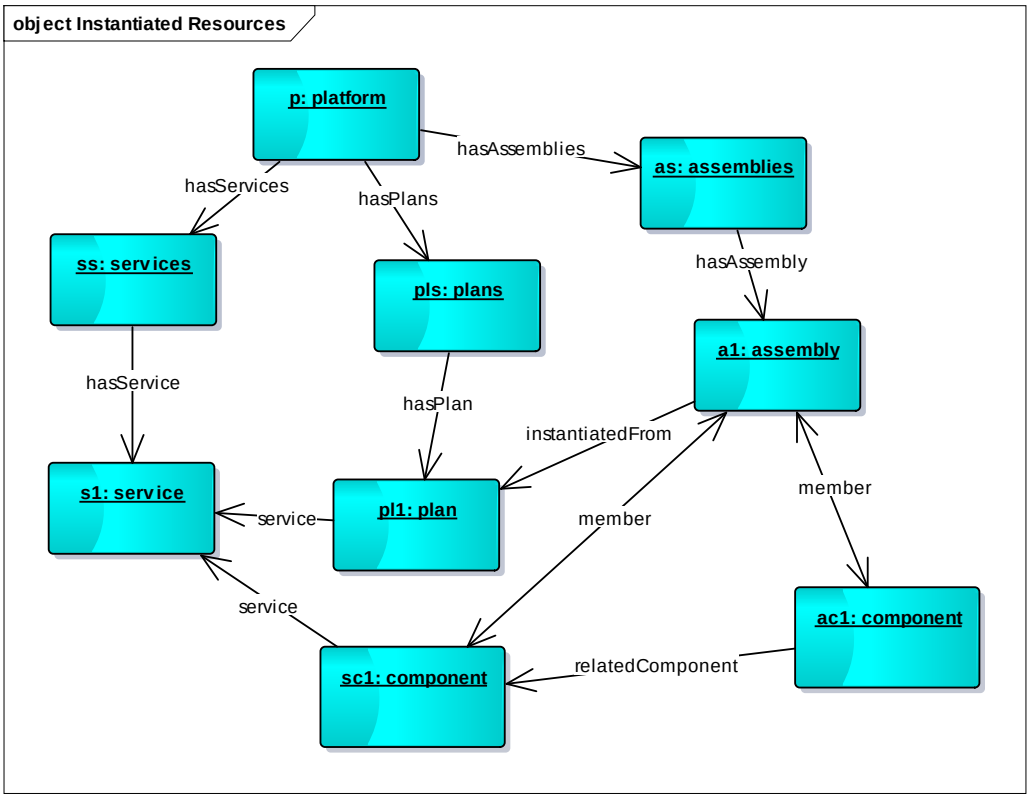

Figure 3-4: Instantiated Resources

To manage the operation of the application, the Application Administrator interacts with the *assembly resource* and the related *component resources*.

The traversal of the resources in the model can be accomplished by following the navigation arrows on the associations in these object instance diagrams, from each resource to the other resources it depends on.

The Application Administrator can observe real-time operational metrics through *sensor resources* on *assembly resources* and *component resources*. In response to these metrics, the Application Administrator — or an automated process such as a management system — can affect changes to those resources through the *operation resources* linked from those same resources.

3.5 Removing Assemblies

When finished working with an application, an Application Administrator can delete an *assembly resource* using a DELETE request. The CAMP platform will typically soon thereafter remove the *assembly resource* and all associated resources which are dedicated to that *assembly*. Where such a resource is not removed immediately, for example, when it is in the process of shutting down, it ought to present a representation skew of DESTROYING in the interim.

When the original *plan resource* is no longer needed, an Application Administrator can delete it using a DELETE request. Again, the CAMP platform will typically delete the *plan resource* and all associated resources which are dedicated to that *plan resource*. Where this deletion is accepted but not immediate,

such as because an *assembly resource* is in use that references the *plan resource*, again the CAMP platform ought to present a representation skew of DESTROYING for the resources being deleted.

4 Platform Deployment Package

The Platform Deployment Package (PDP) ensures portability across platforms. It can be created by a platform to export to another platform, which then imports it. It can also be created by an Application Development Environment running locally or deployed as Software as a Service in the cloud. The PDP (and the Plan file, see Section 4.2, "Plan Overview") defines the formats for on-boarding new applications onto a CAMP-enabled Provider.

4.1 PDP Package Structure

A PDP is an archive which contains a Plan file named `camp.yaml` at the root of the archive. A PDP archive MAY include other files related to the application including, but not limited to, language-specific bundles, resource files, application content files such as web archives, database schemas, scripts, source code, localization bundles, and icons; and metadata files such as manifests, checksums, signatures, and certificates. **[PDP-01]**

4.1.1 Supported Archive Formats

A Provider SHALL support the following archive formats for a PDP:

- A PDP as a ZIP archive [ZIP] **[PDP-02]**
- A PDP as a TAR archive [TAR] **[PDP-03]**
- A PDP as a GZIP [RFC1952] compressed TAR archive **[PDP-04]**

Providers MAY support additional archive formats for the PDP. **[PDP-05]**

4.1.2 Validating Integrity

A PDP MAY contain a manifest file, named camp.mf, at the root of the archive. **[PDP-06]** This file contains SHA256 [SHA256] digests of some or all files in the package. A Provider SHOULD reject a PDP if any digest listed in the manifest does not match the computed digest for that file in the package. **[PDP-07]**

A PDP MAY contain a certificate, named camp.cert, at the root of the archive. **[PDP-08]** This file contains a signed SHA256 digest for the manifest file and the corresponding X.509 certificate. A Provider SHOULD reject any PDP for which the signature verification fails. **[PDP-09]**

The format of the manifest file and the certificate file SHALL be as defined by the OVF specification [OVF]. **[PDP-10]**

4.2 Plan Overview

The Plan provides a description of the artifacts that make up an application, the services that are required to execute or utilize those artifacts, and the relationship of the artifacts to those services. As discussed previously, Plans can be represented in two ways, either as YAML files or as CAMP resources. The examples in this section show Plans as YAML files.

Example 1: Minimal Plan describing an application consisting of a single RPM file

```
00 camp_version: CAMP 1.1
01 artifacts:
02   -
03     artifact_type: org.rpm:RPM
04     content: { href: my-app.rpm }
```

The above example describes an application that consists of a single RPM (RPM Package Manager) package, named "my-app.rpm", which exists at the root of the PDP archive.

4.2.1 Types

Plans can contain descriptions of artifacts, services and their relationships. However, it is outside the scope of this specification to provide detailed definitions of these entities. Instead Plans use 'type' nodes to identify these things. 'Type' nodes are Strings that describe entities that are managed by CAMP, but whose value and semantics are defined outside the CAMP specification. For example, a group of PaaS providers could agree to use the artifact type "org.rpm:RPM" to identify RPM packages. Line 03 in Example 1, above, is an example of the use of such a type.

To promote portability, both providers and consumers of the CAMP API are encouraged to namespace-qualify the types that they use. For example, if a PaaS provider supports a requirement type that expresses the relationship "deploy on a Spring container", the value "com.paas-r-us.spring.DeployOn" is preferable to the value "DeployOn", as the latter is likely to collide with similar types.

In addition to defining the labels for artifacts, services, and their relationships it is expected that those individuals and organizations that define such labels will also define additional attributes that qualify and constrain the entity that is referenced.

Note that this specification does not specify a normative mechanism for Providers to advertise their supported type nodes.

4.2.2 Requirement Specifications

Although Example 1 is a complete, CAMP-conformant Plan, it is somewhat abstract. It essentially says "this application is made up of the following RPM file". It does **not** say anything about what a CAMP Provider is supposed to do with the RPM file. When presented with the Plan in Example 1, a Provider is free to do whatever it likes with the artifact. The obvious action is to install the RPM file (the fact that there is an obvious action is what makes Example 1 workable) but not all artifacts will necessarily have such obvious actions.

Requirement Specifications allow Application Developers to specify what the Provider should do with an artifact.

Example 2: Expanded Plan describing details of how to install the RPM

```
00 camp_version: CAMP 1.1
01 artifacts:
02   -
03     artifact_type: org.rpm:RPM
04     content: { my-app.rpm }
05     requirements:
06       -
07         requirement_type: org.rpm:Install
08         org.rpm.installopts.excludedocs: true
```

Example 2 adds a Requirement Specification (lines 07-08) to indicate (through the `requirement_type` value of "org.rpm:Install" on line 07) that target Providers are to install the RPM. Furthermore it indicates, on line 08, something about how the Provider should install the RPM (i.e. in a way that excludes documentation).

The correct processing of this requirement is predicated upon the Providers understanding of the structure and semantics of the "org.rpm:Install" type. The Provider has to know that this type indicates that the parent RPM artifact is to be installed on a Linux instance and that Requirement Specifications of this type may contain a `org.rpm.installopts.excludedoc` node whose value is a boolean that indicates whether files marked as documentation should be installed. It is assumed that the semantics associated with "org.rpm:Install" requirement type are documented, and that this documentation also describes the value space and semantics of the `org.rpm.installopts.excludedocs` node.

4.2.3 Service Specifications

Example 2 is more specific than Example 1, but it is still silent about what kind of Linux instance we want the RPM installed on. When presented with the Plan in Example 2, a Provider is free to install the RPM on any kind of Linux instance or even, hypothetically, a non-Linux operating system that supports RPM.

Service Specifications allow Application Developers to constrain or outline the services that can be used to support the requirements they have specified.

Example 3: Expanded Plan that includes a Service Specification

```
00 camp_version: CAMP 1.1
01 artifacts:
02   -
03     artifact_type: org.rpm:RPM
04     content: { my-app.rpm }
05     requirements:
06       -
07         requirement_type: org.rpm:Install
08         org.rpm.installopts.excludedocs: true
09         fulfillment:
10           characteristics:
11             -
12               characteristic_type: com.example:Linux
13               com.example.linux.kernelVersion: [3.9.6]
14               org.iaas.bitsize: 64
```

Example 3 adds a Service Specification to the Requirement Specification from Example 2. This Service Specification indicates that, wherever the Provider decides to install the RPM, it has to be a 64-bit Linux instance running a kernel of version 3.9.6.

As in the case of our Requirement Specification, the correct processing of this Plan depends upon the Providers ability to understand Characteristic Specifications of the type “com.example:Linux”. The Provider needs to know that characteristics of this type can contain, among other possible nodes, the `com.example.linux.kernelVersion` and `org.iaas.bitsize` nodes as well as understand the allowed values and the semantics of these nodes.

Although Example 3 is fairly specific (“this application is made up of an RPM file which is to be installed, excluding documentation, on a 64-bit Linux instance running a kernel of version 3.9.6”), it is still somewhat abstract in that it does not reference the specific Linux instance that the RPM is to be installed on. If an Application Developer wishes to explicitly identify the exact Linux instance on which their RPM is to be installed, they can do so using the `href` node of a Service Specification.

Note: See Appendix D for suggested version range values to use when no prevailing scheme already exists for the type.

Example 4: Concrete Plan with service resource reference

```
00 camp_version: CAMP 1.1
01 artifacts:
02   -
03     artifact_type: org.rpm:RPM
04     content: { my-app.rpm }
05     requirements:
06       -
07         requirement_type: org.rpm:Install
08         org.rpm.installopts.excludedocs: true
09         fulfillment:
10           href: http://example.org/my_paas/services/8675309
```

Example 4 amends the Service Specification introduced in Example 3 to reference an instance of a *service resource* provided by the CAMP implementation at “example.org”. Since the Application Developer is calling for the use of a specific service, it is no longer necessary to indicate to the Provider the general characteristics of the services that are suitable for fulfilling the “org.rpm:Install” requirement. Therefore, the Characteristic Specification in Example 3 has been removed

Note that the specificity in Example 4 comes at the expense of portability. Due to the reference to a specific resource, it is doubtful that the Plan in Example 4 could be successfully deployed on any CAMP instance other than the one at “example.org” whereas the Plan in Example 3 can be deployed on any CAMP instance that supports the “org.rpm:RPM” artifact type, the “org.rpm:Install” requirement type, and the “com.example:Linux” characteristic type. This tradeoff between specificity and portability is a design

feature of CAMP Plans. There are cases (e.g. when developing an application) in which it makes sense to target a Plan for a specific CAMP instance and there are cases (e.g. when moving an application to a different provider) when it makes sense to make the Plan as generic as possible. Plans are designed to allow the Application Developer to be as specific or generic as necessary to accomplish their particular task.

4.2.3.1 Shared Services

There are situations in which an application can have two or more artifacts that need to share the same runtime instance of a service.

Example 5: Plan with shared Service Specification

```
00 camp_version: CAMP 1.1
01 artifacts:
02   -
03     artifact_type: com.java:WAR
04     content: { href: vitaminder.war }
05     requirements:
06       -
07         requirement_type: com.java:HostOn
08         com.java.servlet.contextName: "/vitaM"
09         fulfillment:
10           …
11       -
12         requirement_type: com.java.jdbc:ConnectTo
13         fulfillment: id:db
14   -
15     artifact_type: org.sql:SqlScript
16     content: { href: vitaminder.sql }
17     requirements:
18       -
19         requirement_type: org.sql:ExecuteAt
20         fulfillment: id:db
21 services:
22   -
23     id: db
24     characteristics:
25       -
26         characteristic_type: org.storage.db:RDBM
27         …
28       -
29         characteristic_type: org.storage.db:Replication
30         …
31       -
32         characteristic_type: org.iso.sql:SQL
```

The above example describes an application with two components, a WAR file and an SQL script. In the case of this particular application, the SQL script is used to initialize the database that will be used by the WAR file. The components created from the two artifacts need to share a common database instance or the application will not work. Lines 23-33 describe the shared target database service. Line 23 is an ‘id’ node with the value ‘db’. This node is used as the target for the ‘fulfillment’ nodes on lines 13 and 20. The common use of the “id:db” value in lines 13 and 20 indicates that, whatever service used to satisfy the Service Specification in lines 23-33, it will be shared by the components that are created by resolving the requirements on lines 12-13 and lines 19-20.

4.2.3.2 Service Frameworks

There are situations in which the artifacts of an application are dynamically added (e.g. via a git [Git] push operation) after the creation of a “service framework” on which these artifacts can be deployed. Such a framework can be specified via a Plan that contains Service Specifications but no Artifact Specifications.

Example 6: Plan with only Services Specifications

```
camp_version: CAMP 1.1
services:
  -
    name: Rails Runtime
    characteristics:
      -
        characteristic_type: org.ruby-lang:Ruby
        …
      -
        characteristic_type: org.rubyonrails:Rails
        …
  -
    name: Database
    characteristics:
      -
        characteristic_type: org.storage.db:RDBM
        …
  -
    name: Git Repo
    characteristics:
      -
        characteristic_type: com.git-scm:GIT
        …
```

The above example specifies a set of services onto which the user can deploy Rails components by pushing them to the git repository that will be created as a result of deploying this Plan.

4.2.4 Names, Description, and Tags

Plans, artifacts and services can be decorated with names, descriptions, and tags. CAMP implementations can use this information when creating the resources that correspond to these entities. For example, the following Plan file:

Example 7: Plan with names, descriptions, and tags

```
name: Mike's Drupal Instance
description: Drupal 6.28
tags: [ PHP, Drupal6, mikez ]
camp_version: CAMP 1.1
artifacts:
  -
    artifact_type: net.php:Module
    content:
      href: ftp://ftp.drupal.org/files/projects/drupal-6.28.tar.gz
…
```

when successfully registered, could result in the creation of the following *plan resource*:

```
{
  "type": "plan",
  "uri": "http://uswest.paas-r-us.com/camp/plan/101",
  "name": "Mike's Drupal Instance",
  "description": "Drupal 6.28",
  "tags": [ "PHP", "Drupal6", "mikez" ],
  …
}
```

4.3 Plan Schema

A Platform Deployment Package (PDP) SHALL contain a single Plan file. **[PDP-11]** The Plan file SHALL be located at the root of the PDP archive. **[PLAN-01]** The Plan file SHALL be named "camp.yaml".

[PLAN-02] The Plan file SHALL conform to YAML 1.1 [YAML 1.1]. **[PLAN-08]** The Plan file SHALL conform to the description provided in this section. **[PLAN-09]**

4.3.1 General Nodes

Plans, Artifact Specifications, and Service Specifications can contain the following nodes:

4.3.1.1 name

Type: String

Required: false

This node expresses the human-readable name of the Plan or Specification. Providers MAY reflect the value of this attribute in the names of any resources that are created in the processing the Plan. **[PDP-14]**

4.3.1.2 description

Type: String

Required: false

This node expresses the human-readable description of the Plan or Specification. Providers MAY reflect the value of this attribute in the descriptions of the resources that are in the processing the Plan. **[PDP-15]**

4.3.1.3 tags

Type: String[]

Required: false

This node expresses an array of human-readable tags for the Plan or Specification. Providers MAY reflect the values of this attribute in the tags of the resources that are created in the processing of the Plan. **[PDP-16]**

4.3.2 Plan

A Plan defines the structure the elements in a Plan file or resource. A Plan file SHALL contain a single instance of a Plan. **[PLAN-03]** A Plan has the following, general representation:

```
name: String ?
description: String ?
tags: String[] ?
camp_version: String
origin: String ?
artifacts: ArtifactSpecification[] ?
services: ServiceSpecification[] ?
```

In addition to the general nodes, a Plan contains the following nodes:

4.3.2.1 camp_version

Type: String

Required: true

The value of this node expresses the version of the CAMP specification to which the Plan conforms. For Plans that conform to this document, the value of this node SHALL be as defined in Section 1.8 "Specification Version". **[PLAN-05]**

4.3.2.2 origin

Type: String

Required: false

The value of this node specifies the origin of the Plan. For example, when exporting a *plan resource* into a PDP, a Provider might use the URL of its *platform resource* for this value. Alternatively, an Application Development Environment could use its name and version.

4.3.2.3 artifacts

Type: ArtifactSpecification[]

Required: false

This node lists the artifacts that comprise the application described by the Plan. For portability reasons, Providers are cautioned against regarding the order of the elements in this array as significant.

4.3.2.4 services

Type: ServiceSpecification[]

Required: false

This node describes the services that the application described by the Plan requires in order to function. For portability reasons, Providers are cautioned against regarding the order of the elements in this array as significant.

4.3.3 ArtifactSpecification

An ArtifactSpecification describes an artifact of the application. The artifact MAY be contained within the PDP or MAY exist in some other location. **[PDP-22]**

An ArtifactSpecification has the following, general representation:

```
name: String ?
description: String ?
tags: String[] ?
artifact_type: String
content: ContentSpecification
requirements: RequirementSpecification[] ?
```

In addition to the general nodes, an ArtifactSpecification contains the following nodes:

4.3.3.1 artifact_type

Type: String

Required: true

The value of an artifact_type node specifies the type of an artifact.

Note: Values for an artifact_type node are not defined by this specification. See Section 4.2.1, "Types".

4.3.3.2 content

Type: ContentSpecification

Required: true

This node identifies the location of the content of the artifact described by this Artifact Specification. See Section 4.3.4, "ContentSpecification", for details.

4.3.3.3 requirements

Type: RequirementSpecification[]

Required: false

This array specifies the ways in which the artifact described by this Artifact Specification engages with the services provided by the platform. See Section 4.3.5, "RequirementSpecification", for details. For portability reasons, Providers are cautioned against regarding the order of the elements in this array as significant.

4.3.4 ContentSpecification

A ContentSpecification defines the content of a component. A ContentSpecification has one of two, mutually exclusive, nodes: `href` or `data`. It has the following, general representation:

```
href: URI
```

or

```
data: String
```

When `href` is used in a ContentSpecification its value is interpreted as follows:

- Providers SHALL support the "https" URI scheme as defined in RFC 2818 [RFC2818]. **[PDP-27]** A Provider MAY support additional URI schemes listed at http://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml. **[PDP-28]**
- URL's with the special scheme "pdp:" are interpreted as files contained in the PDP.
    - If the path segment (after the "pdp:") begins with a "/" it is an absolute path.
    - If the path segment *is* "!" (i.e. the URL is "pdp:!"), the reference is to the PDP archive itself. This is useful in making an existing deployment package (such as a WAR) function as a PDP.
    - For any other path segment, the path is relative to the location of the file which contains the Content Specification, subject to the guidelines below.
    - Where the path segment contains the special character "!", it is treated as a delimiter to look for the path to the right of "!" inside the archive at the path to the left of the "!". Providers SHALL understand this delimiter and SHALL NOT resolve any content if the archive format is unsupported. **[PDP-29]** Consumers SHALL follow the syntax and semantics described here when using URIs with a "pdp" scheme. **[PLAN-07]** For example "pdp:/certs.zip!/id_rsa.pub" refers to a file "id_rsa.pub" contained at the root of a "certs.zip" file located at the root of the PDP, and is valid only on platforms which support the ZIP format in conjunction with "!". On other platforms the link will not be resolved.
- Where the value is not a URI, it is interpreted as a "pdp:" protocol link, as though it were preceded by "pdp:/".

Example 8: A Plan describing an application consisting of the contents of the PDP

```
00 artifacts:
01   -
02     type: org.oasis-open.tosca:CSAR
03     content: { href: pdp:! }
04     requirements:
05       -
06         type: com.oasis-open.tosca:DeployOn
```

The above example illustrates the use of the "pdp:!" construct wherein the content being referenced (on line 03) is the PDP itself. In this case the PDP is also an OASIS TOSCA v1 Cloud Service Archive.

4.3.5 RequirementSpecification

A RequirementSpecification describes the relationship between an artifact and a platform. Providers are expected to use the information in a RequirementSpecification to determine what actions to perform on/with the artifact described in the containing Artifact Specification.

A RequirementSpecification has the following, general representation:

```
requirement_type: String
fulfillment: (String | ServiceSpecification) ?
```

4.3.5.1 requirement_type

Type: String

Required: true

The value of this node defines the relationship of the artifact that contains this RequirementSpecification to a service. For example, “com.java:HostOn”. See Section 4.2.1, “Types”, for a general description of the definition and treatment of these values.

It is expected that RequirementSpecifications will contain extension nodes that modify or provide additional information about the relationship that they describe. The value space and semantics of these extensions ought to be part of the definition of the value used in the “type” node. For example, the definition of the “com.java:HostOn” relationship might define a “com.java:contextPath” node whose value specifies the desired context path for the artifact when it is deployed on its selected service.

4.3.5.2 fulfillment

Type: String or ServiceSpecification

Required: false

The value of this node either describes, or references a description of, the other party in the relationship (i.e. the service) defined by this RequirementSpecification. In the case where this node references a description, the value is a String that corresponds to the `id` node of a ServiceSpecification (e.g. “id:db”). In the case where this node contains the description, the value is a ServiceSpecification. See Section **Error! Reference source not found.**, “ServiceSpecification”, for details.

4.3.6 ServiceSpecification

A ServiceSpecification describes a service used by the application. This description is not intended to be a complete list of every detail of the service but, instead, an enumeration of those facets that, for whatever reason, are important to the application described by the Plan. Providers are expected to use the information in a ServiceSpecification to select an appropriate service for resolving the containing RequirementSpecification.

A ServiceSpecification has the following, general representation:

```
name: String ?
description: String ?
tags: String[] ?
id: String ?
href: URI ?
characteristics: CharacteristicSpecification[] ?
```

In addition to the general nodes, a ServiceSpecification contains the following nodes:

4.3.6.1 id

Type: String

Required: false

The value of this node serves as an anchor for intra-Plan references. See Section 4.3.5.2, “fulfillment”, for information on how this anchor is used. Plans SHALL use id values that are unique within the scope of the Plan. **[PLAN-06]**

4.3.6.2 href

Type: URI

Required: false

The value of this node is a reference to a *service resource* (see Section 5.13, “service Resource”) that resolves the service described by this ServiceSpecification. If a Consumer includes this node in a Plan, the value of this node SHALL reference a Consumer-visible resource within the target Platform. **[RMR-01]**

4.3.6.3 characteristics

Type: CharacteristicSpecification[]

Required: true

This array provides the characteristics of the service described by this ServiceSpecification. See Section 4.3.6, "CharacteristicSpecification", for details. For portability reasons, Providers are cautioned against regarding the order of the elements in this array as significant.

4.3.7 CharacteristicSpecification

A CharacteristicSpecification describes a desired characteristic or capability of a service. It has the following, general representation.

```
characteristic_type: String
String: String *
```

The inclusion of a CharacteristicSpecification in a ServiceSpecification indicates that the characteristics being described are significant to the application, but the degree of this significance (e.g. "absolutely necessary" versus "would be nice to have") is not indicated.

4.3.7.1 characteristic_type

Type: String

Required: true

The value of this node defines the characteristic being described by this CharacteristicSpecification. For example, "com.java:ServletContainer". See Section 4.2.1, "Types", for a general description of the definition and treatment of these values.

It is expected that CharacteristicSpecifications will contain extension nodes that modify or provide additional information about the characteristic that they describe. The value space and semantics of these extensions ought to be part of the definition of the value used in the `characteristic_type` node. For example, the definition of the "org.rubyonrails:Rails" characteristic might define a "org.rubyonrails:version" node whose value specifies the version of Rails provided by the service.

Note: See Appendix D for suggested version range values to use when no prevailing scheme already exists for the type.

5 Resources

The following sub-sections describe the resources defined by this specification.

When supporting such a Resource, a Provider SHALL implement it and serialize it as described in the corresponding sub-section. **[RE-70]**

A Consumer SHALL serialize Resource data in its requests based on the definition of this Resource as described in the corresponding sub-section. **[RE-71]**

5.1 Attribute Constraints

Resource attributes are constrained along a number of axes. These are:

5.1.1 Required

If the Required boolean constraint for an attribute of a resource type has a value of "true", then a resource of this type SHALL have the attribute present. **[RE-06]** If the value is "false" then the resource is valid with or without the attribute present.

5.1.2 Mutable

This boolean indicates the mutability of the attribute's value(s). "false" indicates that the value of the attribute, once set, SHALL NOT change for the lifetime of the resource. **[RE-07]** "true" indicates that the value of the attribute MAY change due to the actions or activity of either the provider or the Consumer. **[RE-08]**

5.1.3 Consumer-mutable

This boolean indicates the ability of a consumer to set the value of the attribute. It is only relevant for mutable attributes. "false" indicates that the value(s) of the attribute SHALL NOT be changed by Consumers. **[RE-09]** A value of "true" indicates that Consumers MAY change the value of the attribute. **[RE-10]** Note that a value of "true" does not preclude the Provider from changing the value of the attribute.

5.2 Attribute Types

Resource attributes are defined using the following types:

5.2.1 Boolean

As defined by JSON [RFC4627], a token having a literal value of either `true` or `false`. The use of this type is indicated in metadata by an *attribute_definition resource* with an `attribute_type` value of "Boolean".

5.2.2 String

A UNICODE string as defined by JSON [RFC4627]. The use of this type is indicated in metadata by an *attribute_definition resource* with an `attribute_type` value of "String".

5.2.3 URI

A String (see above) that conforms to the syntax defined in RFC 3986 [RFC3986]. The use of this type is indicated in metadata by an *attribute_definition resource* with an `attribute_type` value of "URI".

5.2.4 Timestamp

A String (see above) that conforms to the syntax defined in ISO 8601 [ISO 8601:2004]. Consumers and Providers SHALL express Timestamps in UTC (Coordinated Universal Time), with the special UTC designator ("Z"). **[RE-65]** The use of this type is indicated in metadata by an *attribute_definition resource* with an `attribute_type` value of “Timestamp”.

5.2.5 Link

The management model defined in this specification involves resource entity attribute values that link to other resource entities. The “Link” type defined here is used for such attribute values.

```
{
  "href": URI,
  "target_name": String
}
```

The use of this type is indicated in metadata by an *attribute_definition resource* with an `attribute_type` value of “Link”.

5.2.5.1 href

Type: URI

Required: true

Mutable: false

This attribute is the URI [RFC3986] of the resource referenced by this Link.

5.2.5.2 target_name

Type: String

Required: true

Mutable: true

Consumer-mutable: false

This attribute echoes the value of the `name` attribute of the resource referenced by this Link. The value of this attribute may be changed by the Platform.

5.3 CAMP Resource Type Inheritance

Each CAMP resource has a resource type associated with it. This is specified by the attribute named `type` as defined in Section 5.4.5, “type”. The resource type defines the attributes for that resource along with the constraints and semantics of those attributes. Resource types form an inheritance hierarchy with *camp_resource* (See Section 5.4, “camp_resource Resource”) at its root. When a resource type (sub-type) inherits from another resource type (super-type), the sub-type inherits, and therefore includes, all the super-type’s attributes along with its constraints and semantics. A sub-type can add additional attributes not present in its super-type(s). A sub-type MAY further restrict the constraints of an attribute inherited from its super-type(s). **[MO-01]** A sub-type SHALL NOT loosen the constraints of an attribute inherited from its super-type(s). **[MO-02]** As a consequence, a resource of a super-type can always be substituted with a resource of any of its sub-types. A resource type MAY inherit from more than one super-type. **[MO-03]** If there is an attribute name collision when a sub-type inherits from multiple super-types, the inherited attributes of the same name SHALL NOT contradict the constraints and semantics of the attributes defined in its super-types. **[MO-04]**

5.4 camp_resource Resource

All CAMP resources SHALL inherit directly or indirectly from this resource. **[MO-05]** This resource contains the following attributes:

5.4.1 uri

Type: URI

Required: true

Mutable: false

This attribute expresses the URI of the resource.

5.4.2 name

Type: String

Required: true

Mutable: true

Consumer-mutable: true

This attribute expresses the human-readable name of the resource.

5.4.3 description

Type: String

Required: false

Mutable: true

Consumer-mutable: true

This attribute expresses the human-readable description of the resource.

5.4.4 tags

Type: String[]

Required: false

Mutable: true

Consumer-mutable: true

This attribute is an array of String values that may be assigned by the provider or the user. These values can be used for keywording and terms-of-interest.

5.4.5 type

Type: String

Required: true

Mutable: false

This attribute expresses the CAMP resource type. Every CAMP resource type defined in this specification specifies the required value for this attribute.

5.4.6 representation_skew

Type: String

Required: false

Mutable: true

Consumer-mutable: false

The `representation_skew` attribute expresses the relationship between the information presented in the resource and the status of the platform implementation artifacts that are represented by that resource (see Section 2.7, "Representation Skew"). It is an optional, enumerated String. If present, `representation_skew` SHALL have one of the following values: **[RE-11]**

- “CREATING” – describes a resource that is in the process of being created. The client can expect that the resource will have a skew of “NONE” once this process has completed.
- “NONE” – is an assertion by the CAMP server that the information in the resource is an accurate representation of the underlying platform implementation. Absent some action by the client or some other event (e.g. platform shutdown), a resource with a skew of NONE can be expected to remain in synch with the platform implementation.
- “UNKNOWN” – indicates that the CAMP server cannot accurately depict the aspect of the platform implementation represented by this resource. Users can attempt to address the underlying issues(s) by manipulating this and/or other resources as specified by the API.
- “DESTROYING” – describes a resource that is in the process of being destroyed. The client can expect that the resource will cease to exist once this process has completed.

The absence of the `representationSkew` attribute is semantically equivalent to a value of “NONE”.

The value of the `representation_skew` attribute applies only to the resource of which it is part. The skew of any resources that are contained (via Link relationships) by another resource (e.g. in the manner in which the *assembly resource* contains *component resources*) is conveyed by the individual `representation_skew` of those sub-resources and not aggregated or “rolled up” into the containing resource.

The value of the `representation_skew` attribute affects the availability of the HTTP methods for that resource. For example, resources with a `representation_skew` value of CREATING might support the GET, HEAD and DELETE methods, but no other HTTP methods. The following table lists the methods that SHALL be supported for each representation_skew value. **[RE-12]**

representation_skew value	Methods Available
CREATING	GET, DELETE
NONE	All supported methods for that resource.
UNKNOWN	All supported methods for that resource.
DESTROYING	GET

Table 5-1: representation_skew Available Methods

For each representation_skew value, CAMP Providers MAY support HTTP methods in addition to those listed in the corresponding row of Table 5-1. **[RE-13]**

5.5 HTTP Method Support

As described in Section 6.1, “Transfer Protocol”, Consumers use HTTP [RFC2616] to interact with CAMP-defined resources. To foster interoperability it is necessary to define the HTTP methods supported by each resource. Note that a requirement on the Provider to support a particular HTTP method on a resource does not ensure that all requests to that resource using that method will succeed; it simply guarantees that the Provider will not fail such requests with a 405 (Method Not Allowed) error.

Providers SHALL support the HTTP GET, PUT, and PATCH methods on all of the resources defined in this section. **[RE-53]** Requirements for the support of additional HTTP methods are outlined in the descriptions of each resource below. Providers MAY elect to support additional HTTP methods in addition to those described here. **[RE-54]**

5.6 platform_endpoints Resource

A Provider MAY concurrently offer multiple instances of the CAMP API. **[RE-15]** The primary example of why a provider might do this is to simultaneously support two or more incompatible versions/implementations of the CAMP API, but there are many reasons for a provider to offer multiple instances of the CAMP API.

Concurrent instances are supported through the use of multiple *platform resources*. The platform_endpoints resource allows Consumers to discover all the instances of the CAMP API that are currently available. It contains an array of Links to platform_endpoint resources (that each reference *platform resources*), and has the following general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "platform_endpoints",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "platform_endpoint_links": Link[]
}
```

Note: Because of the unique function of this resource, future versions of the CAMP specification are cautioned against making non-backwards compatible changes to this resource.

A Provider MAY expose the platform_endpoints and corresponding platform_endpoint resources in a way that allows for version discovery before the client has authenticated. **[RE-17]**

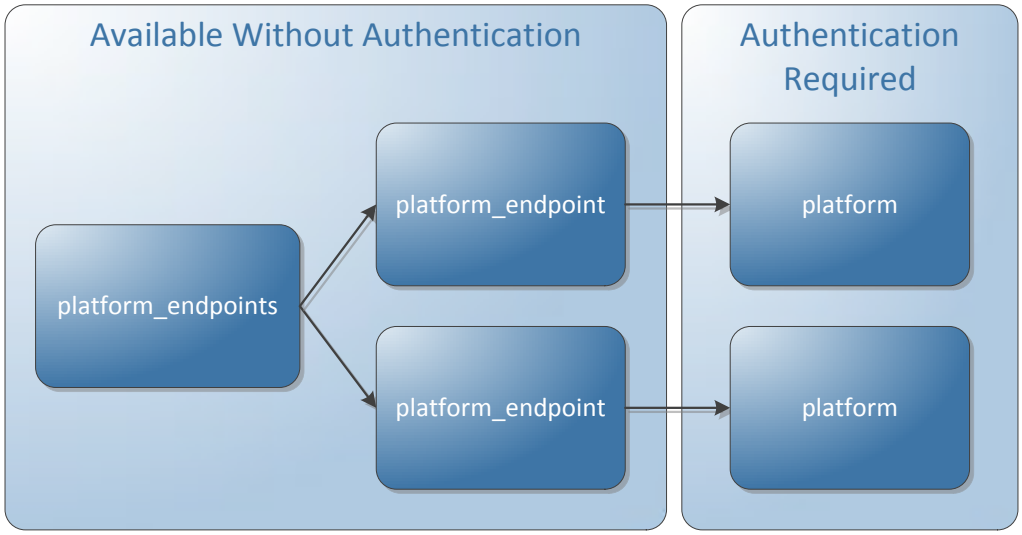

Figure 5-1: Example Implementation

The platform_endpoints resource contains the following attributes:

5.6.1 platform_endpoint_links

Type: Link []

Required: true

Mutable: false

This attribute is an array of Links to *platform_endpoint resources*. This array SHALL contain at least one Link. **[RE-18]** References between the resources (*platform_endpoints*, *platform_endpoint*, and *platform*) SHALL be self-consistent. **[RE-19]**

5.7 platform_endpoint Resource

Each *platform_endpoint resource* SHALL refer to exactly one *platform resource*, and indicate the versions supported by the Platform. **[RE-20]** This specification is deliberately silent about any relationship between resources within different *platform* trees. Each *platform resource* could represent a different CAMP API

"view" of the same applications and services. On the other hand, each *platform* could represent a completely independent system.

A *platform_endpoint resource* has the following general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "platform_endpoint",
  "description": String ?,
  "tags": String[], ?
  "representation_skew": String ?,
  "platform_uri": URI,
  "specification_version": String,
  "backward_compatible_specification_versions": String[] ?,
  "implementation_version": String ?,
  "backward_compatible_implementation_versions": String[] ?,
  "auth_scheme": String ?
}
```

Note: Because of its unique function, future versions of the CAMP specification are cautioned against making non-backwards compatible changes to this resource.

Instances of the platform_endpoint resource contain the following attributes:

5.7.1 platform_uri

Type: URI

Required: true

Mutable: false

This attribute is the URI of the *platform resource* that this *platform_endpoint resource* describes.

5.7.2 specification_version

Type: String

Required: true

Mutable: false

Each *platform resource* is the root of a tree of resources, the syntax and semantics of which conform to one or more versions of the CAMP specification. The value of this attribute is the Specification Version String of the CAMP specification that is supported by the resources rooted in the Platform referenced by the `platform_uri` attribute of this resource.

For Platforms that implement this version of the CAMP specification, the value of this attribute SHALL be as defined in Section 1.8, "Specification Version". **[RE-22]**

5.7.3 backward_compatible_specification_versions

Type: String[]

Required: false

Mutable: false

The values in this array identify each version of the CAMP specification that is backwards compatible with the current `specification_version` of the Platform (referenced in the `platform_uri` attribute of this resource). The values in this array SHALL be the Specification Version Strings of previous CAMP specification versions. **[RE-23]**

If this attribute is not present, the version of the CAMP specification implemented by the Platform (referenced in the `platform_uri` attribute of this resource) is not backwards compatible with any previous version of the CAMP specification.

platform_endpoint resources that reference *platform resources* with a `specification_version` value of "CAMP 1.1" SHALL NOT include this attribute because no previous versions are compatible. **[RE-24]**

5.7.4 implementation_version

Type: String

Required: false

Mutable: false

Multiple implementations of the same CAMP specification MAY be offered concurrently. **[RE-25]** For example, a Provider could offer an initial beta version of "CAMP 1.1" and, later, a production version; allowing a period of overlap for their customers to migrate from the beta to the production version. The value of this attribute is an arbitrary String that expresses the Provider-specific implementation version supported by the resources rooted in the Platform (referenced in the `platform_uri` attribute of this resource).

5.7.5 backward_compatible_implementation_versions

Type: String[]

Required: false

Mutable: false

The values in this array list the provider-specific implementation versions that are backwards compatible with the implementation version of the Platform (referenced in the `platform_uri` attribute of this resource). The values in this array are arbitrary Strings that correspond to previous `implementation_version` Strings.

If this attribute is not present, the implementation version offered by the Platform (referenced in the `platformURI` attribute of this resource) is not backwards compatible with any previous implementation versions.

5.7.6 auth_scheme

Type: String

Required: false

Mutable: false

The value of the auth_scheme attribute indicates the authentication scheme expected by the referenced Platform. For interoperability reasons, Providers are encouraged to offer at least one of the following three (case sensitive) values:

Value	Description
RFC2617	HTTP Basic Authentication [RFC2617]
RFC6749	OAuth2 [RFC6749]
KEYSTONE-2.0	OpenStack Keystone Authentication. [Keystone]
NONE	No authentication required.

Table 5-2 - auth_scheme values

Providers are allowed to extend this list, and provide values of their own. Absence of this attribute means that no authentication scheme is advertised.

Note: Omitting the auth_scheme attribute is discouraged for interoperability reasons.

Note: If Providers wish to offer multiple authentication schemes, they may use multiple platform_endpoint resources each with a different auth_scheme value.

5.8 platform Resource

The *platform resource* represents the Consumer's initial view of the accessible resources and deployed entities. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "platform",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "supported_formats_uri": URI,
  "extensions_uri": URI,
  "type_definitions_uri": URI,
  "platform_endpoints_uri": URI,
  "specification_version": String,
  "implementation_version": String ?,
  "assemblies_uri": URI,
  "services_uri": URI,
  "plans_uri": URI ?
}
```

The *platform resource* contains the following attributes:

5.8.1 supported_formats_uri

Type: URI

Required: false

Mutable: false

This attribute is a URL reference to the *formats resource* for the purpose of identifying all Supported Formats for this Platform. See Section 5.16, "formats Resource", for details.

5.8.2 extensions_uri

Type: URI

Required: true

Mutable: false

This attribute is a URL reference to the Extensions this Platform supports. See Section 7.2, "extensions Resource", for details.

5.8.3 type_definitions_uri

Type: URI

Required: true

Mutable: false

This attribute is a URL reference to the *type_definitions resource* that provides information on the resource types that the Platform supports. See Section 5.18, "type_definitions Resource", for details.

5.8.4 platform_endpoints_uri

Type: URI

Required: true

Mutable: false

This attribute is a URL reference to the *platform_endpoints resource*. The *platform_endpoints resource* enumerates the currently available CAMP implementations. See Section 5.6, "platform_endpoints Resource", for details.

5.8.5 specification_version

Type: String

Required: true

Mutable: false

Each *platform resource* is the root of a tree of resources, the syntax and semantics of which conform to one or more versions of the CAMP specification. The value of this attribute is the Specification Version String of the CAMP specification that is supported by the resources rooted in this Platform.

For Platforms that implement this version of the CAMP specification, the value of this attribute SHALL be as defined in Section 1.8, "Specification Version". **[RE-26]**

The value of this attribute SHALL exactly match the value of the `specification_version` attribute of any *platform_endpoint resource* that references this *platform resource*. **[RE-27]**

5.8.6 implementation_version

Type: String

Required: false

Mutable: false

A Provider MAY choose to offer multiple implementations of the same CAMP specification. **[RE-28]** For example, a Provider could offer an initial beta version of "CAMP 1.1" and, later, a production version; allowing a period of overlap for their customers to migrate from the beta to the production version. The value of this attribute is an arbitrary String that expresses the Provider-specific implementation version supported by the resources rooted in this Platform.

The value of this attribute SHALL exactly match the value of the `implementation_version` attribute of any *platform_endpoint resource* that references this *platform resource*. **[RE-29]**

5.8.7 assemblies_uri

Type: URI

Required: true

Mutable: false

This attribute is a URL reference to the *assemblies resource*. The *assemblies resource* enumerates the applications deployed on this platform. See Section 5.9, "assemblies Resource", for details.

5.8.8 services_uri

Type: URI

Required: true

Mutable: false

This attribute is a URL reference to the *services resource*. The *services resource* enumerates the services available to the Consumer on this platform. See Section 5.12, "services Resource", for details.

5.8.9 plans_uri

Type: URI

Required: false

Mutable: false

This attribute is a URL reference to the *plans resource*. The (optional) *plans resource* enumerates the *plans* deployed on this platform. See Section 5.14, "plans Resource", for details.

5.9 assemblies Resource

This resource acts as a container for the *assembly resources* on this platform. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "assemblies",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "assembly_links": Link[] ?,
  "parameter_definitions_uri": URI
}
```

In addition to the methods defined in Section 5.5, "HTTP Method Support", Providers SHALL support the HTTP POST method on the *assemblies resource* as described in Section 6.11, "Deploying an Application". **[RMR-02]**

The *assemblies resource* contains the following attributes:

5.9.1 assembly_links

Type: Link[]

Required: false

Mutable: true

Consumer-mutable: false

This attribute contains Links to the *assembly resources* that represent the applications deployed on this platform.

5.9.2 parameter_definitions_uri

Type: URI

Required: true

Mutable: false

The value of the `parameter_definitions_uri` attribute references a resource that contains links to *parameter_definitions resources* that describe the parameters accepted by this resource on an HTTP POST method. Each of the *parameter_definition resources* provides metadata for a parameter as described in Section 5.21, "parameter_definitions Resource". The *assemblies resource* accepts the `pdp_uri`, `plan_uri`, `pdp_file`, or `plan_file` parameters to create a new *assembly resource* upon a POST. The *assemblies resource* SHALL indirectly reference *parameter_definition resources* that describes the `pdp_uri`, `plan_uri`, `pdp_file`, and `plan_file` parameters. **[RMR-03]**

5.10 assembly Resource

An *assembly resource* represents an instantiated application at runtime. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "assembly",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "components": Link[],
  "plan_uri": URI ?,
  "operations_uri": URI ?,
  "sensors_uri": URI ?
}
```

In addition to the methods defined in Section 5.5, "HTTP Method Support", Providers SHALL support the HTTP DELETE method on the *assembly resource*. **[RE-61]** On reception of a DELETE request a Provider SHALL remove the *assembly resource* from the system along with any *component resources* referenced by that assembly resource. (i.e. the tree of resources that was created when the application was instantiated). **[RE-73]** On reception of a DELETE request a Provider SHALL remove the reference to the *assembly resource* from the *assemblies resource's* `assembly_links` array. **[RE-74]**

An *assembly resource* contains the following attributes:

5.10.1 components

Type: Link[]

Required: true

Mutable: true

Consumer-mutable: false

The value of the `components` attribute is an array of Links to the *component resources* that make up this *assembly resource*. An *assembly resource* SHALL have at least one reference to a *component resource*. **[RE-39]**

5.10.2 plan_uri

Type: URI

Required: false

Mutable: false

The value of this optional attribute is a URL reference to the *plan resource* for this *assembly resource*. Providers that support Plans SHALL include this attribute in all *assembly resources*. **[RMR-04]**

5.10.3 operations_uri

Type: URI

Required: false

Mutable: false

This attribute contains the URI of the *operations resource*. The *operations resource* lists the *operation resource* links available for the *assembly resource*.

5.10.4 sensors_uri

Type: URI

Required: false

Mutable: false

This attribute contains a URI of the *sensors resource* listing the *sensor resources* available on this resource.

5.11 component Resource

A *component* represents a runtime component. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "component",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "assemblies": Link[],
  "artifact": URI ?,
  "service": URI ?,
  "status": String,
  "external_management_resource": URI ?
  "related_components": Link[] ?,
  "operations_uri": URI ?,
  "sensors_uri": URI ?
}
```

In addition to the methods defined in Section 5.5, "HTTP Method Support", Providers SHALL support the HTTP DELETE method on the *component resource*. **[RE-62]** A successful DELETE request stops the underlying component, removes the *component resource* from the system, and removes its reference from the `components` array of its containing *assembly resource*.

Each *component resource* contains the following attributes:

5.11.1 assemblies

Type: URI

Required: true

Mutable: true

Consumer-mutable: false

The value of the `assemblies` attribute is an array of Links that reference to the *assembly resources* of which this *component resource* is a member.

5.11.2 artifact

Type: URI

Required: false

Mutable: false

The value of the `artifact` attribute is a URL reference to the artifact on which this *component resource* is based. This artifact is not a CAMP resource, but a representation of the actual artifact (e.g. WAR file, Ruby gem file, etc.)

The `artifact` attribute and the `service` attribute are mutually exclusive.

5.11.3 service

Type: URI

Required: false

Mutable: false

The value of the `service` attribute is a URL reference to the *service resource* on which this *component resource* is based.

The `service` attribute and the `artifact` attribute are mutually exclusive.

5.11.4 status

Type: String

Required: true

Mutable: true

Consumer-mutable: false

The value of this attribute indicates the status of the component represented by the *component resource*. This attribute MAY have one of the following values:

- "RUNNING" – indicates that the component is functioning as expected.
- "ERROR" – indicates that the component has encountered some sort of error. **[RE-68]**

Providers MAY extend this list with additional values. **[RE-69]**

The value of this attribute can change in response to the invocation of an operation (see Section 5.24, "operation Resource") or as a result of some change in the underlying system.

As with other attributes, the value of this attribute cannot be construed to accurately reflect the status of the underlying component if the `representation_skew` has a value other than "NONE".

5.11.5 external_management_resource

Type: URI

Required: false

Mutable: false

A URI to an external management interface to manage the underlying component (such as an IaaS API to manage the virtual machines that support this component). The entity referred to by this attribute is platform dependent and requires external documentation to understand its meaning.

5.11.6 related_components

Type: Link[]

Required: false

Mutable: false

This attribute is an array of Links to the *component resources* that this *component* is related to.

5.11.7 operations_uri

Type: URI

Required: false

Mutable: false

This attribute contains the URI of the *operations resource*. The *operations resource* lists the *operation resource* links available for the *component resource*.

5.11.8 sensors_uri

Type: URI

Required: false

Mutable: false

This attribute contains a URI of the *sensors resource* listing the *sensor resources* available on this resource.

5.12 services Resource

This resource acts as a container for the *service resources* of this platform. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "assemblies",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "service_links": Link[] ?,
}
```

The Services resource contains the following attributes:

5.12.1 service_links

Type: Link[]

Required: false

Mutable: true

Consumer-mutable: false

This attribute contains Links to the *service resources* that represent the services available to the Consumer.

5.13 service Resource

A *service resource* represents a particular configuration of a service available for use by one or more applications. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "service",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "parameter_definitions_uri": URI ?
  "characteristics": [
    {
      characteristic_type: String,
      String: String *
    } +
  ] ?
}
```

The *service resource* contains the following attributes:

5.13.1 parameter_definitions_uri

Type: URI

Required: false

Mutable: false

This attribute references the URI of the *parameter_definitions* resource that defines parameters that may be passed to this resource. The *parameter_definitions* resource referenced by this attribute SHALL define parameters to allow setting the ‘name’, ‘description’, and ‘tags’ attributes of any new resource created in the course of interacting with this resource. **[RE-37]**

If this attribute is present in the resource, Providers SHALL support the POST method on that resource in addition to the methods defined in Section 5.5, "HTTP Method Support". **[RE-38]**

5.13.2 characteristics

Type: Array of CharacterisicSpecifications

Required: false

Mutable: false

The optional `characteristics` attribute describes the capabilities of the service described by the *service resource*. The elements of this array have the same schema as the CharacteristicSpecification (described in Section 4.3.7, "CharacteristicSpecification") of a Plan.

Note that this specification is deliberately silent about the process of matching the ServiceSpecifications in a Plan to the services described by *service resources*. Any correspondence between the information in a Plan's ServiceSpecification and the information in the `characteristics` attribute does not necessarily constitute a contract to resolve the containing requirement with that service, though Providers are free to implement and advertise such contracts if they wish.

5.14 plans Resource

This optional resource acts as a container for the *plan resources* deployed by the Consumer. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "plans",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "plan_links": Link[] ?,
  "parameter_definitions_uri": URI
}
```

In addition to the methods defined in Section 5.5, "HTTP Method Support", Providers SHALL support the HTTP POST method on the *plans resource* as described in Section 6.12, "Registering a Plan". **[RMR-05]**

The Plans resource contains the following attributes:

5.14.1 plan_links

Type: Link[]

Required: false

Mutable: true

Consumer-mutable: false

This attribute contains Links to the *plan resources* that represent the blueprints for applications deployed on the platform.

5.14.2 parameter_definitions_uri

Type: URI

Required: true

Mutable: false

The value of the `parameter_definitions_uri` attribute references a resource that contains links to *parameter_definition resources* that describe the parameters accepted by this resource on an HTTP POST method. Each of the *parameter_definition resources* provides metadata for a parameter as described in Section 5.21, "parameter_definitions Resource". The Plans resource accepts the `pdp_uri`,

`plan_uri`, `pdp_file`, or `plan_file` parameters to create a new *plan resource* upon a POST. The *plans resource* SHALL indirectly reference *parameter_definition resources* that describe the pdp_uri, plan_uri, pdp_file, and plan_file parameters. **[RMR-06]**

5.15 plan Resource

This optional resource stores the *plan* for an application. As discussed in Section 2.3, "Deployment", this information is supplied to the platform as part of the operation of deploying an application. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "plan",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "camp_version": String,
  "origin": String ?,
  "artifacts": [
    {
      "name": String ?,
      "description": String ?,
      "tags": String[] ?,
      "artifact_type": String,
      "content": { "href": URI },
      "requirements": [
        {
          "requirement_type": String,
          "fulfillment": {
            "name": String ?,
            "description": String ?,
            "tags": String[] ?,
            "id": String ?,
            "href": URI ?,
            "characteristics": [
              {
                "characteristic_type": String,
                String: String *
              } +
            ] ?
          } ?
        } +
      ] ?,
    } +
  ] ?,
  "services": [
    {
      "name": String ?,
      "description": String ?,
      "tags": String[] ?,
      "id": String ?,
      "href": URI ?,
      "characteristics": [
        {
          "characteristic_type": String,
          String: String *
        } +
      ] ?,
    } +
  ] ?
}
```

The schema of the *plan resource* returned from a CAMP Provider SHALL conform to the schema for Plans described in Section 4.3, "Plan Schema", with the following additional requirements: **[RMR-07]**

- Representations of the *plan resource* SHALL be serialized as JSON, unless another format is negotiated. **[RMR-08]**
- Any href attributes of ServiceSpecifications SHOULD refer to a Service resource. **[RMR-09]**
- All href attributes in the *plan resource* SHOULD be set to a consumer accessible URL. If the original Plan file referred to a local file, the URL indicates where the Provider stored the content. **[RMR-10]**
- The *plan resource* inherits from the *camp_resource* defined in Section 5.4, "camp_resource Resource", and therefore inherits all its attributes. The value for the `type` attribute is "plan".

For example, if the consumer-supplied Plan file describes an artifact with an href pointing to a file contained in a PDP, the platform-supplied *plan resource* will point to a copy of that artifact, such as one hosted at the platform or in an object store.

Support for the *plan resource* is uniform across a CAMP implementation. Regardless of whether a Consumer attempts to create an *assembly resource* by POSTing to the *assemblies resource* or creates a *plan resource* by POSTing to the *plans resource*, a Provider that supports *plans* and *plan resources* SHALL create a *plan resource* for every deployed application. **[RMR-11]**

5.15.1 Advertising Support for the Plan Resource

As an aid to interoperability it is helpful if Consumers can easily discover if a particular Provider supports the *plans resource* and *plan resources*. Section 7.2, "extensions Resource", defines a mechanism for advertising extensions to the CAMP specification. This mechanism is used to advertise support for the *plans resource* and *plan resources*.

Providers that support the *plans* and *plan resources* SHALL advertise such support using the following *extension resource*: **[RMR-12]**

```
{
  "uri": <as appropriate>,
  "name": "CAMP Plans Extension",
  "type": "extension",
  "description": "indicates support for the plans and plan resources",
  "version": "CAMP 1.1",
  "documentation": "http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-
v1.1.pdf"
}
```

5.16 formats Resource

The Formats resource contains an array of Links to Format resources. It allows the identification of Supported Formats. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "formats",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "format_links": Link[]
}
```

The Formats resource contains the following attribute:

5.16.1 format_links

Type: Link[]

Required: true

Mutable: false

This attribute contains Links to Format resources that contain information about data serialization formats supported by the Platform. For every format that the Platform supports, there SHALL be a Format resource Link that represents such a format. **[RE-40]** The Required JSON Format Resource SHALL be listed first in the format_links array. **[RE-41]**

5.17 format Resource

A Format resource represents exactly one supported data serialization format. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "format",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "mime_type": String,
  "version": String ?,
  "documentation": URI
}
```

Instances of the Format resource contain the following attributes:

5.17.1 mime_type

Type: String

Required: true

Mutable: false

This attribute contains the mime-type to be used by the Platform in HTTP [RFC2616] compliant content negotiation for this Format. For example: “application/json”.

5.17.2 version

Type: String

Required: false

Mutable: false

This attribute contains the version identifier of the data serialization format used.

5.17.3 documentation

Type: URI

Required: true

Mutable: false

The value of the `documentation` attribute is a URI reference to a document that describes the format identified by this resource. See the following sub-section for an example.

5.17.4 Required JSON Format Resource

The Required JSON Format Resource is defined as:

```
{
  "uri": URI,
  "name": "JSON",
  "type": "format",
  "description": "JavaScript Object Notation",
  "tags": String[] ?,
  "mime_type": "application/json",
  "version": "RFC4627",
  "documentation": "http://www.ietf.org/rfc/rfc4627.txt"
}
```

The *name*, *mime_type*, *version*, and *documentation* attribute values for the JSON Format Resource SHALL reflect the above values. **[RE-42]**

5.18 type_definitions Resource

This resource contains an array of Links to the *type_definition resources*. The *platform resource* SHALL provide a Link to the *type_definitions resource* in the required attribute named `type_definitions_uri`. **[RE-43]** This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "typeDefinitions",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "type_definition_links": Link[]
}
```

The *type_definitions resource* contains the following attribute:

5.18.1 type_definition_links

Type: Link[]

Required: true

Mutable: false

This attribute contains Links to *type_definition resources* that contain information about resource types supported by the Platform. If the Platform does not extend this specification to add new resource types then the array can be empty. If the array is non-empty, for every resource type that the Platform supports, there SHALL be a *type_definition resource* Link that represents such a resource type. **[RE-44]** To help developers implement this requirement a package containing the *type_definition resources* for every resource defined in this specification is provided as a non-normative auxiliary file.

5.19 type_definition Resource

A *type_definition resource* describes a resource type supported by the Platform. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "typeDefinition",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "documentation": URI,
  "inherits_from": Link[],
  "attribute_definition_links": AttributeLink[]
}
```

The value of the `name` attribute in a *type_definition resource* SHALL match the value of the `type` attribute for the resource type that it describes. **[RE-75]** This constraint allows Consumers to locate the metadata that describes a resource in the `typeDefinitions` array of the *type_definitions resource* using the `type` value of that resource as a key.

The *type_definition resource* contains the following attributes:

5.19.1 documentation

Type: URI

Required: true

Mutable: false

This attribute contains a URI that points to the documentation for the resource type. For resource types that are defined in this specification, the URI can point to this specification.

5.19.2 inherits_from

Type: Link[]

Required: false

Mutable: false

This attribute contains an array of Links. Each Link in this array points to a *type_definition resource* that the described resource's type inherits from. Links in this array SHALL NOT either directly or transitively point to the described resource. **[MO-06]** If a type inherits only from the *camp_resource* type then this attribute MAY be absent. **[MO-07]**

5.19.3 attribute_definition_links

Type: AttributeLink[]

Required: true

Mutable: false

This attribute contains an array extended of Link elements termed "AttributeLinks". Each AttributeLink in this array references an *attribute_definition resource*. Each of these *attribute_definition resources* represents an attribute of the type described by this *type_definition resource*.

For every attribute of the type not inherited from its super-types, there SHALL be an AttributeLink that references the *attribute_definition resource* that defines that attribute. **[RE-45]** In cases where a sub-type adds additional constraints to an attribute inherited from its super-types (e.g. makes an optional attribute required), a Provider SHALL include an AttributeLink that references the *attribute_defintion resource* for that attribute. **[RE-76]** For more information on the *attribute_definition resource* see the next section.

AttributeLinks are extensions of the Link attribute type with the following, additional sub-attributes:

5.19.3.1 required

Type: Boolean

Required: true

Mutable: false

The value of the `required` attribute determines if the attribute defined by the *attribute_definition resource* referenced by this AttributeLink is required for resources of the type defined by the containing *type_definition resource*. A value of "true" indicates that the referenced attribute will always be present in resources of the type defined by the containing *type_definition resource*.

5.19.3.2 mutable

Type: Boolean

Required: true

Mutable: false

The value of the `mutable` attribute determines if the attribute defined by the *attribute_definition resource* referenced by this AttributeLink is mutable for resources of the type defined by the containing *type_definition resource*. A value of “true” indicates that the referenced attribute can change during the lifetime of resources of the type defined by the containing *type_definition resource*.

5.19.3.3 consumer_mutable

Type: Boolean

Required: false

Mutable: false

The value of the `consumer_mutable` attribute determines if the attribute defined by the *attribute_definition resource* referenced by this AttributeLink is writable by Consumers for resources of the type defined by the containing *type_definition resource*. A value of “true” indicates that Consumers can change the referenced attribute for resources of the type defined by the containing *type_definition resource*. This attribute is not required in cases when the attribute defined by the *attribute_definition resource* referenced by this AttributeLink is not mutable (see above).

5.20 attribute_definition Resource

An *attribute_definition resource* represents exactly one supported attribute of one or more resource types. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "attributeDefinition",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "documentation": URI,
  "attribute_type": String
}
```

Instances of the *attribute_definition resource* contain the following attributes:

5.20.1 documentation

Type: URI

Required: true

Mutable: false

The value of the `documentation` attribute is a URI that references the documentation for the attribute that this resource represents. For attributes that are defined in this specification, this URI references this specification.

5.20.2 attribute_type

Type: String

Required: true

Mutable: false

The value of the `attribute_type` attribute specifies the type of the attribute that this described by this resource. See Section 5.2, “Attribute Types”, for a list of the values defined by this specification.

The appearance of the square bracket symbols, “[]”, appended to the value of the `attribute_type` attribute indicates that the value of the attribute that is described by this resource is an array of the

specified type. For example, an `attribute_type` value of “Link[]” indicates that the value of the attribute being described by is an array of Links.

5.21 parameter_definitions Resource

A *parameter_definitions resource* represents a collection of supported parameters for a particular resource. Multiple resources MAY reference the same *parameter_definitions resource*. **[RE-46]** This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "parameter_definitions",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "parameter_definition_links": ParameterLink[]
}
```

parameter_definitions resources contain the following attributes:

5.21.1 parameter_definition_links

Type: ParameterLink[]

Required: true

Mutable: false

The value of the `parameter_definition_links` attribute is an array of extended Link elements termed “ParameterLinks”. Each ParameterLink in this array refers to one *parameter_definition resource*.

ParameterLinks are extensions of the Link attribute type with the following, additional sub-attributes:

5.21.1.1 required

Type: Boolean

Required: true

Mutable: false

The value of the `required` attribute specifies whether the parameter defined by the *parameter_definition resource* referenced by this ParameterLink is required for HTTP POST requests on the resource that references the containing *parameter_definitions resource*. A value of “true” indicates that the referenced parameter is required for all POST requests on the resource that references the containing *parameter_definitions resource*.

5.21.1.2 default_value

Type: As defined by referenced *parameter_definition resource*.

Required: false

Mutable: false

The value of the `default_value` attribute, when present, specifies the default value for the parameter defined by the *parameter_definition resource* referenced by this ParameterLink. If the Consumer does not supply a value for the parameter defined by the *parameter_definition* resource referenced by this ParameterLink, the value of this attribute will be used. Note that the presence of the `default_value` attribute is mutually exclusive with a `required` value (see above) of “true”.

5.22 parameter_definition Resource

A *parameter_definition resource* represents exactly one supported parameter of one or more resource types. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "parameter_definition",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "parameter_type": String,
  "parameter_extension_uri": String ?
}
```

parameter_definition resources contain the following attributes:

5.22.1 parameter_type

Type: String

Required: true

Mutable: false

This attribute specifies the type of the attribute that this resource represents. For example, "String", "Timestamp".

5.22.2 parameter_extension_uri

Type: URI

Required: false

Mutable: false

If this parameter is handled by an extension, this attribute refers to the *extension resource* that represents that Extension and documents how the parameter is handled.

5.23 operations Resource

An *operations resource* represents a collection of *operation resources* available on a target resource. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "operations",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "target_resource": URI,
  "operation_links": Link[]
}
```

Instances of the *operations resource* contain the following attributes:

5.23.1 target_resource

Type: URI

Required: true

Mutable: false

This attribute indicates the CAMP resource on which the linked operations are invoked. Linked operations are those referred to by the operation_links attribute. We use the term “target resource” to identify the resource referred to by this attribute.

5.23.2 operation_links

Type: Link[]

Required: true

Mutable: false

This attribute contains Links to the *operation resources* available on the target resource.

5.24 operation Resource

An *operation resource* represents exactly one operation or action available on a target resource. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "operation",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "documentation": URI,
  "target_resource": URI,
  "parameter_definitions_uri": URI ?
}
```

In addition to the methods defined in Section 5.5, "HTTP Method Support", Providers SHALL support the HTTP POST method on the *operation resource*. **[RE-64]**

A POST request on the *operation resource* invokes the Operation on the target resource. The Operation MAY require content in the body of the POST, such as parameters. **[RE-47]** The response to a POST request on an *operation resource* SHOULD indicate what changes were made on the target resource. **[RE-48]** For asynchronous operations, the response SHOULD indicate how to track the progress of the request operation. **[RE-49]**

NOTE: For asynchronous operations, a Provider can accept a webhook URL from the Consumer as a parameter to the Operation POST request and notify the client at that URL upon completion of the operation. It can also allow for polling of the resource to indicate completion.

Instances of the *operation resource* contain the following attributes:

5.24.1 name

Type: String

Required: true

Mutable: false

This attribute contains the name of the operation that this resource represents. For example, "deploy" or "resize".

5.24.2 documentation

Type: URI

Required: true

Mutable: false

This attribute contains a URI of documentation for the operation this resource represents. The documentation SHOULD describe the behavior of the operation, the form of the body expected in POST requests, and the semantics and form of the response to such requests. **[RE-50]**

5.24.3 target_resource

Type: URI

Required: true

Mutable: false

This attribute indicates the CAMP resource on which the linked operation is invoked.

5.24.4 parameter_definitions_uri

Type: URI

Required: false

Mutable: false

The value of the `parameter_definitions_uri` attribute is a URI that references a *parameter_definitions resource* that contains links to the *parameter_definition resources* that describe the parameters accepted by this resource on an HTTP POST method. Each of the *parameter_definition resources* provides metadata for a parameter as described in Section 5.21, "parameter_definitions Resource".

5.25 sensors Resource

A *sensors resource* represents a collection of *sensor resources* available on a target resource. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "sensors",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "target_resource": URI,
  "sensor_links": Link[]
}
```

Instances of the *sensors resource* contain the following attributes:

5.25.1 target_resource

Type: URI

Required: true

Mutable: false

This attribute indicates the CAMP resource for which the linked sensors supply runtime data. Linked sensors are those referred to by the sensor_links attribute. We use the term "target resource" to identify the resource referred to by this attribute.

5.25.2 sensor_links

Type: Link[]

Required: true

Mutable: false

This attribute contains Links to the *sensor resources* available on the target resource.

5.26 sensor Resource

A *sensor resource* represents exactly one supported sensor on one or more resources. A *sensor resource* represents dynamic data about resources, such as metrics or state. A *sensor resources* is useful

for exposing data that changes rapidly, or that may need to be fetched from a secondary system. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "sensor",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "documentation": URI,
  "target_resource": URI,
  "sensor_type": String,
  "value": <sensor_type> ?,
  "timestamp": Timestamp ?,
  "operations_uri": URI ?
}
```

Instances of the *sensor resource* contain the following attributes:

5.26.1 documentation

Type: URI

Required: true

Mutable: false

This attribute contains a URI that points to the documentation for the sensor this resource represents.

5.26.2 target_resource

Type: URI

Required: true

Mutable: false

This attribute indicates the CAMP resource for which this *sensor resource* supplies runtime data.

5.26.3 sensor_type

Type: String

Required: true

Mutable: false

This attribute specifies the type of the data that this *sensor resource* collects. For example, "String", "Timestamp". Attribute types are defined in Section 5.2, “Attribute Types”. type_definitions may also be used to specify types. See Section 5.18, “type_definitions Resource”.

5.26.4 value

Type: As defined in sensor_type

Required: false

Mutable: true

Consumer-mutable: false

This attribute contains the current or most recent available value for this sensor. It can be omitted, for example, to indicate that no current value is available; either because no data has been collected or the collected data is stale.

5.26.5 timestamp

Type: Timestamp

Required: false

Mutable: false

This attribute contains the timestamp of the last collection or relevant activity of the sensor. When a "value" attribute is supplied, any timestamp provided in this attribute SHOULD correspond to when that value was observed. **[RE-51]**

5.26.6 operations_uri

Type: URI

Required: false

Mutable: false

This attribute contains the URI of the *operations resource*. The *operations resource* lists the *operation resource* links available for the *sensor resource*.

Extensions MAY be defined to govern common sensor management operations, such as enabling, disabling, adjusting collection frequency, specifying the history of values which should be remembered, or collecting immediately. **[RE-52]**

6 Protocol

6.1 Transfer Protocol

The CAMP API is based on the Hypertext Transfer Protocol, version 1.1 [RFC2616]. Requests sent from Consumers across unsecured networks SHOULD use the HTTPS protocol. **[PR-40]** TLS 1.1 [RFC4346] SHALL be implemented by the Provider. **[PR-41]** TLS 1.2 [RFC5246] is RECOMMENDED. **[PR-42]** When TLS is implemented, the following cipher suites are RECOMMENDED to ensure a minimum level of security and interoperability between implementations:

- TLS_RSA_WITH_AES_128_CBC_SHA (mandatory for TLS 1.1/1.2) **[PR-43]**
- TLS_RSA_WITH_AES_256_CBC_SHA256 (addresses 112-bit security strength requirements) **[PR-44]**

Note: For interoperability reasons, Providers are encouraged to support a common authentication scheme in order to simplify the implementation of client tools that are intended to work with multiple Providers. The *platform_endpoint resource* `auth_scheme` attribute (see Section 5.7.6, "auth_scheme") makes available authentication schemes discoverable by unauthenticated clients.

6.2 URI Space

The resources in the system are identified by URIs. Dereferencing the URI will yield a representation of the resource containing resource attributes and links to associated resources.

Note: Consumers are cautioned against making assumptions about the layout of the URIs or the structure of the URIs of the resources.

6.3 Media Types

6.3.1 Required Formats

In this specification resource representations are encoded in JSON, as specified in [RFC4627]. The media-type associated with CAMP JSON resource representations is "application/json".

Providers SHALL provide representations of all available resources in JSON. **[PR-01]**

6.3.1.1 Duplicate Keys in JSON Objects

CAMP defined JSON objects do not contain duplicate keys. Consumers and Providers SHALL NOT transmit JSON objects that contain duplicate keys. **[PR-02]**

If a Consumer sends a Provider a request containing duplicate keys in a JSON object, the Provider SHOULD reject the request by sending back a '400 Bad Request' status code. **[PR-03]** If a Provider sends a Consumer a response containing duplicate keys in a JSON object, the Consumer SHOULD raise an error to the user indicating the response from the server was malformed. **[PR-04]**

Note: Duplicate keys in JSON objects are allowed by JSON [RFC4627]. This specification prohibits duplicate keys for interoperability reasons.

6.3.2 Supported Formats

If Supported Formats besides JSON are defined in the *formats resource* referenced by the supported_formats_uri attribute of the *platform resource*, then the indicated resource representations are allowed in the Supported Formats.

For each Supported Format, Consumers MAY request any resource from the Provider in that format. **[PR-45]** Providers SHALL respond in the requested Supported Format. **[PR-05]**

A client can request any Supported Format using HTTP content negotiation.

6.4 Request Headers

This API does not impose any requirements on clients' use of HTTP headers. All PUT requests that update a resource SHOULD contain the If-Match header field with a single entity tag value. **[PR-06]** If the If-Match header field value in the request does not match the one on the server-side, the Provider SHALL send back a '412 Precondition Failed' status code. **[PR-07]**

6.5 Request Parameters

To retrieve a subset of the attributes in a resource, the Consumer MAY use the 'select_attr' query parameter in conjunction with the HTTP GET method. **[PR-08]** A Provider SHALL return only those attributes of the queried resource whose name occurs in the list specified by the value of 'select_attr'. **[PR-47]**

Format	Description	Example
?select_attr=*attr1,attr2,…*	Comma (",") separated attribute names of the resource to return. If an attribute listed in the value of the 'select_attr' query parameter is not part of the resource, a "400 Bad Request" status code SHALL be returned. **[PR-09]**	assembly132?select_attr =name%2Cdescription%2Ctags Would access only "name", "description", "tags" attributes of assembly132.

Table 6-1: Request Parameters

The "select_attr" query parameter MAY appear more than once (separated by an "&"). **[PR-10]**

6.6 POST Body Parameters

Parameters MAY be included when performing a POST request on any resource with a `parameter_definitions_uri` attribute defined. **[PR-14]** Supported parameters are defined by the parameter_definitions resource referenced by the parameter_definitions_uri attribute of the resource handling the POST request.

Example of a POST Parameter:

```
POST /<assembly-template-resource-url> HTTP/1.1
Host: example.org
Content-Type: application/json
Content-Length: ...

{ "EXAMPLE:someParameter": "bar" }

HTTP/1.1 201 Created
Location: http://example.org/paas/assembly/1
Content-Type: ...
Content-Length: ...
```

6.6.1 Parameter Handling

Parameters allow customizing resources upon creation. Parameters MAY have the same name as an attribute of the resource. **[PR-15]** In such cases the Provider SHOULD set that attribute to take the value of the parameter OR clearly document alternate behavior. **[PR-16]** The `parameter_extension_uri` MAY be used to reference the extension which documents how the parameter is handled. **[PR-17]**

If a POST request body does not contain a value for a required parameter, a "400 Bad Request" response SHALL be returned. **[PR-18]**

If a POST request body does not contain an acceptable value for a parameter, a “400 Bad Request” response SHALL be returned. **[PR-19]**

6.7 Response Headers

Responses returned by the Provider make standard use of HTTP headers. All HTTP responses that return representation of a resource SHOULD use strong Etag response header field indicating the current value of the entity tag for the resource. **[PR-20]**

6.8 HTTP Status Codes

The API returns standard HTTP response codes.

6.9 Mutability of Resource Attributes

Consumers SHALL NOT send a request that changes the value of a resource attribute that is declared with a constraint of 'Mutable=false' or 'Consumer-mutable=false'. **[PR-21]** On receiving such a request the Provider SHALL generate an HTTP response with 403 HTTP status code. **[PR-22]**

6.10 Updating Resources

Attributes of the resources defined with “consumer_mutable: true” can be modified by Consumers in two ways. Consumers MAY use the HTTP PUT method to replace the representation of a resource, in its entirety, with a new representation that adds, omits or replaces the values for some of the attributes. **[PR-23]** Alternatively, Consumers MAY use the HTTP PATCH [HTTP PATCH] method and the “application/json-patch+json” media type [RFC6902] to add, delete, or replace specific attributes. **[PR-24]**

6.10.1 Updating with PUT

HTTP PUT requests are requests for complete replacement of the resource identified by the request URL.

On successfully processing an HTTP PUT request a Provider SHALL update all the Consumer-mutable attributes of the target resource, and only these, with the values of the matching attributes in the request. **[PR-48]** If a resource attribute is present on a resource and if an HTTP PUT request omits that attribute, it SHOULD be treated by the Provider as a request to delete the attribute. **[PR-25]**

6.10.1.1 Partial Updates with PUT

Section 6.5, “Request Parameters”, describes the use of the ‘select_attr’ query parameter to circumscribe the GET method to a subset of a resource’s attributes. To allow an update of a subset of a resource’s attributes, Providers SHALL support the use of the ‘select_attr’ query parameter in conjunction with the HTTP PUT method. **[PR-76]** A Consumer SHALL NOT include attributes, whose name does not occur in the list specified by the value of the ‘select_attr’ query parameter, in the entity body of a PUT request. **[PR-12]** Upon receiving such a malformed request the Provider SHALL respond with a “400 Bad Request” status code. **[PR-13]**

One way to think about this is to regard the values listed in ‘select_attr’ as a mask on the base resource where HTTP requests are interpreted within the context of this mask. Another way to think about this is to consider a URL that includes the ‘select_attr’ query parameter as dynamically addressing a resource that contains only those attributes whose name occurs in the value of that query parameter.

For example, the following request should be considered as an attempt to delete the ‘tags’ attribute but, because they are not listed in the value of ‘select_attr’, not an attempt to delete any other attributes:

```
PUT /my_paas/assembly/273?select_attr=tags HTTP/1.1
Host: example.org
Content-Type: application/json
Content-Length: …
```

6.10.2 Updating with JSON Patch

JSON Patch [RFC6902] defines a JSON document structure for expressing a sequence of operations to apply to a JSON document, suitable for use with the HTTP PATCH method. The "application/json-patch+json" media type is used to identify such patch documents.

Providers SHALL support the HTTP PATCH method in conjunction with the “application/json-patch+json” media type with the following, additional provisions with respect to the operations defined in Section 4 of the JSON Patch specification: **[PR-26]**

- Providers SHALL support the ‘add’, ‘remove’, and ‘replace’ operations. **[PR-27]**
- Providers MAY support the ‘move’, ‘copy, and ‘test’ operations. **[PR-28]**

6.11 Deploying an Application

Deploying an application uploads the artifacts and metadata that make up the application, allocates the necessary resources and services, and, in the successful case, creates a running application (represented by an *assembly resource*).

There are two ways to deploy an application using a PDP: by POSTing the entire PDP to the *assemblies resource* (by value) or by POSTing the URI of the PDP to the *assemblies resource* (by reference). Similarly, there are two ways to deploy an application using a Plan: by POSTing the entire Plan file to the *assemblies resource* (by value) or by POSTing the URI of the Plan file or *plan resource* to the *assemblies resource* (by reference). All of these methods are described below. Providers SHALL support PDPs that use either the ZIP [ZIP], TAR [TAR], or GZIP [RFC1952] compressed TAR formats. **[RMR-13]**

6.11.1 Deploying an Application by Reference

To deploy an application by reference, a Consumer sends a reference to either a PDP, Plan file, or *plan resource* in a POST request to the *assemblies resource*. Providers SHALL support the deployment of applications via HTTP POST requests on the *assemblies resource* as described in this section. **[PR-49]** The entity body of the request contains a URI that identifies the PDP, Plan file, or *plan resource* that is being deployed. If the URI that identifies the PDP, Plan file, or *plan resource* is a relative URI, its base URI is that of the *platform resource*. When deploying a PDP the JSON serialization of the HTTP request entity body is:

```
{"pdp_uri": "<uri-of-the-pdp>"}
```

When deploying a Plan file or *plan_resource* the JSON serialization of the HTTP request entity body is:

```
{"plan_uri": "<uri-of-the-plan>"}
```

Where, the value of `pdp_uri` is the URI of the PDP to be deployed and the value of `plan_uri` is the URI of the Plan to be deployed. To support the deployment of applications via a reference to either a PDP, Plan file, or *plan resource*, Providers SHALL accept the "application/json" media type. **[PR-68]** The JSON object MAY contain additional name-value pairs that are not defined in this specification. **[PR-33]**

Note that the value of `plan_uri` can refer to either a Plan file or a *plan resource*. A referenced *plan resource* can exist either on the same CAMP Platform as the target *assemblies resource*, or on some other CAMP Platform. In the case where the referenced *plan resource* exists within the same Platform as the target *assemblies resource*, Consumers are advised to use a relative URL for the *plan resource* reference to help Providers identify the *plan resource* as local.

An example HTTP request-response is as follows:

```
POST /paas/assemblies HTTP/1.1
Host: example.org
Content-Type: application/json
Content-Length: 46
…

{"pdp_uri": "/paas/pdp/1",
 "description": "Mike's other Drupal instance"}

HTTP/1.1 201 Created
Location: http://example.org/paas/assembly/11
…
```

Note the inclusion of `description` parameter/attribute in the body of the above POST request.

On successfully processing the request the Provider SHALL create an *assembly resource* and return a 201 Created status code in the HTTP response. **[PR-50]** The Provider SHALL include the *Location* header in the HTTP response and the value of this header SHALL reference the newly created *assembly resource*. **[PR-51]** The Provider SHALL update the `assembly_links` attribute of the *assemblies resource* to include a reference to the newly created *assembly resource*. **[PR-52]**

6.11.2 Deploying an Application by Value

To deploy an application by value, a Consumer transmits the contents of either a PDP or a Plan file in a POST request to the *assemblies resource*. The Consumer can do this in two ways, either by including the data as a part in a multipart MIME message or by simply including the data in the entity body of the HTTP request.

To support the deployment of applications using a PDP, Providers SHALL accept the media types associated with the various formats as follows:

- ZIP: "application/x-zip" **[PR-29]**
- TAR: "application/x-tar" **[PR-30]**
- GZIP compressed TAR: "application/x-tgz" **[PR-31]**

To support the deployment of applications using a Plan file, Providers SHALL accept the use of the "application/x-yaml" media type. **[PR-32]**

On successfully processing the request the Provider SHALL create an *assembly resource* and return a 201 *Created* status code in the HTTP response. **[PR-53]** The Provider SHALL include the *Location* header in the HTTP response and the value of this header SHALL reference the newly created *assembly resource*. **[PR-54]** The Provider SHALL update the `assembly_links` attribute of the *assemblies resource* to include a reference to the newly created *assembly resource*. **[PR-55]**

For large PDPs, the Consumer can use existing HTTP facilities like chunked transfer encoding.

6.11.2.1 Deploying an Application by Value Using MIME

Providers SHALL support the deployment of applications via HTTP POST requests on the *assemblies resource* as described in this section. **[PR-74]** The entity body of the request is a multipart MIME message that contains the PDP or the Plan file that is being deployed. The value of the HTTP Content-Type header is “multipart/form-data” [RFC2388]. The MIME part that contains the file data has the value of the `name` parameter of its `Content-Disposition` header set to “pdp_file”, if a PDP is being deployed, or “plan_file”, if a Plan file is being deployed. CAMP parameters can be included in the request as additional MIME parts using the value of the `name` parameter of the `Content-Disposition` header to indicate the CAMP parameter being included.

An example HTTP request-response is as follows:

```
POST /paas/assemblies HTTP/1.1
Host: example.org
Content-Length: 9768506
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryU6AnBoMx
…

------WebKitFormBoundaryU6AnBoMx
Content-Disposition: form-data; name="pdp_file"; filename="Mike_Drupal2.pdp"
Content-Type: application/x-zip

… binary octets …
------WebKitFormBoundaryU6AnBoMx
Content-Disposition: form-data; name="description"

Mike's other Drupal instance
------WebKitFormBoundaryU6AnBoMx--

HTTP/1.1 201 Created
Location: http://example.org/paas/assembly/12
Content-Type: application/json
…
```

Note the inclusion of the description parameter as a separate MIME part.

6.11.2.2 Deploying an Application by Value Without MIME

Providers SHALL support the deployment of applications via HTTP POST requests on the *assemblies resource* in which the entity body of the request contains the PDP or the Plan file that is being deployed. **[PR-60]**

An example HTTP request-response is as follows:

```
POST /paas/assemblies HTTP/1.1
Host: example.org
Content-Length: 976361
Content-Type: application/x-zip
Content-Transfer-Encoding: binary
…

… binary PDP ZIP octets …

HTTP/1.1 201 Created
Location: http://example.org/paas/assembly/12
…
```

Note that it is not possible to include additional parameters when using this mechanism to deploy by an application.

6.12 Registering a Plan

As described in Section 2.3, "Deployment", CAMP implementations can choose to support the expression of Plans as CAMP resources. This feature allows Consumers to register an application (upload the artifacts and metadata, validate the Plan, resolve dependencies, etc.) without creating a running instance of that application. Consumers can later instantiate an application from the *plan resource*.

Similarly to deploying an application, Plans can be registered using either a PDP or a Plan file. The PDP or Plan file can be supplied by value or by reference.

The archive formats available to the PDP are identical to those used to deploy an application.

6.12.1 Registering a Plan by Reference

To register a Plan by reference, a Consumer sends a reference to either a PDP or a Plan file in a POST request to the plans resource. Providers that support the *plans resource* and *plan resources* SHALL

support the registration of Plans via an HTTP POST request on the *plans resource* as described in this section. **[PR-56]** The entity body of the request contains a URI that identifies the PDP or the Plan file that is being registered. If the URI that identifies the PDP or the Plan file is a relative URI, its base URI is that of the *platform resource*. When registering a PDP the JSON serialization of the HTTP request entity body is:

```
{"pdp_uri": "<uri-of-the-pdp>"}
```

When registering a Plan file the JSON serialization of the HTTP request entity body is:

```
{"plan_uri": "<uri-of-the-plan>"}
```

Where, the value of `pdp_uri` is the URI of the PDP to be registered and the value of `plan_uri` is the URI of the Plan to be registered. To support the registration of Plans via a reference to either a PDP or a Plan file, Providers SHALL accept the "application/json" media type. **[PR-69]** The JSON object MAY contain additional name-value pairs that are not defined in this specification. **[PR-46]**

An example HTTP request-response is as follows:

```
POST /paas/plans HTTP/1.1
Host: example.org
Content-Type: application/json
Content-Length: 46
…

{"pdp_uri": "/paas/pdp/1",
 "description": "Mike's other Drupal instance"}

HTTP/1.1 201 Created
Location: http://example.org/paas/plan/9
…
```

Note the inclusion of `description` parameter/attribute in the body of the above POST request.

On successfully processing the request the Provider SHALL create a *plan resource* and return a 201 Created status code in the HTTP response. **[PR-57]** The Provider SHALL include the *Location* header in the HTTP response and the value of this header SHALL reference the newly created *plan resource*. **[PR-58]** The Provider SHALL update the `plan_links` attribute of the *plans resource* to include a reference to the newly created *plan resource*. **[PR-59]**

6.12.2 Registering a Plan by Value

To register a Plan by value, a Consumer transmits the contents of either a PDP or a Pan file in a POST request to the *plans resource*. The Consumer can do this in two ways, either by including the data as a part in a multipart MIME message or by simply including the data in the entity body of the HTTP request.

To support the registration of Plans using a PDP, Providers SHALL accept the media types associated with the various formats as follows:

- ZIP: "application/x-zip" **[PR-70]**
- TAR: "application/x-tar" **[PR-71]**
- GZIP compressed TAR: "application/x-tgz" **[PR-72]**

To support the registration of Plans using a Plan file, Providers SHALL accept the use of the "application/x-yaml" media type. **[PR-73]**

On successfully processing the request the Provider SHALL create a *plan resource* and return a 201 Created status code in the HTTP response. **[PR-62]** The Provider SHALL include the *Location* header in the HTTP response and the value of this header SHALL reference the newly created *plan resource*. **[PR-63]** The Provider SHALL update the `plan_links` attribute of the *plans resource* to include a reference to the newly created *plan resource*. **[PR-64]**

For large PDPs, the Consumer can use existing HTTP facilities like chunked transfer encoding.

6.12.2.1 Registering a Plan by Value Using MIME

Providers that support the plans resource and plan resources SHALL support the registration of Plans via HTTP POST requests on the plans resource as described in this section. **[PR-75]** The entity body of the request is a multipart MIME message that contains the PDP or the Plan file that is being registered. The value of the HTTP `Content-Type` header is "multipart/form-data" [RFC2388]. The MIME part that contains the file data has the value of the `name` parameter of its `Content-Disposition` header set to "pdp_file", if a PDP is being registered, or "plan_file", if a Plan file is being registered. CAMP parameters can be included in the request as additional MIME parts using the value of the `name` parameter of the `Content-Disposition` header to indicate the CAMP parameter being included.

An example HTTP request-response is as follows:

```
POST /paas/plans HTTP/1.1
Host: example.org
Content-Length: 1685
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryE733b300
…

------WebKitFormBoundaryE733b300
Content-Disposition: form-data; name="plan_file"; filename="Mike_Drupal.yaml"
Content-Type: application/x-yaml

… unicode characters …
------WebKitFormBoundaryE733b300
Content-Disposition: form-data; name="description"

Mike's other Drupal instance
------WebKitFormBoundaryE733b300--

HTTP/1.1 201 Created
Location: http://example.org/paas/plan/9
…
```

Note the inclusion of the description parameter as a separate MIME part.

6.12.2.2 Registering a Plan by Value Without MIME

Providers that support the *plans resource* and *plan resources* SHALL support the registration of Plans via HTTP POST requests on the *plans resource* in which the entity body of the request contains the PDP or the Plan file that is being registered. **[PR-61]**

An example HTTP request-response is as follows:

```
POST /paas/plans HTTP/1.1
Host: example.org
Content-Length: 1465
Content-Type: application/x-yaml
…

… unicode characters …

HTTP/1.1 201 Created
Location: http://example.org/paas/plan/9
…
```

Note that it is not possible to include additional parameters when using this mechanism to register a Plan.

6.13 Stopping an Application Instance

One way of stopping an application instance is to send an HTTP DELETE request to the URL of the corresponding *assembly resource*.

7 Extensions

Features provided by this specification can be extended to provide additional information and functionality. Extensions MAY be added by registering the new functionality in the *extensions resource*. **[EX-02]** Extensions SHALL NOT change or remove any features or functionality of this specification. **[EX-03]** Each extension SHALL satisfy all the criteria in Section 8, "Conformance", and SHALL NOT contradict any normative statements in this document. **[EX-04]** The following extensions are allowed:

Category	Functionality	Description
API Extension	New HTTP Request Verbs	Support for additional HTTP Request Verbs that are not used by this specification, such as HEAD.
API Extension	HTTP Header Handlers	Processing of specific HTTP headers provided by clients. For example, an API Extension may require an authentication token header.
API Extension	New Resources	Addition of new resource types that MAY handle HTTP requests such as POST or PUT to create new resources of this type.
API Extension	New Resource Methods	Allow the creation of new methods or actions that may cause different sequences of state changes than happen by default.
PDP Extension	New Metadata in the PDP	Additional metadata provided in the PDP to allow for more sophisticated handling of the bundled data.
Resource Extension	New Resource Types	Addition of new resource types.
Resource Extension	New Resource Attributes	Addition of new attributes to existing resources.
Resource Extension	New States in any Application Lifecycle	Addition of new application states, such as an intermediate state between the states defined by the specification.

Table 7-1: Extension Categories and Functionality

7.1 Unique Name Requirement

Entities
• Resources • Attributes

- Methods
- PDP Metadata Keys

Table 7-2: Entities

Entities are enumerated in Table 7-2. The Extension Developer SHALL use a unique name for new entities within an existing namespace. **[EX-05]** Entities added by an extension SHALL NOT interfere with names of existing entities, including any added by another extension. **[EX-06]**

NOTE: Each resource has its own namespace. It is acceptable to create a resource named *example.org:Foo*, and another resource named *example.org:Bar*, where both resources have an attribute named *fooBar*.

The use of your registered ICAAN Internet domain name followed by a colon (":") character as a prefix to all your entity names is RECOMMENDED to comply with these requirements. **[EX-07]**

Example: New Attribute "foo" added by Example Organization

```
example.org:foo
```

Example: New Attribute "foo" added by Example Inc.

```
EXAMPLE-INC:foo
```

Extension Category	New Entity	Exception
API Extension	Adding HTTP Request Verbs	Unique name not required for HTTP verbs
API Extension	Adding HTTP Header Handlers	Unique name not required for HTTP headers

Table 7-3: Unique Name Exceptions

A unique name is not required for entities listed in Table 7-3.

NOTE: RFC-3986 identifies Unreserved Characters that may be used in a URI without any encoding. Percent-Encoding allows any character to be represented in a URI. Special characters such as ":" and "." have specific meanings in scripting languages such as JavaScript. Special characters must be properly escaped in order to use them as part of a name string. Your data serialization format may not escape all problematic characters, so you may need to add logic to your clients to escape special characters to enable interaction with an Extension.

7.2 extensions Resource

The *extensions resource* contains an array of Links to *extension resources*. It allows the identification of extensions. The *extensions resource* is represented as:

```
{
  "uri": URI,
  "name": String,
  "type": "extensions",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "extension_links": Link[]
}
```

The *extensions resource* contains the following attribute:

7.2.1 extension_links

Type: Link[]

Required: true

Mutable: false

This attribute contains Links to *extension resources* that contain information about the extensions available on the Platform. For every extension available, there SHALL be an *extension resource* Link that represents the extension. **[EX-08]** The *platform resource* SHALL provide a Link to the *extensions resource* in the required attribute named `extensions_uri`. **[EX-09]**

Example of an `extension_links` value:

```
[
  {
    "target_name" : "EXAMPLE:Auth",
    "href": "http://example.org/paas1/extension/1"
  },
  {
    "target_name" : "EXAMPLE:PDPforFooLang",
    "href" : "http://example.org/paas1/extension/2"
  }
  …
]
```

7.3 extension Resource

An *extension resource* represents new functionality added to the Platform. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "extension",
  "description": String ?,
  "tags": String[] ?,
  "representation_skew": String ?,
  "version": String,
  "documentation": URI ?
}
```

The *extension resource* contains the following attributes:

7.3.1 version

Type: String

Required: true

Mutable: false

This attribute contains a string identifier of the version of this extension.

7.3.2 documentation

Type: URI

Required: false

Mutable: false

This attribute is a URI that references a human readable document that describes the extension.

7.4 Extending Existing Resources

New attributes MAY be added to an existing resource using an *extension resource* if the Unique Name Requirement in 7.1 is met. **[EX-10]** A new resource type is not required in order to add new attributes.

Example of an extended *extension resource*:

```
{
  "uri": URI,
  "name": String,
  "type": "extension",
  "description": String,
  "version": String,
  "documentation": URI ?,
  "acme.com:foo": String ?
}
```

Note that in the above example, the new attribute `acme.com:foo` was added, and the type attribute remained set to the original value “extension”.

8 Conformance

There are three conformance targets defined in this specification:

- CAMP Provider
- CAMP Consumer
- Platform Deployment Package
- Plan

8.1 CAMP Provider

An implementation claiming to conform to the requirements of a CAMP Provider defined in this specification SHALL comply with all of the CAMP Provider or Provider mandatory requirements listed in this specification, as summarized in Appendix C.1, "Mandatory Statements".

8.2 CAMP Consumer

An implementation claiming to conform to the requirements of a CAMP Consumer defined in this specification SHALL comply with all of the CAMP Consumer or Consumer mandatory requirements listed in this specification, as summarized in Appendix C.1, "Mandatory Statements".

8.3 Platform Deployment Package

For a document to be a valid PDP, it SHALL comply with all of the PDP mandatory requirements listed in this specification, as summarized in Appendix C.1, "Mandatory Statements".

8.4 Plan

For a document to be a valid Plan, it SHALL comply with all of the Plan mandatory requirements listed in this specification, as summarized in Appendix C.1, "Mandatory Statements".

Appendix A. Acknowledgments

This appendix is non-normative. The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

Roshan Agrawal	Rackspace Hosting, Inc.
Michael Behrens	US Department of Defense (DoD)
Bhaskar Reddy Byreddy	Software AG, Inc.
Mark Carlson	Oracle
Martin Chapman	Oracle
Francesco D'Andria	Cloud4SOA
Jacques Durand	Fujitsu Limited
Panagiotis Gouvas	Cloud4SOA
Keith Grange	JumpSoft
Alex Heneveld	Cloudsoft Corporation Limited
Gershon Janssen	Individual Member
David Jilk	Standing Cloud, Inc.
Duncan Johnston-Watt	Cloudsoft Corporation Limited
Anish Karmarkar	Oracle
Tobias Kunze	Red Hat
Eugene Luster	US Department of Defense (DoD)
Ashok Malhotra	Oracle
Alex McDonald	NetApp
Rich Miller	Cloudsoft Corporation Limited
Jeff Mischkinsky	Oracle
Adrian Otto	Rackspace Hosting, Inc.
Derek Palma	Vnomic
Gilbert Pilz	Oracle
Krishna Raman	Red Hat
Tom Rutt	Fujitsu Limited
Zhexuan Song	Huawei Technologies Co., Ltd.
Charles Tupitza	JumpSoft
Jeffrey West	Oracle
Prasad Yendluri	Software AG, Inc.

Appendix B. Glossary

Application – a set of components that act together to provide useful functions and are typically exposed as a service to Application end-users.

Artifact - a static element of an application that either provides a set of related services and functionality or contains a set of related information. Code examples include Ruby gems, Java libraries, and PHP modules. Examples of resources include data sets, identity sets (i.e. collections of user account and attribute information), and collections of graphical images.

Component – a dynamic element of an application that provides a set of related services and functionality. Examples include Ruby processes, Spring web applications, and database instances.

Application Development Environment (ADE) – a developer tool used to create an application (can be an offline tool installed locally or part of the platform offering itself).

Assembly – a management resource that represents a running application.

Deploy – the act of using a PDP or Plan to create a running application (represented by an *assembly resource*) on the platform

Extension - a systematic representation of additional features and functionality added by an Extension Developer.

Plan - packaging management meta-data that provides a description of the artifacts that make up an application, the services that are required to execute or utilize those artifacts, and the relationship of the artifacts to those services.

Platform – The collection of management resources that constitute the consumer visible view of the Platform as a Service offering. The Platform management resource is an aggregation and discovery point for all the Applications and their dependencies currently deployed and running.

Platform as a Service (PaaS) - A type of cloud computing in which the service provider offers customers/consumers access to one or more instances of a running application computing platform or application service stack.

Platform Deployment Package (PDP) - an archive of executable images, dependency descriptions and metadata (including a Plan file) that can be used to move an Application and its Components from Platform to Platform, or between an Application Development Environment and a Platform (e.g. a storefront application with component binaries, database images and all the configurations needed to install and run).

Register – the act of creating a Plan on the platform.

Supported Formats - one or more data serialization format for data representation. JSON format is required, but other data serialization formats are also allowed. The *platform resource* identifies all Supported Formats in the optional supported_formats_uri attribute. If the supported_formats_uri attribute is absent from the *platform resource*, then only JSON is supported.

Appendix C. Normative Statements

C.1 Mandatory Statements

Tag	Statement
[PDP-02]	A Provider SHALL support the following archive formats for a PDP: • A PDP as a ZIP archive [ZIP]
[PDP-03]	A Provider SHALL support the following archive formats for a PDP: • A PDP as a TAR archive [TAR]
[PDP-04]	A Provider SHALL support the following archive formats for a PDP: • A PDP as a GZIP [RFC1952] compressed TAR archive
[PDP-10]	The format of the manifest file and the certificate file SHALL be as defined by the OVF specification [OVF].
[PDP-11]	A Platform Deployment Package (PDP) SHALL contain a single Plan file.
[PDP-27]	Providers SHALL support the “https” URI scheme as defined in RFC 2818 [RFC2818].
[PDP-29]	Providers SHALL understand this delimiter and SHALL NOT resolve any content if the archive format is unsupported.
[PLAN-01]	The Plan file SHALL be located at the root of the PDP archive.
[PLAN-02]	The Plan file SHALL be named “`camp.yaml`”.
[PLAN-03]	A Plan file SHALL contain a single instance of a Plan.
[PLAN-05]	For Plans that conform to this document, the value of this node SHALL be as defined in Section 1.8 “Specification Version”.
Error! Reference source not found.	Plans SHALL use id values that are unique within the scope of the Plan.
[PLAN-07]	Consumers SHALL follow the syntax and semantics described here when using URIs with a “pdp” scheme.
[PLAN-08]	The Plan file SHALL conform to YAML 1.1 [YAML 1.1].
[PLAN-09]	The Plan file SHALL conform to the description provided in this section.
[RE-06]	If the Required boolean constraint for an attribute of a resource type has a value of "true", then a resource of this type SHALL have the attribute present.
[RE-07]	This boolean indicates the mutability of the attribute’s value(s). “false” indicates that the value of the attribute, once set, SHALL NOT change for the lifetime of the resource.
[RE-09]	“false” indicates that the value(s) of the attribute SHALL NOT be changed by Consumers.
[RE-11]	If present, `representation_skew` SHALL have one of the following values: • “CREATING” – describes a resource that is in the process of being created. The client can expect that the resource will have a skew of “NONE” once this process

<table>
<tr><td></td><td>has completed.
“NONE” – is an assertion by the CAMP server that the information in the resource is an accurate representation of the underlying platform implementation. Absent some action by the client or some other event (e.g. platform shutdown), a resource with a skew of NONE can be expected to remain in synch with the platform implementation.
“UNKNOWN” – indicates that the CAMP server cannot accurately depict the aspect of the platform implementation represented by this resource. Users can attempt to address the underlying issues(s) by manipulating this and/or other resources as specified by the API.
“DESTROYING” – describes a resource that is in the process of being destroyed. The client can expect that the resource will cease to exist once this process has completed.</td></tr>
<tr><td>[RE-12]</td><td>The following table lists the methods that SHALL be supported for each <code>representation_skew</code> value.<table><tr><th>representation_skew value</th><th>Methods Available</th></tr><tr><td>CREATING</td><td>GET, DELETE</td></tr><tr><td>NONE</td><td>All supported methods for that resource.</td></tr><tr><td>UNKNOWN</td><td>All supported methods for that resource.</td></tr><tr><td>DESTROYING</td><td>GET</td></tr></table></td></tr>
<tr><td>[RE-18]</td><td>This array SHALL contain at least oneLink.</td></tr>
<tr><td>[RE-19]</td><td>References between the resources (platform_endpoints, platform_endpoint, and platform) SHALL be self-consistent.</td></tr>
<tr><td>[RE-20]</td><td>Each platform_endpoint resource SHALL refer to exactly one platform resource, and indicate the versions supported by the Platform.</td></tr>
<tr><td>[RE-22]</td><td>For Platforms that implement this version of the CAMP specification, the value of this attribute SHALL be as defined in Section 1.8, “Specification Version”.</td></tr>
<tr><td>[RE-23]</td><td>The values in this array SHALL be the Specification Version Strings of previous CAMP specification versions.</td></tr>
<tr><td>[RE-24]</td><td>platform_endpoint resources that reference platform resources with a <code>specification_version</code> value of “CAMP 1.1” SHALL NOT include this attribute because no previous versions are compatible.</td></tr>
<tr><td>[RE-26]</td><td>For Platforms that implement this version of the CAMP specification, the value of this attribute SHALL be as defined in Section 1.8, “Specification Version”.</td></tr>
<tr><td>[RE-27]</td><td>The value of this attribute SHALL exactly match the value of the <code>specification_version</code> attribute of any platform_endpoint resource that references this platform resource.</td></tr>
<tr><td>[RE-29]</td><td>The value of this attribute SHALL exactly match the value of the <code>implementation_version</code> attribute of any platform_endpoint resource that references this platform resource.</td></tr>
<tr><td>[RE-37]</td><td>The parameter_definitions resource referenced by this attribute SHALL define parameters to allow setting the ‘name’, ‘description’, and ‘tags’ attributes of any new resource created</td></tr>
</table>

	in the course of interacting with this resource.
[RE-38]	If this attribute is present in the resource, Providers SHALL support the POST method on that resource in addition to the methods defined in Section 5.5, "HTTP Method Support".
[RE-39]	An *assembly resource* SHALL have at least one reference to a *component resource*.
[RE-40]	For every format that the Platform supports, there SHALL be a Format resource Link that represents such a format.
[RE-41]	The Required JSON Format Resource SHALL be listed first in the format_links array.
[RE-42]	The name, mime_type, version, and documentation attribute values for the JSON Format Resource SHALL reflect the above values.
[RE-43]	The *platform resource* SHALL provide a Link to the *type_definitions resource* in the required attribute named `type_definitions_uri`.
[RE-44]	If the array is non-empty, for every resource type that the Platform supports, there SHALL be a *type_definition resource* Link that represents such a resource type.
[RE-45]	For every attribute of the type not inherited from its super-types, there SHALL be an AttributeLink that references the *attribute_definition resource* that defines that attribute.
[RE-53]	Providers SHALL support the HTTP GET, PUT, and PATCH methods on all of the resources defined in this section.
[RE-61]	In addition to the methods defined in Section 5.5, "HTTP Method Support", Providers SHALL support the HTTP DELETE method on the *assembly resource*.
[RE-62]	In addition to the methods defined in Section 5.5, "HTTP Method Support", Providers SHALL support the HTTP DELETE method on the *component resource*.
[RE-64]	In addition to the methods defined in Section 5.5, "HTTP Method Support", Providers SHALL support the HTTP POST method on the *operation resource*.
[RE-65]	Consumers and Providers SHALL express Timestamps in UTC (Coordinated Universal Time), with the special UTC designator ("Z").
[RE-70]	When supporting such a Resource, a Provider SHALL implement it and serialize it as described in the corresponding sub-section.
[RE-71]	A Consumer SHALL serialize Resource data in its requests based on the definition of this Resource as described in the corresponding sub-section.
[RE-73]	On reception of a DELETE request a Provider SHALL remove the *assembly resource* from the system along with any *component resources* referenced by that assembly resource. (i.e. the tree of resources that was created when the application was instantiated).
[RE-74]	On reception of a DELETE request a Provider SHALL remove the reference to the *assembly resource* from the *assemblies resource's* `assembly_links` array.
[RE-75]	The value of the `name` attribute in a *type_definition resource* SHALL match the value of the `type` attribute for the resource type that it describes.
[RE-76]	In cases where a sub-type adds additional constraints to an attribute inherited from its super-types (e.g. makes an optional attribute required), a Provider SHALL include an AttributeLink that references the *attribute_defintion resource* for that attribute.
[PR-01]	Providers SHALL provide representations of all available resources in JSON.

[PR-02]	Consumers and Providers SHALL NOT transmit JSON objects that contain duplicate keys.
[PR-05]	Providers SHALL respond in the requested Supported Format.
[PR-07]	If the *If-Match* header field value in the request does not match the one on the server-side, the Provider SHALL send back a '412 Precondition Failed' status code.
[PR-09]	If an attribute listed in the value of the 'select_attr' query parameter is not part of the resource, a "400 Bad Request" status code SHALL be returned.
[PR-12]	A Consumer SHALL NOT include attributes, whose name does not occur in the list specified by the value of the 'select_attr' query parameter, in the entity body of a PUT request.
[PR-13]	Upon receiving such a malformed request the Provider SHALL respond with a "400 Bad Request" status code.
[PR-18]	If a POST request body does not contain a value for a required parameter, a "400 Bad Request" response SHALL be returned.
[PR-19]	If a POST request body does not contain an acceptable value for a parameter, a "400 Bad Request" response SHALL be returned.
[PR-21]	Consumers SHALL NOT send a request that changes the value of a resource attribute that is declared with a constraint of 'Mutable=false' or 'Consumer-mutable=false'.
[PR-22]	On receiving such a request the Provider SHALL generate an HTTP response with 403 HTTP status code.
[PR-26]	Providers SHALL support the HTTP PATCH method in conjunction with the "application/json-patch+json" media type with the following, additional provisions with respect to the operations defined in Section 4 of the JSON Patch specification:
[PR-27]	Providers SHALL support the 'add', 'remove', and 'replace' operations.
[PR-29]	To support the deployment of applications using a PDP, Providers SHALL accept the media types associated with the various formats as follows: • ZIP: "application/x-zip"
[PR-30]	To support the deployment of applications using a PDP, Providers SHALL accept the media types associated with the various formats as follows: • TAR: "application/x-tar"
[PR-31]	To support the deployment of applications using a PDP, Providers SHALL accept the media types associated with the various formats as follows: • GZIP compressed TAR: "application/x-tgz"
[PR-32]	To support the deployment of applications using a Plan file, Providers SHALL accept the use of the "application/x-yaml" media type.
[PR-41]	TLS 1.1 [RFC4346] SHALL be implemented by the Provider.
[PR-47]	A Provider SHALL return only those attributes of the queried resource whose name occurs in the list specified by the value of 'select_attr'.
[PR-48]	On successfully processing an HTTP PUT request a Provider SHALL update all the Consumer-mutable attributes of the target resource, and only these, with the values of the matching attributes in the request.

[PR-49]	To deploy an application by reference, a Consumer SHALL send an HTTP POST request to the URL of the *assemblies resource* as described in this section.
[PR-50]	On successfully processing the request the Provider SHALL create an *assembly resource* and return a 201 Created status code in the HTTP response.
[PR-51]	The Provider SHALL include the *Location* header in the HTTP response and the value of this header SHALL reference the newly created *assembly resource*.
[PR-52]	The Provider SHALL update the `assembly_links` attribute of the *assemblies resource* to include a reference to the newly created *assembly resource*.
Error! Reference source not found.	On successfully processing the request the Provider SHALL create an *assembly resource* and return a 201 Created status code in the HTTP response.
Error! Reference source not found.	The Provider SHALL include the *Location* header in the HTTP response and the value of this header SHALL reference the newly created assembly resource.
Error! Reference source not found.	The Provider SHALL update the `assembly_links` attribute of the *assemblies resource* to include a reference to the newly created *assembly resource*.
[PR-56]	Providers that support the *plans resource* and *plan resources* SHALL support the registration of Plans via an HTTP POST request on the *plans resource* as described in this section.
[PR-57]	On successfully processing the request the Provider SHALL create a *plan resource* and return a 201 Created status code in the HTTP response.
[PR-58]	The Provider SHALL include the *Location* header in the HTTP response and the value of this header SHALL reference the newly created *plan resource*.
[PR-59]	The Provider SHALL update the `plan_links` attribute of the *plans resource* to include a reference to the newly created *plan resource*.
[PR-60]	Providers SHALL support the deployment of applications via HTTP POST requests on the *assemblies resource* in which the entity body of the request contains the PDP or Plan file that is being deployed.
[PR-61]	Providers that support the *plans resource* and *plan resources* SHALL support the registration of Plans via HTTP POST requests on the *plans resource* in which the entity body of the request contains the PDP or the Plan file that is being registered.
[PR-62]	On successfully processing the request the Provider SHALL create a *plan resource* and return a 201 Created status code in the HTTP response.
[PR-63]	The Provider SHALL include the *Location* header in the HTTP response and the value of this header SHALL reference the newly created *plan resource*.
[PR-64]	The Provider SHALL update the `plan_links` attribute of the Plans resource to include a reference to the newly created *plan resource*.
[PR-68]	To support the deployment of applications via a reference to either a PDP, Plan file, or *plan* resource, Providers SHALL accept the "application/json" media type.
[PR-69]	To support the registration of Plans via a reference to either a PDP or a Plan file,

	Providers SHALL accept the "application/json" media type.
Error! Reference source not found.	To support the registration of Plans using a PDP, Providers SHALL accept the media types associated with the various formats as follows: • ZIP: "application/x-zip"
Error! Reference source not found.	To support the registration of Plans using a PDP, Providers SHALL accept the media types associated with the various formats as follows: • TAR: "application/x-tar"
Error! Reference source not found.	To support the registration of Plans using a PDP, Providers SHALL accept the media types associated with the various formats as follows: • GZIP compressed TAR: "application/x-tgz"
Error! Reference source not found.	To support the registration of Plans using a Plan file, Providers SHALL accept the use of the "application/x-yaml" media type.
[PR-74]	Providers SHALL support the deployment of applications via HTTP POST requests on the *assemblies resource* as described in this section.
[PR-75]	Providers that support the *plans resource* and *plan resources* SHALL support the registration of Plans via HTTP POST requests on the *plans resource* as described in this section.
[PR-76]	To allow an update of a subset of a resource's attributes, Providers SHALL support the use of the 'select_attr' query parameter in conjunction with the HTTP PUT method.
[EX-03]	Extensions SHALL NOT change or remove any features or functionality of this specification.
[EX-04]	Each extension SHALL satisfy all the criteria in Section 8, "Conformance", and SHALL NOT contradict any normative statements in this document.
[EX-05]	The Extension Developer SHALL use a unique name for new entities within an existing namespace.
[EX-06]	Entities added by an extension SHALL NOT interfere with names of existing entities, including any added by another extension.
[EX-08]	For every extension available, there SHALL be an *extension resource* Link that represents the extension.
[EX-09]	The *platform resource* SHALL provide a Link to the *extensions resource* in the required attribute named `extensions_uri`.
Error! Reference source not found.	If a Consumer includes this node in a Plan, the value of this node SHALL reference a Consumer-visible resource within the target Platform.
[RMR-02]	In addition to the methods defined in Section 5.5, "HTTP Method Support", Providers SHALL support the HTTP POST method on the *assemblies resource* as described in Section 6.11, "Deploying an Application".
[RMR-03]	The *assemblies resource* SHALL indirectly reference *parameter_definition resources* that describe the `pdp_uri`, `plan_uri`, `pdp_file`, and `plan_file` parameters.

[RMR-04]	Providers that support Plans SHALL include this attribute in all *assembly resources*.
[RMR-05]	In addition to the methods defined in Section 5.5, "HTTP Method Support", Providers SHALL support the HTTP POST method on the *plans resource* as described in Section 6.12, "Registering a Plan".
[RMR-06]	The *plans resource* SHALL indirectly reference *parameter_definition resources* that describe the `pdp_uri`, `plan_uri`, `pdp_file`, and `plan_file` parameters.
[RMR-07]	The schema of the *plan resource* returned from a CAMP Provider SHALL conform to the schema for Plans described in Section 4.3, "Plan Schema", with the following additional requirements:
[RMR-08]	Representations of the *plan resource* SHALL be serialized as JSON, unless another format is negotiated.
[RMR-11]	Regardless of whether a Consumer attempts to create an *assembly resource* by POSTing to the *assemblies resource* or creates a *plan resource* by POSTing to the *plans resource*, a Provider that supports the *plans* and *plan resources* SHALL create a *plan resource* for every deployed application.

[RMR-12]	Providers that support *plans* and *plan resources* SHALL advertise such support using the following *extension resource*: **[RMR-12]**
	`{`
	`  "uri": <as appropriate>,`
	`  "name": "CAMP Plans Extension",`
	`  "type": "extension",`
	`  "description": "indicates support for plans and plan resources",`
	`  "version": "CAMP 1.1",`
	`  "documentation": "http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.pdf"`
	`}`

[RMR-13]	Providers SHALL support PDPs that use either the ZIP [ZIP], TAR [TAR], or GZIP [RFC1952] compressed TAR formats.
[MO-02]	A sub-type SHALL NOT loosen the constraints of an attribute inherited from its super-type(s).
[MO-03]	A resource type MAY inherit from more than one super-type.
[MO-04]	If there is an attribute name collision when a sub-type inherits from multiple super-types, the inherited attributes of the same name SHALL NOT contradict the constraints and semantics of the attributes defined in its super-types.
[MO-05]	All CAMP resources SHALL inherit directly or indirectly from this resource.
[MO-06]	Links in this array SHALL NOT either directly or transitively point to the described resource.

C.2 Non-Mandatory Statements

Tag	Statement
[PDP-01]	A PDP archive MAY include other files related to the application including, but not limited to, language-specific bundles, resource files, application content files such as web archives, database schemas, scripts, source code, localization bundles, and icons; and metadata files

	such as manifests, checksums, signatures, and certificates.
[PDP-05]	Providers MAY support additional archive formats for the PDP.
[PDP-06]	A PDP MAY contain a manifest file, named `camp.mf`, at the root of the archive.
[PDP-07]	A Provider SHOULD reject a PDP if any digest listed in the manifest does not match the computed digest for that file in the package.
[PDP-08]	A PDP MAY contain a certificate, named `camp.cert`, at the root of the archive.
[PDP-09]	A Provider SHOULD reject any PDP for which the signature verification fails.
[PDP-14]	Providers MAY reflect the value of this attribute in the names of any resources that are created in the processing the *plan*.
[PDP-15]	Providers MAY reflect the value of this attribute in the descriptions of the resources that are in the processing the *plan*.
[PDP-16]	Providers MAY reflect the values of this attribute in the tags of the resources that are created in the processing of the *plan*.
[PDP-22]	The artifact MAY be contained within the PDP or MAY exist in some other location.
[PDP-28]	A Provider MAY support additional URI schemes listed at http://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml.
[RE-08]	“true” indicates that the value of the attribute MAY change due to the actions or activity of either the provider or the consumer.
[RE-10]	A value of “true” indicates that Consumers MAY change the value of the attribute.
[RE-13]	For each `representation_skew` value, CAMP Providers MAY support HTTP methods in addition to those listed in the corresponding row of Table 5-1.
[RE-15]	A Provider MAY concurrently offer multiple instances of the CAMP API.
[RE-17]	A Provider MAY expose the platform_endpoints and corresponding platform_endpoint resources in a way that allows for version discovery before the client has authenticated.
[RE-25]	Multiple implementations of the same CAMP specification MAY be offered concurrently.
[RE-28]	A Provider MAY choose to offer multiple implementations of the same CAMP specification.
[RE-46]	Multiple resources MAY reference the same *parameter_definitions resource*.
[RE-47]	The Operation MAY require content in the body of the POST, such as parameters.
[RE-48]	The response to a POST request on an *operation resource* SHOULD indicate what changes were made on the target resource.
[RE-49]	For asynchronous operations, the response SHOULD indicate how to track the progress of the request operation.
[RE-50]	The documentation SHOULD describe the behavior of the operation, the form of the body expected in POST requests, and the semantics and form of the response to such requests.
[RE-51]	When a “value” attribute is supplied, any timestamp provided in this attribute SHOULD correspond to when that value was observed.
[RE-52]	Extensions MAY be defined to govern common sensor management operations, such as enabling, disabling, adjusting collection frequency, specifying the history of values which should be remembered, or collecting immediately.

[RE-54]	Providers MAY elect to support additional HTTP methods in addition to those described here.
[RE-68]	This attribute MAY have one of the following values: • "RUNNING" – indicates that the component is functioning as expected. • "ERROR" – indicates that the component has encountered some sort of error.
[RE-69]	Providers MAY extend this list with additional values.
[PR-03]	If a Consumer sends a Provider a request containing duplicate keys in a JSON object, the Provider SHOULD reject the request by sending back a '400 Bad Request' status code.
[PR-04]	If a Provider sends a Consumer a response containing duplicate keys in a JSON object, the Consumer SHOULD raise an error to the user indicating the response from the server was malformed.
[PR-06]	All PUT requests that update a resource SHOULD contain the *If-Match* header field with a single entity tag value.
[PR-08]	To retrieve a subset of the attributes in a resource, the Consumer MAY use the 'select_attr' query parameter in conjunction with the HTTP GET method.
[PR-10]	The "select_attr" query parameter MAY appear more than once (separated by an "&").
[PR-14]	Parameters MAY be included when performing a POST request on any resource with a `parameter_definitions_uri` attribute defined.
[PR-15]	Parameters MAY have the same name as an attribute of the resource.
[PR-16]	In such cases the Provider SHOULD set that attribute to take the value of the parameter OR clearly document alternate behavior.
[PR-17]	The `parameter_extension_uri` MAY be used to reference the extension which documents how the parameter is handled.
[PR-20]	All HTTP responses that return representation of a resource SHOULD use strong *Etag* response header field indicating the current value of the entity tag for the resource.
[PR-23]	Consumers MAY use the HTTP PUT method to replace the representation of a resource, in its entirety, with a new representation that adds, omits or replaces the values for some of the attributes.
[PR-24]	Alternatively, Consumers MAY use the HTTP PATCH [HTTP PATCH] method and the "application/json-patch+json" media type [RFC6902] to add, delete, or replace specific attributes.
[PR-25]	If a resource attribute is present on a resource and if an HTTP PUT request omits that attribute, it SHOULD be treated by the Provider as a request to delete the attribute.
[PR-28]	Providers MAY support the 'move', 'copy, and 'test' operations.
[PR-33]	The JSON object MAY contain additional name-value pairs that are not defined in this specification.
[PR-40]	Requests sent from Consumers across unsecured networks SHOULD use the HTTPS protocol.
[PR-42]	TLS 1.2 [RFC5246] is RECOMMENDED.
[PR-43]	When TLS is implemented, the following cipher suites are RECOMMENDED to ensure a minimum level of security and interoperability between implementations:

	• TLS_RSA_WITH_AES_128_CBC_SHA (mandatory for TLS 1.1/1.2)
[PR-44]	When TLS is implemented, the following cipher suites are RECOMMENDED to ensure a minimum level of security and interoperability between implementations: • TLS_RSA_WITH_AES_256_CBC_SHA256 (addresses 112-bit security strength requirements)
[PR-45]	For each Supported Format, Consumers MAY request any resource from the Provider in that format.
[PR-46	The JSON object MAY contain additional name-value pairs that are not defined in this specification.
[EX-02]	Extensions MAY be added by registering the new functionality in the *extensions resource*.
[EX-07]	The use of your registered ICAAN Internet domain name followed by a colon (":") character as a prefix to all your entity names is RECOMMENDED to comply with these requirements.
[EX-10]	New attributes MAY be added to an existing resource using an Extension if the Unique Name Requirement in 7.1 is met.
[RMR-09]	Any href attributes of ServiceSpecifications SHOULD refer to a Service resource.
[RMR-10]	All href attributes in the *plan resource* SHOULD be set to a consumer accessible URL. If the original Plan file referred to a local file, the URL indicates where the Provider stored the content.
[MO-01]	A sub-type MAY further restrict the constraints of an attribute inherited from its super-type(s).
[MO-07]	If a type inherits only from the *camp_resource* type then this attribute MAY be absent.

Appendix D. Example Version Scheme

This appendix is non-normative. The table below describes the version semantics from the Maven POM Syntax [POM-Syntax] for your consideration. In the absence of a prevailing version range scheme for your types, this approach is suggested. Range values can be used as version string values to convey matching semantics to the platform. This is useful when using a CharacteristicSpecification (see Section 4.3.7 “CharacteristicSpecification”) that can be satisfied by more than one potential match.

Range	Meaning
1.0	x >= 1.0
(,1.0]	x <= 1.0
(,1.0)	x < 1.0
[1.0]	x == 1.0
[1.0,)	x >= 1.0
(1.0,)	x > 1.0
(1.0,2.0)	1.0 < x < 2.0
[1.0,2.0]	1.0 <= x <= 2.0
(,1.0],[1.2,)	x <= 1.0 or x >= 1.2. Multiple sets are comma-separated.
(,1.1),(1.1,)	x != 1.1

Table D-1: POM Version Semantics

In cases where a prevailing scheme is already popular for a given technology, use the prevailing scheme.

Appendix E. Revision History

Revision	Date	Editor	Changes Made
1	2012-12-04	Anish Karmarkar	Applied OASIS template to version 1.0
2	2012-12-18	Anish Karmarkar	Included resolutions for issues 7, 12, 13, 15, 24, 33.
3	2013-02-05	Anish Karmarkar	Included resolutions for issues 2, 6, 10, 14, 25, 35
4	2013-02-12	Adrian Otto	Included resolutions for 19, 38
5	2013-02-13	Adrian Otto	Included resolutions for 1, 49
6	2013-02-27	Adrian Otto	Included resolutions for 36, 48, 53
7	2013-02-27	Adrian Otto	Included resolutions for 34, 52
8	2013-03-13	Adrian Otto	Included resolution for 40
9	2013-04-29	Anish Karmarkar	Included resolutions for 50, 57
10	2013-05-01	Adrian Otto	Included resolution for 30, 60
11	2013-06-05	Adrian Otto	Included resolution for 58
12	2013-06-13	Gilbert Pilz, Tom Rutt	Included resolutions for issues 17, 67 and 68
13	2013-06-27	Gilbert Pilz	Included resolution for 3.
14	2013-07-01	Tom Rutt	Updated Figures for 3, kept revision marks from 12
15	2013-07-08	Gilbert Pilz	Included resolution for 4.
16	2013-07-10	Gilbert Pilz, Tom Rutt	Included resolution for 9. Miscellaneous, non-material changes and cleanups.
17	2013-07-11	Gilbert Pilz, Tom Rutt	Included resolution for 65. Miscellaneous, non-material changes and cleanups.
18	2013-07-19	Gilbert Pilz, Tom Rutt	Includes resolution for 55. Miscellaneous, non-material changes and cleanups.
19	2013-07-25	Gilbert Pilz	Included resolutions for 45, 56, 61, 75, 76, and 78. Miscellaneous, non-material changes and cleanups.
20	2013-07-26	Gilbert Pilz, Adrian Otto	Added names to Appendix A, “Acknowledgements”. Tagged normative statements that were missed in WD19. Homogenized captions for figures and tables. Homogenized and corrected cross-references. Miscellaneous editorial cleanups.
21	2013-08-14	Gilbert Pilz	Include resolution for 81. Fix the omission of the resolution for 60.

22	2013-08-21	Gilbert Pilz	Include resolution for 71. Fixed a couple of cross references.
23	2013-09-11	Gilbert Pilz	Fix bug where the resolution to issue 54 was not incorporated as the TC directed. Fix references to use the tag [RFC6902] to be consistent with other references to RFCs. Miscellaneous editorial cleanups.
24	2013-09-18	Gilbert Pilz	Include resolutions for 51 and 112. Miscellaneous editorial cleanups.
25	2013-10-10	Anish Karmarkar, Gilbert Pilz, Tom Rutt	Include resolutions for 18, 31, and 111. Numerous reformats to fix effect of Word bug. Miscellaneous editorial cleanups.
26	2013-10-23	Anish Karmarkar	Include resolutions for 89, 92, 93, 94, 99, 119, 120, 122, 123, 125, 127, 131, 136, 137, 138, and 145. Clean up of Appendix C and conformance item tags is not yet completely done. Update of figures not yet done.
27	2013-10-25	Anish Karmarkar	Added Tom's figures. Fixed appendix C. Added tags for new normative statements. Some ed. cleanup.
28	2013-10-31	Gilbert Pilz	Include resolutions for 85, 115, and 135. Miscellaneous editorial cleanups.
29	2013-11-06	Gilbert Pilz	Include resolutions for 83 and 149. Miscellaneous editorial cleanups.
30	2013-11-14	Gilbert Pilz	Include resolutions for 74, 80, and 86. Miscellaneous editorial cleanups.
31	2013-11-21	Gilbert Pilz, Tom Rutt	Include resolutions for 72, 73, 98, 105, 109, 110, 113, 116, 117, 118, and 144. Miscellaneous editorial cleanups.
32	2013-12-03	Adrian Otto, Anish Karmarkar, Tom Rutt, Gilbert Pilz	Include resolution for 151. Miscellaneous editorial cleanups.
33	2013-12-09	Adrian Otto	Editorial correction of 8 instances of RequirementType in section 4 to requirement_type in accordance with the resolution to issue 151.
34	2013-12-11	Gilbert Pilz	Include resolution for 157. Miscellaneous editorial cleanups.
35	2014-01-09	Gilbert Pilz	Include resolution for 44 and 150. Miscellaneous editorial cleanups.
36	2014-01-16	Gilbert Pilz	Include resolution for 155. Clean up broken references.
37	2014-01-23	Gilbert Pilz, Tom Rutt	Include resolutions for 156, 158, 160, and 161.
38	2014-01-29	Gilbert Pilz	Include resolution for 63. Add references to type definitions package and TA document on

			cover page. Update acknowledgements. Miscellaneous editorial fixes.
39	2014-02-03	Gilbert Pilz	Accepted all changes to clean up candidate for CD04.
40	2014-03-14	Gilbert Pilz	Include resolution for 164. Miscellaneous editorial fixes.
41	2014-05-07	Gilbert Pilz	Include resolution for 168.
42	2014-06-11	Gilbert Pilz	Include resolution for 169.
43	2014-07-07	Gilbert Pilz	Include resolution for 171.
44	2014-07-14	Gilbert Pilz	Include resolution for 172.
45	2014-07-23	Gilbert Pilz	Include resolution for 174.
46	2014-08-06	Gilbert Pilz, Tom Rutt	Include resolution for 173; update UML diagrams; miscellaneous editorial fixes.
47	2014-08-18	Gilbert Pilz, Tom Rutt	Editorial fixes; fix Figure 2-4.
48	2014-08-19	Gilbert Pilz	Accepted all changes to clean up candidate for CD05.