
# ONAP Common Versioning Strategy (CVS) for APIs

Dana Bobko, AT&T
dw2049@att.com

April 11, 2018

# Agenda

- Executive Summary
- Goal of the ONAP API Common Versioning Strategy (CVS)
- Implementing Semantic Versioning for APIs
- API URLs
- API Backwards Compatibility (BWC) Policy
- API Custom Headers and Behavior (4 slides)
- Swagger and Cataloging
- Swagger Guidance

# Executive Summary

At a high-level, these are the objectives of the ONAP Common Versioning Strategy (CVS):

- Implement **semantic versioning** (MAJOR.MINOR.PATCH) for APIs
- Implement pre-defined custom headers to communicate **MINOR, PATCH, and LATEST VERSION**
- Align **URLs to include only the MAJOR version**
- Adopt a **BWC policy** for APIs that is current **MAJOR release minus 1 year**
- If necessary, refactor APIs to support the concept of MINOR releases
- Documentation using **Swagger** using the guidance provided

# Goal of the ONAP API Common Versioning Strategy (CVS)

**The goal of the ONAP API CVS is to standardize API versioning, establish a backwards compatibility (BWC) policy, and expedite development/testing of ONAP APIs:**

- APIs will be “speaking the same language” in terms of how versions are characterized by employing the semantic versioning methodology.
- APIs can be released as MINOR versions, instead of a MAJOR version each release, which will expedite testing and minimize development introducing breaking changes.
- An established BWC policy limits how long previous versions need to be active/available.
- API clients can target specific versions and servers can evolve APIs without breaking existing clients (within the BWC timeframe).
- Lay the foundation for API cataloging, consistent Swagger documentation, and automated compatibility/dependency matrices.

# Implementing Semantic Versioning for APIs

- Utilizes the same semantic versioning methodology that is being used for ONAP's Release Versioning Strategy; therefore, development teams are familiar with the definition of the methodology.
- For a given a version number, MAJOR.MINOR.PATCH, increment the:
  - MAJOR position when you make any incompatible API or component change
  - MINOR position when you add functionality in a backwards-compatible manner
  - PATCH (or BUILD) position when you make invisible (and thus backwards-compatible) bug fixes
- Details of the specification can be found at http://semver.org/

# API URLs

- The URL shall only contain the **MAJOR** version number to minimize changes in the URL for MINOR and PATCH releases, assuming MINOR.PATCH releases are BWC.
- The structure of the URL shall be as follows, where version is placed after the "service" or API name:

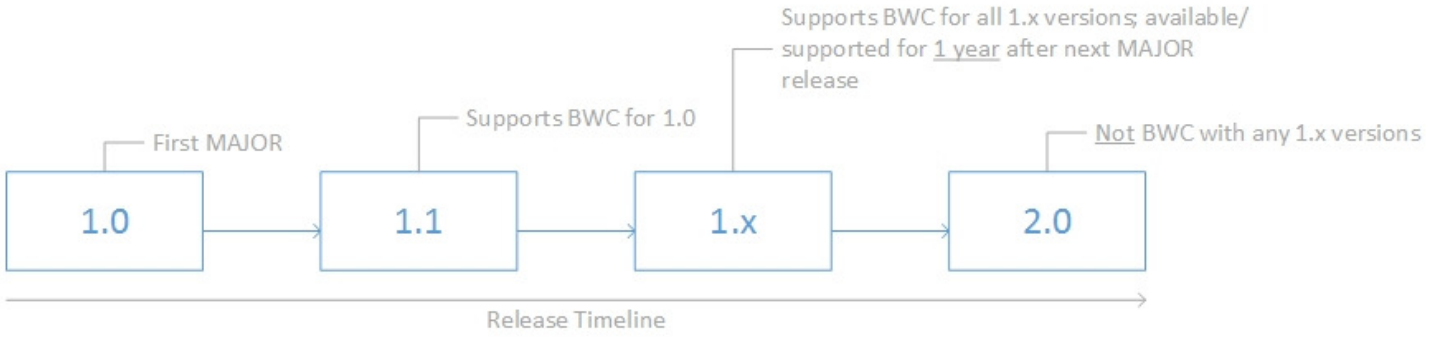
**…/root/{service or API name}/v{MAJOR version number}*/{resource path}**

**Example: {hostname}/aai/resource/v14/complexes**

*Note: "v" should precede the MAJOR version number in the URL. Service or API name is not the resource; it is intended to group of set of related resources.

# API BWC Policy

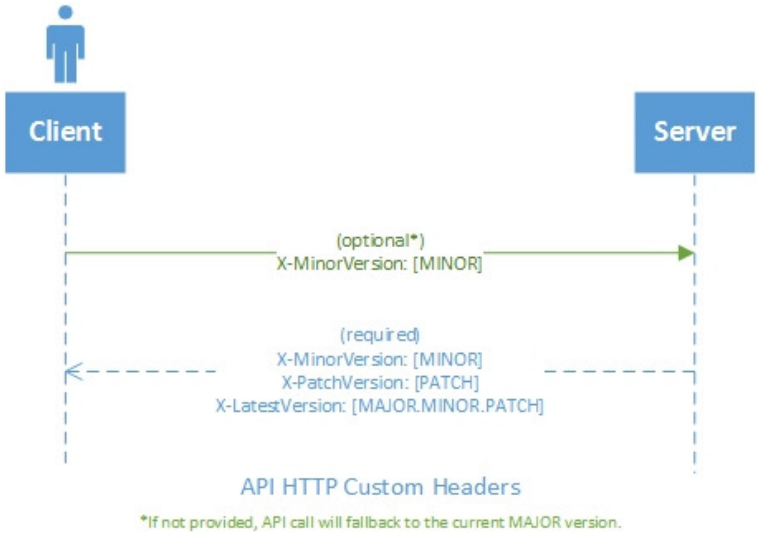
- API BWC shall be defined for **MAJOR** releases as the **current release - 1 year**. In other words, if an API is currently at 1.12 and a MAJOR release occurs to increment the version to 2.0, 1.12 (which is BWC for versions 1.0-1.11) must be functional/available for the period of 1 year after 2.0 is released.
- API owners shall ensure the previous **MAJOR** release remains available and functioning, in its last available production state, for the period of the BWC policy.
- **MINOR** releases shall be not time or release-based, as they are assumed to be BWC.
- API owners shall ensure no end-to-end services break with the deprecation of an API, due to the BWC Policy. End-to-end services includes, but is not limited to, VNFs, PNFs, Networks, Allotted Resources, etc.




API Evolution and BWC

# API Custom Headers and Behavior /1

- Three custom headers shall support versioning in APIs:
  - *X-MinorVersion*,
  - *X-PatchVersion*, and
  - *X-LatestVersion*.
- The request from the client shall not break, if the headers are absent in the request.
- The server shall employ logic to *fallback*[1] to the **MAJOR** version of the API, in the event that *X-MinorVersion* is not provided.
- Clients shall ignore additional values from the payload in the response, if provided.
  - It shall be explicitly specified in the interface contract that a server may increment a **MINOR** version and add additional fields.
  - The client shall be capable of handling this type of change in contract, if they remain on a previous **MINOR** version.

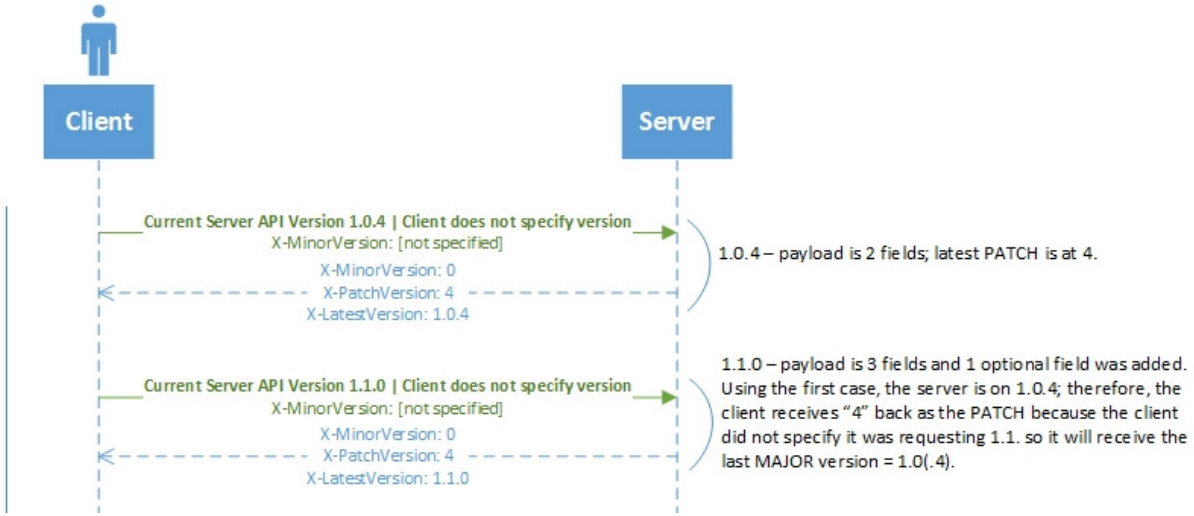


# API Custom Headers and Behavior /2

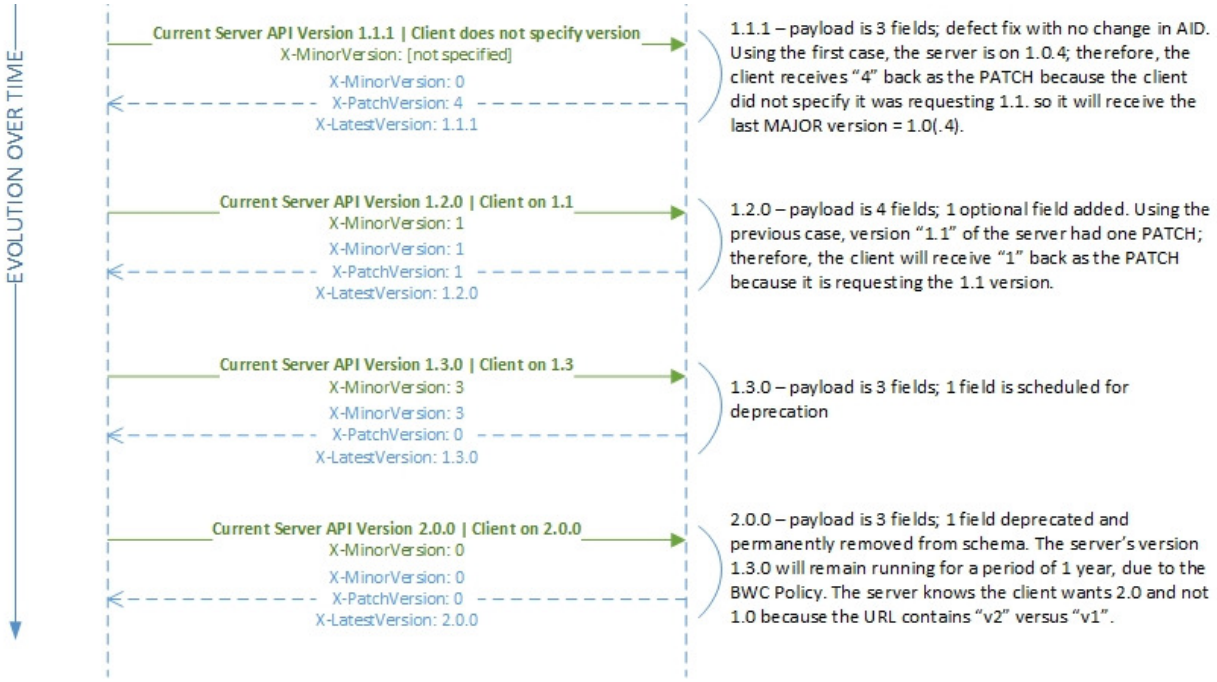
| Header Name | Specification |
|---|---|
| *X-MinorVersion* | • Used to request or communicate a MINOR version back from the client to the server, and from the server back to the client<br>• This will be the MINOR version requested by the client, or the MINOR version of the last MAJOR version (if not specified by the client on the request)<br>• Clarification: This will always be the MINOR version requested by the client - OR - if the client does not specify, it will default back to the very first MAJOR version of the server. For example, if the server is on 1.1 and the client does not send *X-MinorVersion*, the API call will default to 1.0 which makes the MINOR version = 0. This lets the client know they are not receiving the latest version, and they will know because *X-LatestVersion* will notify them.<br>• Contains a single position value (e.g. if the full version is 1.24.5, *X-MinorVersion* = "24")<br>• Is optional for the client on request; however, this header should be provided if the client needs to take advantage of MINOR incremented version functionality<br>• Is mandatory for the server on response |
| *X-PatchVersion* | • Used only to communicate a PATCH version in a response for troubleshooting purposes only, and will not be provided by the client on request<br>• This will be the latest PATCH version of the MINOR requested by the client, or the latest PATCH version of the MAJOR (if not specified by the client on the request)<br>• Clarification: This will always be the PATCH version the server is running.<br>• Contains a single position value (e.g. if the full version is 1.24.5, *X-PatchVersion* = "5")<br>• Is mandatory for the server on response |
| *X-LatestVersion* | • Used only to communicate an API's latest version<br>• Is mandatory for the server on response, and shall include the entire version of the API (e.g. if the full version is 1.24.5, *X-LatestVersion* = "1.24.5")<br>• Used in the response to inform clients that they are not using the latest version of the API |

# API Custom Headers and Behavior /3a

Comprehensive use cases should be provided on the Wiki so the expectation is set for when a MAJOR.MINOR.PATCH should be incremented.



# API Custom Headers and Behavior /3b



# Swagger and Cataloging

- ONAP is already using Swagger to document APIs.
- There is an opportunity to possibly leverage Swagger to drive cataloging efforts, through the use of tags in the metadata to indicate API versions (and Release versions APIs belong to).
- A **centralized catalog** could be leveraged to automate the decision-making for release dependencies and impacts.
  - Versions could be easily queried from a catalog/centralized location to communicate when a client will be impacted by API evolutionary changes.
  - The ability to generate comprehensive compatibility matrices would provide insight into how a particular change in an API would affect clients and other platform components.

# Swagger Guidance

- All components shall use Swagger 2.0. The specification may be found here. OpenAPI 3.0 is on the roadmap.
- Swagger files shall be generated at build time, and be placed in a centralized repository.
- Within the **Info Object**, the following annotations are **included** in the Swagger specification and shall be **required**, even if they are optional in the Swagger spec:
  **title**
  **description**
  **version** - fully-qualified version number of the Swagger file (ex: 1.4.18)
- Within the **Info Object**, the following are **extensions** of the Swagger specification and shall be **required**:
  **x-planned-retirement-date** - use YYMM; string type. *This is the date that the API shall be deprecated, based on the BWC Policy. APIs may be active after their retirement date, but are not guaranteed to remain in production. An API retirement may be pushed out to accommodate BWC for clients.*
  **x-component** - SDC, SO, etc., or the mS name; string type. *This is the component that owns the API.*
- Under the Path, the following shall be **required**:
  **x-interface info** - this contains two attributes:
    - **api-version** - fully-qualified version number of the API (ex: 1.3.6); string type. *This is the version of the API. This differs from the version in the second bullet above.*
    - **last-mod-release** - in ONAP, use release name. *This is the last release that the API was modified in.*
- Within the **Path Item Object**, the following are **included** in the Swagger specification and shall be **required**:
  **description** - string
  **parameters**
    - **required** - boolean
    - **type** – string