9 API versioning

This section describes the API versioning interface available to all privileged code.

9.1 API call

9.1.1 api_set_version

```
trap#              CORE_TRAP
function#          API_SET_VERSION
arg0               api_group
arg1               major_number
arg2               req_minor_number

ret0               status
ret1               act_minor_number
```

The API service enables a guest to request and check for a version of the Hypervisor APIs with which it may be compatible. It uses its own trap number to ensure consistency between future versions of the virtual machine environment. API services are grouped into sets that are specified by the argument *api_group*, (defined in the table below). For the specified group the guest's requested API major version number is given by the argument *major_number* and a requested API minor version number is given by the argument *req_minor_number*.

If the *major_number* is supported, the actual minor version implemented by the Hypervisor is returned in ret1 (*act_minor_number*). Note that the actual minor version number may be less than, equal to, or greater than the requested minor version number. (See Notes, below). If the returned *act_minor_number* is greater than the *req_minor_number* then the APIs enabled by the Hypervisor for *api_group* will be compatible with *req_minor_number*.

If the *major_number* is not supported, the Hypervisor returns an error code in ret0, and ret1 is undefined. (See Errors, below.)

If the *major_number* requested is zero, the version of the api_group selected is requested to return to the initial un-set (disabled) state. If the call succeeds it will return with EOK in *status*, and *zero in act_minor_number*.

The version number of a specified API group may be set at any time with this API service, however;

1. The act of selecting an API version for an api_group, or requesting that the group return to being un-set (major_number=0), does not reset any previous state associated with services within a group - unless specified explicitly for that group associated state after a api_set_version call is undefined.

2. Any API calls belonging to the same *api_group* being made concurrently with this api_set_version service will have undefined results.

3. Calls to APIs made concurrently with api_set_version that are not in api_group proceed as normally defined.

4. Simultaneous calls to api_set_version using the same *api_group*, may succeed but leave the *api_group* in an undefined state.

5. Simultaneous calls to api_set_version and api_get_version using the same *api_group* have undefined results for api_get_version.

6. api_set_version does not affect the CORE_TRAP API calls - these remain unaffected and may be called at any time.

The API groups are defined in Appendix B: Number Registry (on page 123) together with the approved version numbers for each of the API services defined in this specification.

Programming note: Each API group is treated independently of the others from a versioning perspective, so one group can have its version negotiated while APIs from other groups are actively being used. However, a guest operating system should take care to ensure that while a api_set_version is in progress, no APIs from the same api_group are used, and no other calls to api_set_version or api_get_version are made using the same api_group.

9.1.1.1 Errors

EINVAL	If api_group field is unknown to this hypervisor, (this error takes precedent over ENOTSUPPORTED)
ENOTSUPPORTED	If major number for that api_group is not supported
EOK	If api_group and major_number match, or major_number is zero
EWOULDBLOCK	Operation would block
EBUSY	The api_group is currently in use, and the requested version would leave the virtual machine in an illegal state

9.1.1.2 Usage Notes:

This API uses its own trap number, not for performance reasons, but to ensure its constancy even in the face of new API major versions.

Regardless of version number, the Hypervisor core APIs (CORE_TRAP) defined above enables any guest to print a message and cleanly exit its virtual machine environment in the event it is unsuccessful in negotiating an API version with which to communicate with other hypervisor functions.

The following informative text is provided as a guide to assist the reader in understanding the hypervisor versioning API.

API functions and returned data structures are categorized into specific groups. Each group represents an area of hypervisor functionality that may change independently of the others, and therefore may be versioned independently.

For each API group there is a major and a minor version number. Differences in the major version number indicate incompatible changes. Differences in the minor number indicate compatible changes, such that a higher version number espoused by the hypervisor will be compatible with a lower minor number requested by a guest. If the api_group is not supported the api_version function will return EINVAL. If the major version number for a valid api_group is not supported the api_version function will return ENOTSUPPORTED.

The handling of an unsupported API version is purely guest policy, however a guest may freely attempt a different major version if it is capable of driving that alternate interface. The suggested minimal behaviour is to print a warning message and exit the virtual machine.

By way of example consider a guest that requests minor version 'Requested', and this API may return minor version 'Actual' for a given *major_number* and *api_group*.

If Requested == Actual, then the requested minor version is available.

If Actual < Requested, the guest must be able to determine if the interface with minor version Actual offers the required services and proceed accordingly. (This is a guest policy issue.)

If Actual > Requested, then the guest may assume it can operate compatibly with version Requested. Minor version number increments are defined to be compatible with the preceeding version, so in general the guest may accept Actual when Actual > Requested. In this case, the guest may want to print a warning, but that is up to the policy of the guest.

Alternatively in the event that Actual>Requested, the hypervisor may elect to emulate version Requested, thus returning Requested.

For situations such as the co-residence of OBP with Solaris, or mutliple Solaris modules using the same API group, a layered software approach must be taken for version negotiation.

For example, it is recommended that OpenBoot intially negotiate to the lowest version number supported for the firmware consolidation for api groups it intends to use. A subsequent guest operating system may then negotiate versions up for each api group by calling though OpenBoot's CIF interface. Using the CIF interface means OpenBoot will be aware of the version negotiation and can adapt itself accordingly to new api versions, or simply veto requested versions it cannot compatibly upgrade to. If a guest negotiates versions directly with the hypervisor bypassing the CIF, the guest is responsible for dismissing OpenBoot and providing OpenBoot services for itself.

9.1.2 api_get_version

```
trap#               CORE_TRAP
function#           API_GET_VERSION
arg0                api_group

ret0                status
ret1                major_number
ret2                minor_number
```

This service is used to determine the major and minor number of the most recently successfully set API version for the specified group (see section 9.1.1). In the event that no API version has been successfully set the call returns the error code EINVAL and ret1 and ret2 are set to 0.

9.1.2.1 Errors

```
EINVAL              - No API version yet successfully set
```