3 Common definitions

3.1 Trap numbers

The following are the *sw_trap_numbers* encoded in the Tcc instruction that enters the hypervisor:

```
FAST_TRAP                   0x80
MMU_MAP_ADDR                0x83
MMU_UNMAP_ADDR              0x84
TTRACE_ADDENTRY             0x85
CORE_TRAP                   0xff
```

Unless assigned to technology or platform specific APIs all other trap numbers (0x86 to 0xfe inclusive) result in EBADTRAP being returned in %o0 as described in section 2.3.

3.2 Function numbers for FAST_TRAP

Function numbers for fast-traps are provided in %o5 as a 64-bit value. The following are the function numbers defined for the core API set:

```
MACH_EXIT                   0x00
MACH_DESC                   0x01
MACH_SIR                    0x02
MACH_SET_SOFT_STATE         0x03
MACH_GET_SOFT_STATE         0x04

CPU_START                   0x10
CPU_STOP                    0x11
CPU_YIELD                   0x12
CPU_QCONF                   0x14
CPU_QINFO                   0x15
CPU_MYID                    0x16
CPU_STATE                   0x17
CPU_SET_RTBA                0x18
CPU_GET_RTBA                0x19

MMU_TSB_CTX0                0x20
MMU_TSB_CTXNON0             0x21
MMU_DEMAP_PAGE              0x22
MMU_DEMAP_CTX               0x23
MMU_DEMAP_ALL               0x24
MMU_MAP_PERM_ADDR           0x25
MMU_FAULT_AREA_CONF         0x26
MMU_ENABLE                  0x27
MMU_UNMAP_PERM_ADDR         0x28
MMU_TSB_CTX0_INFO           0x29
MMU_TSB_CTXNON0_INFO        0x2a
MMU_FAULT_AREA_INFO         0x2b

MEM_SCRUB                   0x31
MEM_SYNC                    0x32

CPU_MONDO_SEND              0x42

TOD_GET                     0x50
TOD_SET                     0x51

CONS_GETCHAR                0x60
CONS_PUTCHAR                0x61

TTRACE_BUF_CONF             0x90
TTRACE_BUF_INFO             0x91
```

```
TTRACE_ENABLE             0x92
TTRACE_FREEZE             0x93
DUMP_BUF_UPDATE           0x94
DUMP_BUF_INFO             0x95

INTR_DEVINO2SYSINO        0xa0
INTR_GETENABLED           0xa1
INTR_SETENABLED           0xa2
INTR_GETSTATE             0xa3
INTR_SETSTATE             0xa4
INTR_GETTARGET            0xa5
INTR_SETTARGET            0xa6
```

Unless assigned to technology specific or platform specific APIs all other function numbers used for fast-traps result in EBADTRAP being returned in %o0 as described in section 2.3.

3.3 Function numbers for CORE_TRAPs

CORE_TRAP APIs follow the same calling conventions as FAST_TRAP API services. The following are the function numbers defined for the core API set:

```
API_SET_VERSION           0x00
API_PUTCHAR               0x01
API_EXIT                  0x02
API_GET_VERSION           0x03
```

CORE_TRAP function numbers are defined as followed:

API_VERSION is defined in section 5.

API_PUTCHAR is an alias for FAST_TRAP function CONS_PUTCHAR.

API_EXIT is an alias for FAST_TRAP function MACH_EXIT.

API_GET_VERSION is defined in section 5.

3.4 Error codes

When a hypervisor API returns, unless explicitly described by the API service, the 64-bit value in %o0 will be one of the following error identification values.

```
EOK               0     Successful return
ENOCPU            1     Invalid CPU id
ENORADDR          2     Invalid real address
ENOINTR           3     Invalid interrupt id
EBADPGSZ          4     Invalid pagesize encoding
EBADTSB           5     Invalid TSB description
EINVAL            6     Invalid argument
EBADTRAP          7     Invalid function number
EBADALIGN         8     Invalid address alignment
EWOULDBLOCK       9     Cannot complete operation without blocking
ENOACCESS         10    No access to specified resource
EIO               11    I/O Error
ECPUERROR         12    CPU is in error state
ENOTSUPPORTED     13    Function not supported
ENOMAP            14    No mapping found
ETOOMANY          15    Too many items specified / limit reached
```

3.5 Guest states

As defined by the Sun4v Architecture Specificiation each virtual CPU can have one of three different states:

5 API versioning

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

5.1 API call

5.1.1 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_numbe***r**.

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

The API groups are defined below together with their version numbers compliant with this specification.

Group	Number (api_group)	Group Definition	Version for this specification
Common	0x0	sun4v version	1-0
	0x1	API version	1-0
	0x2	MMU Fault status area version	1-0
Technology	0x100	PCI	1-0
	0x101	Logical Domain Channels	1-0

5.1.1.1 Errors

```
EINVAL            If api_group field is invalid or unsupported
ENOTSUPPORTED     If major number for that api_group is not
                  supported
EOK               If api_group and major number match
```

5.1.1.2 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 guest that requests minor version X, and this API may return minor version Y for a given *major_number* and *api_group*.

If X = Y, then the requested minor version is available.

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

If Y > X, then the guest may assume it can operate compatibly with version Y. Minor version number increments are defined to be compatible with the preceeding version, so in general the guest may accept Y when Y > X. 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 X<Y, the hypervisor may elect to emulate version X, thus returning X.