

1 Introduction

This document describes the mechanism by which memory from one logical domain may be exported for access by another logical domain. This facility enables shared memory to be utilized for such functionality as virtual device services. Using the interfaces described herein, one logical domain may export a number of its own memory pages across a logical domain channel for access and use by the logical domain at the other end of the channel. The mechanism is intended to be directly analogous to the way a domain would export pages of its memory for access by I/O devices on the other side of an I/O bridge (I/O MMU).

1.1 Map table

The principle means by which a domain may export its local memory across a domain channel is through the use of an export map table that the guest defines within its own local memory - much like a TSB is used to define local virtual memory mappings.

The recipient domain at the other end of the logical channel may make use of the exported memory either by using a hypervisor API call to copy data into or out of its local memory, or by using a hypervisor API call to explicitly map the remote exported memory into its real address space for access.

The real address space of each domain's virtual machine is independent of all the others. Therefore to coordinate references to exported memory between domains, cookies are used to refer to entries within the exporter's map table.

Consider a domain ("domain X") that wishes to export a page of memory to another domain ("Y").

For this to be possible a domain channel must connect X to Y. Let us assume that such a channel has been created by the domain manager.

In order to export any memory across this domain channel, domain X must allocate an export map table from its local memory, and assign that map table to its local channel endpoint.

The assigned map table may be used to export multiple pages, which remain exported until explicitly removed from the map table, or the table itself is un-assigned from the channel endpoint.

The map table must be a power of number of entries in size, and must be aligned in memory on a real address boundary equal to its size in bytes.

Hypervisor API calls are provided to assign a map table to a channel endpoint, un-assign the table, and to get the table info. A map table may not be assigned to more than one channel endpoint at a time.

1.1.1 Map table cookies

For the recipient domain 'Y' to be able to refer to exported memory, it must use a 'cookie' that describes the memory that domain 'X' is exporting. This cookie may be considered a form of address for the remote memory, much like a dma-cookie is used for dma operations by an IO device.

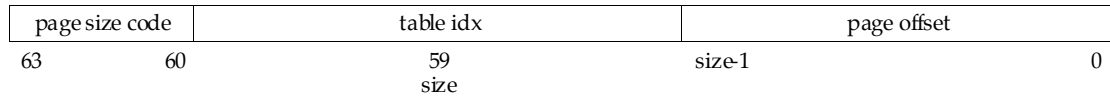
The export cookie is created by the exporting domain 'X' and it contains two essential pieces of information - the size of the exported page mapping, and the index in the exporter's map table of that mapping. A cookie may also contain offset information so as to identify data located within the memory page defined by a mapping.

A cookie only has meaning within the context of the domain channel its associated map table is bound to. Thus if a map table is assigned to a channel endpoint in domain X, then domain Y must also identify its local endpoint when using the cookie. In this way the hypervisor is not responsible for creating or tracking or transferring cookies between domains.

A cookie is created by the exporting domain, and can be communicated by any means to the

importing domain - for example by message over the same domain channel. When a cookie is used (for example with a ldc_copy operation), the associated local channel endpoint enables the hypervisor to determine the remote channel endpoint and the therefore the remote (exporting) domain and the export table itself. The cookie may then be used to locate the entry in the export map table that defines the memory being exported.

Cookies created by an exporting domain have the following format:



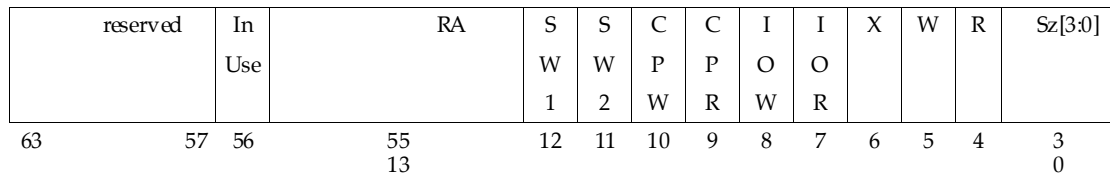
The upper four bits of the cookie identify the page size of the exported page, and use the same page size encodings as the basic sun4v TTE format (defined in the Hypervisor API specification – FWARC/2005/116).

The remainder of a cookie consists of an offset within the specified exported page and an index to the entry within the exporting domain's map table that identifies the actual exported page. The offset field ranges from bit zero, to the number of offset bits relevant for the cookie's page size. The index field starts at the first bit for the page frame number and continues to bit 59. For example, for an 8K page; the page size field (bits 60 to 63) is zero, the page offset is in bits 0 through 12, and the table index is specified in bits 13 through 59.

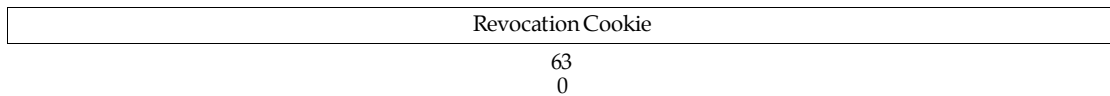
This compressed cookie format enables a page size, index value and page offset to be transferred in one single 64bit value that may in effect be treated as an address itself. Basic arithmetic may be applied to the offset field, which if it overflows will automatically adjust the table index field. In this way a large number of sequential map table entries of the same page size can be described by a single cookie value.

1.1.2 Map table entries

Word 0: map entry (bytes 0 through 7)



Word 1: revocation entry (bytes 8 through 15)



For the export map table, each entry consists of a two 64-bit words illustrated in the figure above.

The map entry (word 0) bit fields are defined as follows:

Bit Field	Mnemonic	Meaning
63 – 57	reserved	Must be written as zero
56	In Use	This bit is set by the hypervisor if a map-table entry is still in use by the importing domain. It is also cleared by the hypervisor if the entry is no longer mapped by the importing domain.
55-13	RA	Real address bits 55 to 13. For page sizes larger than 8KB, the low order address bits below the page size must be set to zero
12	SW1	This bit is available for use by software.
11	SW2	This bit is available for use by software.

10	CPW	Copy writeable; if set to 1 the hypervisor ldc_copy API may be used by the importing domain to write to this exported page.
9	CPR	Copy readable; if set to 1 the hypervisor ldc_copy API may be used by the importing domain to read from this exported page.
8	IOW	I/O writeable; if set to 1 this exported page may be mapped by an IOMMU for writing by an I/O DMA operation.
7	IOR	I/O readable; if set to 1 this exported page may be mapped by an IOMMU for reading by an I/O DMA operation.
6	X	eXecute; if set to 1 instructions may be fetched and executed from this page by the importing domain
5	W	Writeable; if set to 1 this page may be mapped and written to as shared memory by the importing domain
4	R	Readable; if set to 1 this page may be mapped and read from as shared memory by the importing domain
3-0	Sz	Size: page size 0 = 8KB, 1=64KB, 2=512KB, 3=4MB, 4=32MB, 5=256MB, 6=2GB, 7=16GB Sizes 8 through 15 are reserved.

The permissions bits (bits 4 through 10) indicate the access permissions granted by the exporting domain to the importer of the page described by the specific map table entry. If no access permissions are granted, (bits 4 through 10 are all zero), the map table entry is considered invalid.

Note: It is recommended that invalid map table entries have the entire 64bit word set to zero.

75 Map table entries must not contain overlapping or identical real address ranges - do so yields undefined results for both exporter and importer - without guarantee that the exporter will be able to revoke access permissions to the exported page.

1.2 Copying in and out of a peer's exported memory

80 Once a LDC peer has provided access to memory pages via its map table, a guest operating system can request the hypervisor to copy data into and out of those pages by simply presenting cookies provided by the peer with the ldc_copy hypervisor API call.

Each time the call is made the hypervisor validates the presented cookie together with the access permission provided in the exporter's map table to determine whether the copy should indeed be allowed.

85 This is the simplest mechanism by which data may be transferred in bulk between guest operating systems.

1.3 Mapping page use and restrictions

For a guest to use memory exported by one of its LDC peers, it must ask the hypervisor to provide access to the exported page. This is achieved using the ldc_mapin hypervisor API call.

90 The map-in call returns a real-address of where the imported shared memory page was mapped within the importing guests virtual machine real address space. Shared memory is un-imported using the ldc_unmap API call by passing the same real-address that was returned from the ldc_mapin API call.

95 As part of the importer's real address space, the imported shared memory page may be used for virtual memory mappings and IO MMU mappings with the same mechanisms as its own memory pages. However, imported shared-memory pages are not generally accessible like normal memory pages, and the hypervisor enforces a number of restrictions upon their use:

100 The guest exporting a shared memory page may only allow certain types of access to that page (for example for reading only). For example, attempts to map a page without read or write permission for load or store instructions will fail (or in the case of TSB use generate a data or

instruction access exception trap for an invalid real address).

In addition to the restrictions required by the exporting guest, the hypervisor itself requires that importing pages are not aliased either by virtual memory mappings, or IO MMU mappings. Virtual memory mappings are allowed only for context 0 but are available to all virtual CPUs.

105 Imported shared memory must be unmapped and re-mapped in before a new virtual or IOMMU address may be assigned - even if the old virtual address has been de-mapped with the appropriate demap API call.

1.4 Mapping revocation

110 When a guest wishes to discontinue the export of a page to its LDC peer, it can do so by simply denying further access by disabling the access permissions in the map entry word in the corresponding map table entry. (It is recommended that an entry be disabled/invalidated) by writing the value 0 to the whole map entry word (word 0).

Denying future accesses does not automatically revoke existing page mappings to which the LDC peer may have access.

115 Well behaved peers sharing exported memory are recommended to use a communication protocol to determine when exported memory pages are available or no longer in use by a peer. It is anticipated, therefore, that only in extra-ordinary circumstances will a guest that exports memory need to forcibly deny ("revoke") access to a previously exported memory page.

120 To avoid the cost of an export revocation for well behaved peers, the hypervisor provides an indication that an exported page is actually still in use by a peer in the form of a revocation cookie in the second word of the map-table entry for the exported page. This revocation cookie word must be initialized to zero when a page is exported, and will be over-written by the hypervisor with a revocation cookie while the exported page is actually in use by the peer guest.

125 When a page is no longer to be exported, the export mapping permissions should be removed after which the revocation cookie word can be examined to see if the page is actually still in use by the peer guest. A revocation cookie value of zero indicates the page is not in use - at which point the map table entry may be re-used for exporting other pages.

130 A non-zero value for the revocation cookie indicates that the previously exported page is still in use by the peer guest. It then becomes a matter of policy for the exporter as to whether it wishes to forcibly revoke the access permissions for the importer, or simply wait for the importer to clean-up itself.

To forcibly revoke access permission for the peer guest, the exporting guest simply uses the ldc_revoke API call with the LDC cookie for the exported page, and the revocation cookie provided in the export map table.

135 Removing individual permissions for exported pages must be done by unmapping or revoking access to the exported page first, then re-exporting it with the new permissions required.

Forcibly revoking access to an exported page, can have catastrophic consequences for the importer - including failed memory accesses or failed device DMA transactions. Therefore, the exporter should avoid revocation as far as possible.

140 Exit of the exporting guest will cause the hypervisor to automatically forcibly revoke exported page mappings.

An importer of shared memory pages that is intended to be robust should be designed to shield itself against exported mappings being forcibly revoked at any time either by the exporter or automatically by the hypervisor if the exporter exits.

145 Importers wishing to avoid these issues may always use the ldc_copy capability to move data.

2 API Calls

The following fast trap function numbers are defined by this document;

Version 1.0 APIs

	LDC_SET_MAP_TABLE	0xea
150	LDC_GET_MAP_TABLE	0xeb
	LDC_COPY	0xec

Version 1.1 APIs

	LDC_MAPIN	0xed
	LDC_UNMAP	0xee
155	LDC_REVOKE	0xef

These LDC shared memory API calls are assigned API group number 0x101 for the purposes of the versioning API. The *ldc_set_map_table*, *ldc_get_map_table* and *ldc_copy* APIs are usable when version 1.0 of the API group is negotiated. The *ldc_mapin*, *ldc_unmap* and *ldc_revoke* APIs are available in addition to the 1.0 APIs when version 1.1 of the API group is negotiated.

2.1 *ldc_set_map_table*

	trap#	FAST_TRAP
	function#	LDC_SET_MAP_TABLE
	arg0	channel
	arg1	base_ra
165	arg2	nentries
	ret0	status

This API service enables a guest to declare an export map table and bind that map table to the specified logical domain *channel*.

The map table must consist of a power of two number of entries specified by the *nentries* argument. The minimum number of entries is 2. The export map table base real address is specified in *base_ra* and must be aligned to the same boundary as the overall size of the table in bytes (*nentries**8).

Specifying zero (0) for *nentries* un-binds any map table previously bound to the domain *channel*. If *nentries* is zero, *base_ra* is ignored. Unbinding a map table does not automatically revoke exported pages, any pages still in use by an importing domain may remain accessible by that domain for an indeterminate period of time, or until the exporting domain exits.

2.1.1 Errors

	ENORADDR	Invalid <i>base_ra</i> or real address range for the map table
180	EBADALIGN	map table <i>base_ra</i> is not correctly aligned for the size of the table
	EINVAL	<i>nentries</i> is invalid, or specified domain <i>channel</i> does not support a shared memory interface.
	ECHANNEL	Illegal domain <i>channel</i>
185	EWOULDBLOCK	Operation would block

2.2 ldc_get_map_table

	trap#	FAST_TRAP
	function#	LDC_GET_MAP_TABLE
190	arg0	channel
	ret0	status
	ret1	base_ra
	ret2	nentries

195 Retrieves the current map table configuration associated with the given domain channel.
 If no map table is configured, both *base_ra* and *nentries* are returned as zero.

2.2.1 Errors

ECHANNEL	Illegal domain channel
EWOULDBLOCK	Operation would block

200 **2.3 ldc_copy**

	trap#	FAST_TRAP
	function#	LDC_COPY
	arg0	channel
	arg1	flags
205	arg2	cookie
	arg3	raddr
	arg4	length
	ret0	status
210	ret1	ret_length

This API service copies data into or out of a local memory region from or to the logical domain at the other end of the specified domain channel.

215 The local memory buffer to be used is a contiguous real address buffer starting at *raddr*, and of size *length*. Both *raddr* and *length* must be aligned to 8 byte boundaries. The exported page to be accessed by the copy operation is identified by *cookie*.

A copy in or copy out operation is specified by the *flags* argument, for which the following values apply:

	LDC_COPY_IN	0x0	Copy from remote exporting domain into buffer of local domain
220	LDC_COPY_OUT	0x1	Copy from buffer of local domain into page exported by remote domain

All other values for flags are illegal.

In the event of success, the return status is EOK, and *ret_length* contains the actual number of bytes copied. In this event $0 \leq \text{ret_length} \leq \text{length}$.

225	2.3.1 Errors		
		ECHANNEL	Illegal domain channel
		EINVAL	Illegal flags value
		EBADALIGN	Badly aligned raddr, length or cookie
		ENORADDR	Bad real address range for local buffer
230		ENOMAP	Cookie refers to invalid map table entry on exporting side
		ENOACCESS	Requested copy operation is not permitted by exporter's map table entry
235		EBADPGSZ	Page size of cookie does not match page size specified in map table entry
		EWOULDBLOCK	Operation would block

2.4 ldc_mapin

	trap#	FAST_TRAP
	function#	LDC_MAPIN
240	arg0	channel
	arg1	cookie
	ret0	status
	ret1	raddr
245	ret2	perms

250 This API service attempts to map into the local guest's real address space the page identified by the shared memory *cookie*. Upon success the service returns the real address the page was mapped at in *raddr*, and the access permissions granted to that page by the exporter for cpu and IO access in *perms*. Bit 0 in the *perms* value corresponds to bit 4 in the export table entry - namely CPU read permission. Bit 1 in *perms* corresponds to bit 5 in the export map table entry, and so on. Bits 5 through 63 of *perms* are undefined and should be ignored.

2.4.1 Errors

	ECHANNEL	Illegal domain channel
	EINVAL	Illegal flags value
255	EBADALIGN	Badly aligned cookie
	ENOMAP	Cookie refers to invalid map table entry on exporting side
	ENOACCESS	Requested map operation is not permitted by exporter's map table entry
260	EBADPGSZ	Page size of cookie does not match page size specified in map table entry
	ETOOMANY	Too many mapins already exist
	EWOULDBLOCK	Operation would block

2.5 ldc_unmap

265	trap#	FAST_TRAP
-----	-------	-----------

function#	LDC_UNMAP
arg0	raddr
ret0	status

270 This API service attempts to unmap from the local guest's real address space the imported page mapped at the real address *raddr*.

This API may fail if the guest has not already removed any virtual or IOMMU mappings associated with this page.

2.5.1 Errors

275	ENORADDR	Illegal raddr value
	EBADALIGN	Badly aligned raddr
	ENOMAP	raddr refers to a non-existent imported page
	EWOULDBLOCK	Operation would block

2.6 ldc_revoke

trap#	FAST_TRAP	
function#	LDC_REVOKE	
arg0	channel	
arg1	cookie	
285	arg2	revoke_cookie
ret0	status	

290 This API service attempts to forcibly unmap from a remote guest's real address space a page previously exported by the local guest. The remote guest is the peer on the other end of the LDC channel specified by *channel*. The *cookie* is the cookie originally passed to that remote guest, the *revoke_cookie* is the revocation cookie supplied by the hypervisor to assist this API call. This unmapping mechanism also forcibly unmaps any virtual or IOMMU mappings that the remote guest may be using corresponding to this exported page.

295 *Note: As an optimization, in the event that this API fails with EWOULDBLOCK, the caller should re-read the revocation cookie from the corresponding export table entry; in the event that the revocation cookie has been set to zero, this API should no longer be necessary.*

2.6.1 Errors

300	ECHANNEL	Illegal channel value
	EINVAL	Illegal revoke cookie
	EBADALIGN	Badly aligned cookie
	EWOULDBLOCK	Operation would block