Genera 8.0 ECO #1 Notes

Overview of Genera 8.0 ECO #1

This is the first ECO to Genera 8.0 and Genera 8.0XL. The purpose of this ECO is to make the following improvements to Genera:

- Provide patches which lay the groundwork for supporting CLIM.
- Improve the integration of CLOS with Flavors.
- Include patches which support the Symbolics XL1200. These changes affect the VME interface on the XL1200 and the XL400. The revised VME documentation is included in this document. See the section "Genera 8.0 XL Documentation Update".
- Provide support for those UX400S customers who want to upgrade to SunOS 4.1. The UNIX software provided with ECO #1 supports SunOS 4.1 and does *not* run with SunOS 4.0. If you have a UX400S and are still running SunOS 4.0 on the Sun, do not load the UNIX software provided with this ECO on your UX400S. Genera and the UNIX software are completely intercompatible between 8.0 and 8.0 ECO #1: you may run Genera 8.0 ECO#1 with the Genera 8.0 UNIX software.
- Fix the problem with MacIvory Ethernet packet transmission that was corrupting packets, causing problems copying worlds and loading files via CHAOSNET.
- Fix the problem with 3600-family cart tape that caused tapes which were not rewound to give hard tape errors.
- Speed up access to the Macintosh file system from MacIvory.
- Fix an IFEP problem with the XL1200 in Genera 8.0XL where certain errors that should have printed in the cold load stream were causing the processor to reset, forcing a warm boot.
- Fix a bug in Statice that could cause databases to become corrupted.
- Fix some other significant bugs in Genera 8.0.

Installing Genera 8.0 ECO #1

Overview of the Genera 8.0 ECO #1 Installation

Genera 8.0 ECO#1 is being distributed as an incremental world. New Flod files and layered product patches are being distributed on a distribution tape. You need to

1. Restore the Genera 8.0.1 world load from tape.

2. Restore the FEP Flods and layered product patches from the distribution tape.

3. Copy new Flod files to all machines.

4. Update your boot file to boot the new world.

5. Reset the FEP and boot the new world.

The new Flod files for Ivory machines are a new FEP version, so for Ivory machines it is also necessary to update your `Hello.boot` file.

Tapes for Genera 8.0 ECO #1

Genera 8.0 ECO #1 consists of:

1. 3 World tapes:

    a. Symbolics Genera 8.0.1 Distribution World for 3600-family; one QIC-11 tape, IFS Format, containing the incremental ECO world (Genera-8-0-1-from-Genera-8-0.load).

    b. Symbolics Genera 8.0.1 Distribution World for UX400S; one QIC-11 tape containing the incremental ECO world (Genera-8-0-1-from-Genera-8-0.ilod) and the new Network incremental (Network-from-Genera-8-0-1.ilod).

    c. Symbolics Genera 8.0.1 Distribution World for XL400, XL1200, and MacIvory; one QIC-100 tape containing the incremental ECO world (Genera-8-0-1-from-Genera-8-0.ilod).

2. 2 Patch tapes: Symbolics Patches for Genera 8.0.1 Layered Products; one QIC-11 or one QIC-100 tape, Distribution Format, containing new NFEP/IFEP Flod files and layered product patches

3. 1 life support tape: UX400S life support; one QIC-11 tape, extract_unbundled format, UX400S UNIX software for Genera 8.0.1, SunOS 4.1

4. 3 MacIvory support floppies:

    a. MacIvory System

b. MacIvory Applications

c. MacIvory Development

You should receive only those tape formats that correspond to the Symbolics machines at your site.

Restoring the ECO Incremental World

This procedure should be performed once on each machine type at your site, that is once on a 3600-family machine, once on a UX400S, and once on an XL1200, XL400, or MacIvory. The new world can then be copied to all the other machines of the appropriate type at your site.

1. Insert the appropriate *Symbolics Genera 8.0.1 Distribution World* tape into the tape drive of a Symbolics machine at your site. Type the CP command Select Activity FEP-Tape. In the FEP-Tape activity, give the command:

   ```
   Read Tape
   ```

2. You are asked if you want to restore the world load file. Answer Y (for Yes) and accept the default pathname, or specify a FEP pathname that indicates a world load, such as:

   ```
   FEP0:>Genera-8-0-1-from-Genera-8-0.load
   ```

   or

   ```
   FEP0:>Genera-8-0-1-from-Genera-8-0.ilod
   ```

   The FEP-Tape loading program then loads the world from the tape.

   The UX400S tape contains a second world load (`Network-from-Genera-8-0-1.ilod`). This world should also be restored at this time.

3. Update the `boot.boot` file on your machine to load the new world. Your updated `boot.boot` file should look similar to this:

   ```
   Load World FEP0:>Genera-8-0-1-from-Genera-8-0.ilod
   Start
   ```

   See the section "Contents of Hello.Boot Files and Boot.Boot Files".

Restoring the Distribution Tape for Layered Product Patches

This tape only needs to be restored once, to the File Server at your site.

1. Insert the appropriate *Symbolics Patches for Genera 8.0.1 Layered Products* distribution tape into the tape drive of the machine.

2. Type the CP command

   ```
   Restore Distribution :Menu Yes
   ```

3. Click on [Initialize Restoration] and use the mouse to deselect any systems you do not want to load.

   The patch tape contains patches to the following systems:

   - All customers should restore patches to these loadable systems:

     ° Conversion Tools
     ° Hypercard/MacIvory
     ° Mailer
     ° Metering Substrate
     ° Metering
     ° Print

   - All 3600-family customers should restore these:

     ° NFEP Overlays

   - All Ivory customers should restore these:

     ° IFEP Distribution

   - Only UX customers need to restore these:

     ° Additional Files

   - 3600-family customers and UX Delivery world customers may choose to restore these:

     ° RPC Development
     ° MacIvory Support

   - Only Delivery world customers need to restore these:

     ° Zwei
     ° Serial
     ° Hardcopy

   - Customers with a Concordia license should restore these:

     ° Concordia
     ° Essential Image Substrate
     ° Image Substrate

   - Customers with an IP-TCP license should restore these:

     ° IP-TCP

- Customers with an NFS license should restore these:
  - NFS-Client
  - NFS-Server
- Customers with a Statice license should restore these:
  - DBFS Kernel
  - DBFS
  - Statice Model Runtime
  - DBFS-DIR
- Customers with an X Windows license should restore these:
  - CLX

4. Click on [Perform Restoration].

5. When you have restored the tape, you should be sure to copy new flod files to each machine at your site using the command Copy Flod Files.

   On 3600-family machines:

   ```
   Copy Flod Files :version {V127, G206, or G208}
   ```

   On Ivory machines:

   ```
   Copy Flod Files :version I321
   ```

   For Ivory machines, update your `Hello.boot` file to scan the I321 flods.

Installing the New MacIvory Floppy Disks

There are three floppies included in ECO #1.

1. MacIvory System
2. MacIvory Applications
3. MacIvory Development

See the section "Installing the MacIvory Diskettes" for installation instructions.

Note: the software on these floppies is compatible with both 8.0 and 8.0.1, so you can install them before or after booting 8.0.1.

Installing the New UX400 Software

New Life Support software is provided with ECO #1. This software is to support SunOS 4.1. If you have not upgraded your UNIX to 4.1, do not load this software. See the section "Installing and Configuring the Symbolics UX UNIX Software" for installation instructions.

Note: If you load the UX Support system on a 3600-family machine, you should also load SYS:EMBEDDING;UX;UX-SUPPORT-TAPE-PATCH-FOR-3600.BIN. Failure to do this could result in writing unreadable LMFS backup tapes if you dump to a cart tape drive in a UX400S. Note that the contents of this file will be loaded automatically *in any Ivory world* as part of the UX Support system shipped in the 8.0.1 world; you only need to manually load this file if you have loaded the UX Support system into a world on a 3600-family.

Improvements and Bug Fixes in Genera 8.0 ECO #1

Improved Integration Between CLOS and Flavors

ECO #1 provides a new capability, in which CLOS generic functions can be invoked on Flavors instances. A parameter specializer name in a CLOS method can be the name of a flavor as well as the name of a class. In fact every flavor is now also a class, of metaclass **clos-internals::flavor-class**. This capability is required for CLIM. It also lays the groundwork for better integration between Statice and CLOS, in that CLOS methods can specialize on Statice entity handles, which are Flavors instances.

A number of small improvements in the performance and integration of CLOS have been made:

- The Inspector now works on CLOS instances.
- DW and CLOS interact better.
- Zmacs m-. and CLOS interact better.
- **trace**, **breakon**, and **advise** now work on CLOS generic functions and methods.
- The Find Symbol command can find symbols that are defined as CLOS classes. Show Callers, List Callers (m-X), **who-calls** and **what-files-call** can locate callers that instantiate CLOS classes and that are methods.
- There are many improvements to CLOS performance and correctness, especially for relatively obscure corners of the language used by CLIM.
- Method combination has been completely reimplemented and now supports **:arguments** and gives names to the functions that it generates.

- A function name or function spec for a method is now a method object. Old-style **(method ...)** function specs still work.

Users who wish to take advantage of all of these performance improvements should recompile their code.

New FEP for ECO #1

Changes to the FEP flods include:

- A bug in the the NFEP disk flod has been fixed. The bug was in the disk type specification for the XT8760 disk. If a disk was formatted twice, it usually would fail within a few weeks. The XT8760 disk type has been removed, and two new disk types have been added (XT8760-24 and XT8760-25), to support the old and new configurations of this drive.

- The Set Network-Address command has been moved from the Rel-7 flod to the Lisp flod, and the Rel-7 flod has been deleted.

- Some improvements have been made in the FEP debugger.

- The default world load is now found by searching for the world with latest timestamp.

- The default microcode is that required by the default world.

- The Load World command will print the contents of **sys:*lisp-release-string*** if it is of type array. If not, the release is calculated as was done previously.

- The IFU decode rams are loaded from data in the microcode file if the associated microcode block type is present. If not, they are loaded from FEP generated data as before.

Miscellaneous Improvements and Bug Fixes in ECO #1

- The :Before keyword argument to the Show System Modifications command now assumes that the value you specify is a time in the past. This means that a specification like :Before "Tuesday" will now be correctly interpreted as last Tuesday rather than next Tuesday. The new behavior is consistent with the :Since keyword.

- The undo functions for **si:delete-ie-commands** and **si:add-ie-command** have been fixed.

- In Genera 8.0, **package-name** would sometimes return one of the package nicknames rather than the primary name of the package. This bug has been fixed.

- A patch to the Serial system has improved the performance over sync-link gateways. The performance of interactive traffic over a busy gateway has been greatly improved. Larger packets of information are automatically queued to a pending queue enabling smaller packets to be queued immediately.

- MacIvory serial I/O works much more reliably.

- A bug in the UX400S serial interface has been fixed. The bug caused UNIX authentication credentials to be held across logins.

- RPC Authentication now includes checking UNIX passwords.

- Support for SunOS 4.1 NFS automount features has been added to NFS Client.

- TCP performance has been improved by enabling adaptive TCP retransmission. The default for **tcp::*enable-tcp-adaptive-retransmission*** is now **t**.

- Dialnet has been made more robust in cases where two systems get out of synchronization while exchanging characters. This situation is now detected and the connection is aborted. Previously, the two systems exchanged useless data forever, never timing out because there were always characters available, but never making any progress because because they were out of synchronization.

- Some peculiarities with incorrectly interning certain Dialnet host objects in the wrong namespace have been corrected.

- Graphics line drawing has been improved for XL machines.

- Some storage system bugs that manifest themselves as garbage collection bugs have been fixed.

- Another bug in the storage system has also been fixed. This bug could cause the system to waste some paging space. The amount of lost paging space without this ECO varies, but is larger when Dynamic or In-Place Garbage collection is used, particularly on Ivory systems. On Ivory systems which heavily use Dynamic GC, loading this ECO before the first Dynamic GC avoids wasting as much as 3000 pages of paging space.

- The use of Laserwriter functionality via Appletalk has been enhanced somewhat.

- A new function, **si:fix-fep-dpn** for fixing ECC errors on Ivory FEPs has been provided. Its behavior is exactly like **si:fix-fep-block** except that its arguments are *unit* and *page* (Disk Page Number).

- The mailer no longer blows out while trying to issue a warning if dialnet registries (obsolete in 8.0) exist. Instead, it successfully issues the warning.

- Dumping a LMFS to a cart tape on a UX400S now works correctly. Tapes written before loading this ECO, while they may show correct output when using the [List Backup Tape], were written incorrectly and cannot be restored. Note that this was only a problem if a UX400S cart tape was being used to back up the LMFS partition.

- Bugs involving unreliable operation of cart tapes on NBS machines, particularly timeouts during rewind, have been fixed.

- Support has been added for the MacinStor version 2.1 (Storage Dimensions) disk driver. Ivory disk partitions are now discovered when the Ivory is started, not when the Macintosh is started. This means that you can use the partition editor or reconfigure your disks while the Ivory is shut down and have the changes take effect without restarting the Macintosh.

- The bug that permitted you to delete a MacIvory disk partition that was in active use by the Ivory has been fixed.

- The ethernet performance problem with MacIvory IIci/IIfx has been fixed.

- The bug that caused the partition editor to give the wrong information about the amount of free space on a Macintosh volume has been fixed.

- The bug that caused **:draw-multiple-lines** on MacIvory to report errors has been fixed.

Documentation Updates for Genera 8.0 ECO #1

Clarification on Installing NFS

In Genera 8.1, the NFS system is obsolete and should not be loaded.

Instead, users should load NFS Client, NFS Documentation, and/or NFS Server as needed.

Clarifications on Printers

If you are using a 3600-family machine, you must have the systems RPC, Embedding-Support, and UX-Support loaded before you can use a printer spooled from a UNIX machine.

When installing an LGP2 or LGP3, the `Other Options` field in `Interface Options` should include the following:

```
NUMBER-OF-DATA-BITS 8 PARITY :NONE XON-XOFF-PROTOCOL YES
```

This is particularly important for printers connected to XL-family machines. This setting must be overridden if you are *not* using an Apple LaserWriter or LaserWriter II.

Genera 8.0 XL Documentation Update

This section pertains to XL-family machines only.

Changes to the VMEbus Interface in Genera 8.0 XL

- There are two new slave buffer variables, **sys:*vme-slave-buffer-base*** and **sys:*vme-slave-buffer-end***. They are the starting and ending addresses of the slave buffer in words, not bytes.

- The slave buffer base address is now settable in the FEP via the :Slave-Buffer Base keyword to the Set Boot Options command. This address is in bytes. The system defaults are correct unless the jumpers on the processor board have been moved.

- To let an application allocate a portion of the slave buffer, use **(sys:allocate-slave-buffer-memory** *:name :words* &key *from-end*). The function returns a starting and ending address in words. There is no enforcement, but this a simple check-out scheme for slave buffer memory so you do not accidentally use memory allocated by the system (as on the UX400) or by another application (like FrameThrower). Specific allocations can be added to an initialization list. System allocations take place on the system initialization list.

- Flonum data tagging is not supported on the XL1200.

- There are two new functions to enable and disable bus interrupts: **sys:enable-bus-interrupt** and **sys:disable-bus-interrupt**. They take a bus interrupt level from 1 to 7. They are somewhat easier to use than **sys:logior-bus-interrupt-mask** and **sys:logand-bus-interrupt-mask**.

- **sys:install-bus-interrupt** has been changed. It now takes a status ID rather than a level. Interrupt functions are dispatched by the status ID rather than the interrupt level. This change is incompatible with Genera 8.0.

- The default slave buffer address for the XL1200 is #xFAC00000.

- **sys:make-bus-address** of an address within the range of the slave buffer addresses will create the appropriate Ivory address to access the slave buffer. This is based on the variables **sys:*vme-slave-buffer-base*** and **sys:*vme-slave-buffer-end***.

- You can initiate VME SYSReset by calling the function **cli::merlin-ii-sysreset**. **Warning:** The effect of using this function is the same as pressing the RESET button on the machine.

- **sys:bus-read** and **sys:bus-write** now update the system bus parameters such as address-modifier and release-mode.

- The slave buffer on the XL1200 processor is 256K words long. Reservations of slave buffer memory are handled by **sys:allocate-slave-buffer-memory**.

- VMEbus errors on the XL1200 are not signalled on writes. Reads receive errors, as do polled reads using **bus-read**.

- The VMEbus interrupt handler does not disable the interrupt level for an incoming interrupt. The user process must disable and reenable the interrupt, if necessary.

- VMEbus interrupt status-ID #x5F on the XL1200 is reserved for Symbolics use. The Slave Trigger Interrupt uses this vector.

VMEbus Interface

Introduction to the XL-Family VMEbus Interface

The XL-family system is based on a 7-slot, 9U form factor VMEbus backplane and card cage. The VMEbus is a versatile, standard 32-bit bus which provides power and clock distribution, asynchronous data transfer, and interrupt delivery and acknowledgment. Although the performance requirements of modern processor/memory interconnects have exceeded the design range of the VMEbus, it remains popular as a peripheral I/O bus, and is often used in tandem with a separate high-speed memory bus. This is the strategy used in the XL: a private 48-bit bus connects the processor with its memory and I/O board, and the VMEbus interface is used to communicate with optional I/O peripherals and other 32-bit wide devices.

The XL processor is a VMEbus master, meaning that it can issue requests to read and write locations in other VMEbus cards, and can deliver interrupts. The interface provides a flexible *polled access* facility with which nearly all possible data transfer operations may be performed. It also provides a *direct-access* facility with which a portion of the VMEbus address space may be mapped into the XL's physical address space, for high-speed access to 32-bit slaves.

The XL processor is also a VMEbus slave, meaning that other bus masters can issue requests to read and write locations in it, and that it can receive interrupts issued by other bus masters. However, other bus masters may not directly access the XL processor's main memory; they may access only a dedicated memory in the XL VMEbus interface called the *slave buffer*. This design eliminates the hardware complication of arbitration deadlock between the VMEbus and the XL private bus, and eliminates the software complication of negotiating with the Genera virtual memory system to reserve contiguous portions of main memory.

The VMEbus is a flexible bus with many options and modes. In the design of the XL VMEbus interface, particular attention was paid to optimizing both the master and slave for high speed 32-bit data transfer, but the interface supports nearly all possible modes and can accommodate virtually any VMEbus device. The interface hardware includes the following features:

- On the XL400, the slave appears on the VMEbus as a 32K by 32-bit memory. On the XL1200, the slave is one megabyte in size.

- The master can transparently map up to 267 megawords of VMEbus address space into the Ivory physical address space (for 32-bit transfers only).

- Both the master and slave implement 8, 16, 24, and 32-bit data transfers, including transfers not aligned on an address boundary.

- Both the master and slave may specify that data be shuffled to compensate for differences in system bit, nibble, or byte ordering.

- Both the master and slave may request that data returned to Ivory be tagged as either integers or IEEE 32-bit floating-point numbers on the XL400. (On the XL1200, data returned can only be tagged as integers.)

- The master may specify an arbitrary address modifier for a data transfer.

- The arbitration parameters (arbitration level, bus release behavior) of the master are programmable.

- The interface can issue and receive all seven interrupt levels, and supports 8-bit interrupters.

- The slave implements the VMEbus block transfer protocol.

- The interface includes a system controller (containing VMEbus arbiter, clock drivers, and so on), which may be disabled by a jumper.

The interface hardware does not support the following features:

- The master does not implement the VMEbus block transfer protocol, but uses address pipelining to achieve equivalent performance.

- The master cannot issue ADDRESS-ONLY data transfer requests.

- The master cannot issue READ-MODIFY-WRITE data transfer requests, but atomic operations are supported by inhibiting bus release.

Software provided with Genera supports efficient access to all interface features, while shielding the client software from irrelevant hardware details. Data transfers may be performed in isolation via function calls that read or write specified bus locations, or by obtaining a physical address that the interface will map to a range of VMEbus addresses, or by using a Lisp indirect array which may be manipulated by normal array operations and facilities such as BITBLT. Full support for delivering and handling interrupts is provided.

For more information about the VMEbus hardware specification, see "The VMEbus Specification", Revision C.1, published by Printex. For more information about the electrical and mechanical characteristics of the XL VMEbus card cage, contact your Symbolics sales representative.

VMEbus Data Transfers

There are four basic techniques for performing VMEbus data transfers:

- Isolated transfers may be performed by calling the functions **sys:bus-read** and **sys:bus-write**, specifying the desired VMEbus address (and perhaps data) and any optional parameters. This technique is the most flexible, since it uses the polled access hardware in the interface which supports non-32-bit transfers.

However, it incurs some overhead programming the hardware and is therefore not very efficient.

- The function **sys:make-bus-address** may be called to map a portion of the VMEbus into the Ivory address space, and return a physical address pointing to it. That address may be manipulated using subprimitives such as **sys:%pointer-plus**, **sys:%p-ldb**, **sys:%p-dpb**, **sys:%block-read**, and **sys:%block-write**. This technique is very efficient, but is rather cumbersome. It also works for 32-bit slaves only.

- The function **sys:make-bus-array** may be called to map a portion of the VMEbus into the Ivory address space, and return an indirect array pointing to it. This allows high-level, bounds-checked access to array elements of any type, and the array may also be passed to Lisp facilities such as **bitblt**. This technique also works for 32-bit slaves only.

- Atomic operations may be performed by calling **sys:bus-store-conditional**, which works for VMEbus locations the same way **store-conditional** works for virtual memory locations.

All these techniques provide some way to configure the interface hardware to enable options such as bit shuffling, nonstandard address modifiers, and arbitration parameters. For polled transfers (via **sys:bus-read** and **sys:bus-write**), the options are specified as simple keyword arguments. For direct transfers (via **sys:make-bus-address** and **sys:make-bus-array**), most of the options are specified by the **sys:with-bus-mode** macro, which must surround any use of VMEbus addresses. See the section "Summary of VMEbus Transfer Options" for a description of the available options.

In general, clients should use polled transfers to refer to isolated registers on the VMEbus, and direct transfers to map memories, frame buffers, large register banks, etc., into the Ivory physical address space. See the section "VMEbus Direct Data Transfers".

VMEbus Direct Data Transfers

The VMEbus master can perform direct data transfers, in which a portion (called a *window*) of the VMEbus address space is mapped into the Ivory physical address space, and accessed as though it were (32-bit wide) Ivory memory. For direct transfers, some of the data transfer options, such as the arbitration parameters, are controlled by hardware registers that must be set up prior to the data transfer. Others, such as data shuffling, are controlled by fields within the Ivory physical address decoded by the VMEbus interface. **sys:with-bus-mode** and the address-generating functions **sys:make-bus-address** and **sys:make-bus-array** conspire to keep the hardware parameters consistent with the client's intent.

sys:with-bus-mode establishes a context within which VMEbus addresses may be generated and used; it is illegal to use a VMEbus address returned by **sys:make-bus-address** or **sys:make-bus-array** outside the dynamic scope of the **sys:with-**

bus-mode in which it was created. **sys:with-bus-mode** programs the VMEbus interface according to the specified options, and guarantees that those parameters will be maintained throughout the dynamic extent of the macro, even if some other process is trying to use the VMEbus simultaneously in a completely different manner.

The first time an address is generated (that is, **sys:make-bus-address** or **sys:make-bus-array** is called) within a given **sys:with-bus-mode**, the direct access window in the VMEbus interface is programmed to encompass the specified addresses. A subsequent attempt to generate an address that doesn't lie within the same 267-megaword window will signal an error. If this restriction causes problems, they can often be resolved by using polled transfers to refer to some of the disparate locations.

Note that **sys:bus-read**, **sys:bus-write**, and **sys:bus-store-conditional** are polled transfers and are therefore not affected by **sys:with-bus-mode**; they may be used at any time.

Summary of VMEbus Transfer Options

The following options may be specified to **sys:bus-read**, **sys:bus-write**, **sys:make-bus-address**, **sys:make-bus-array**, and **sys:with-bus-mode**:

:shuffle
: One of **:none**, **:byte**, **:nibble**, or **:bit**, this specifies the permutation to be applied to the data words received or transmitted by Ivory. **:bit** shuffling reverses the order of all 32 bits. **:byte** shuffling reverses the order of the four 8-bit bytes in a word, but preserves the order within each byte. **:nibble** does the same for 4-bit groups. The default is **:none**.

:data-type
: One of **:fixnum** or **:single-float**, this specifies the tag to be appended to data received by Ivory. **:fixnum** is the default, **:single-float** might be useful when communicating with an array processor or similar device. This is meaningful only for the XL400.

The following options may be specified to **sys:bus-read**, **sys:bus-write**, and **sys:with-bus-mode**:

:address-modifier
: The 6-bit numeric VMEbus address modifier code to be driven onto the bus during a data transfer cycle. The default is #x09, indicating that the address is 32 bits wide, for a data cycle.

:ownership
: One of **:release-when-done**, **:release-on-request**, or **:bus-hog**, this specifies the condition under which the VMEbus interface will relinquish ownership of the bus once it has control. The default is **:release-on-request**.

:arbitration-priority
An integer from 0 to 3, indicating the priority the VMEbus interface will assert when requesting access to the bus. The default is 3.

The following options may be specified to **sys:bus-read** and **sys:bus-write**:

:byte-size
One of 1, 2, 3, or 4, this specifies the number of bytes of VMEbus data. The VMEbus interface will issue an 8, 16, 24, or 32 bit operation as necessary to perform the transfer.

:byte-offset
One of 0, 1, 2, or 3, this specifies the first significant byte of the VMEbus data.

When using the **:byte-size** and **:byte-offset** options, note that all the specified bytes must be contained within an aligned 32-bit word. That is, the size plus the offset must be greater than zero and less than five.

VMEbus Interrupts

Interrupts may be posted on the VMEbus using the function **sys:post-bus-interrupt**, which issues an interrupt at a specified level, waits for the receiver to acknowledge, then delivers the specified status byte to the interrupt handler. Note that the VMEbus interface cannot deliver an interrupt to itself.

The VMEbus interface will interrupt the Ivory processor upon receipt of any VMEbus interrupt for an enabled level. Which levels are enabled is controlled by a mask in the interface hardware, which may be examined and altered using **sys:logior-bus-interrupt-mask** and **sys:logand-bus-interrupt-mask**. The mask contains a 1 in each bit for which an interrupt is enabled; for example, if the mask were #b00001010, interrupts at levels 1 and 3 would be received, and all others would be ignored. Upon receipt of an interrupt request, the VME software issues an interrupt acknowledge cycle to retrieve the status byte, and calls the appropriate client interrupt handler in a Genera simple process.

You can also use **sys:enable-bus-interrupt** and **sys:disable-bus-interrupt** to enable interrupts. Both functions take a level as an argument.

Client software may supply a handler function for a specific status/ID using **sys:install-bus-interrupt-handler**. An interrupt handler function is a normal Lisp function that takes one argument: the status/id byte received during the interrupt acknowledge cycle. The interrupt level is not disabled when the interrupt is received; the programmer must manage this.

VMEbus Slave Interface

The VMEbus interface for the XL400 and Symbolics UX-family contains a 32K by 32 bit memory that appears on the VMEbus as a slave device. The slave supports A24 and A32 address modes, and D08(EO), D16, and D32 data transfers.

The VMEbus interface for the XL1200 has a 256K by 32 bit buffer that supports the same modes.

The XL slave buffer responds to the following VMEbus address modifiers:

Modifier	*Description*
#x39	Standard normal data access
#x3A	Standard normal program access
#x3B	Standard normal block transfer
#x3D	Standard supervisor data access
#x3E	Standard supervisor program access
#x3F	Standard supervisor block transfer
#x09	Extended normal data access
#x0A	Extended normal program access
#x0B	Extended normal block transfer
#x0D	Extended supervisor data access
#x0E	Extended supervisor program access
#x0F	Extended supervisor block transfer

The XL400 slave buffer responds to VMEbus address #xFADC0000 (extended) and #xDC0000 (standard). The XL1200 slave buffer responds to VMEbus address #xFAC00000 (extended) and #xC00000 (standard).

The UX-family machine slave buffer responds to the following VMEbus addresses:

UX400S Board	*Extended Address*
0	#xFADC0000
1	#xFAEC0000
2	#xFAF40000
3	#xFAF80000
4	#xFABC0000
5	#xFA9C0000
6	#xFAAC0000
7	#xFAB40000
8	#xFAB80000

UX1200S Board	*Extended Address*
1	#xFD000000
2	#xFD200000
3	#xFD400000
4	#xFD600000
5	#xFD800000
6	#xFDA00000

7	#xFDC00000
8	#xFDE00000

The slave buffer may be accessed from Ivory using **sys:make-bus-address** or **sys:make-bus-array**, simply by specifying a VMEbus address that falls within the range of the slave buffer. Note that data transfers to such an address don't actually incur any VMEbus traffic; internal data paths are used. The **:shuffle** and **:data-type** options are supported for slave buffer transfers, and work just as they do for normal VMEbus transfers. See the section "Summary of VMEbus Transfer Options".

- The slave buffer address on XL1200 boards can be set via jumpers on the processor board. They are set at the factory to #xFAC00000.

- The mailbox address for each board is at #x100000 beyond the slave-buffer. For example, for UX1200S #1, (+ #xFD000000 #x100000) → #xFD100000

- Lisp keeps track of the location of the slave buffer in the variables **sys:*vme-slave-buffer-base*** and **sys:*vme-slave-buffer-end***. These addresses are in words, so use **(lsh sys:*vme-slave-buffer-base* 2)** to get the VME address of the slave buffer.

If a different VME address is used for the slave buffer, you can inform Lisp of the change by using the keyword :Slave Buffer Address to the Set Boot Options FEP command.

Note: Sections of slave buffer memory are reserved for use by Symbolics for certain hardware configurations. For more information, see the function **sys:allocate-slave-buffer-memory**.

Resetting the XL-Family and Symbolics UX400S VMEbus

The VMEbus SYSreset signal is asserted on the XL backplane shortly after initial powerup, and whenever the RESET button on the front panel is pressed. The XL processor board responds to SYSreset by initializing the Ivory processor and the I/O board, and cold-booting the FEP. The contents of the XL main memory are preserved, and the FEP software should be able to warm boot Genera if it was running prior to the SYSreset.

The XL400 does not generate or respond to the VMEbus SYSfail signal; the XL1200 does generate SYSfail.

Sun systems also assert SYSreset on powerup. The UX-family machine's processor board responds to SYSreset by initializing the Ivory processor and sending a signal to the machine's life support. When the UX-family machine's life support becomes available, it will cooperate with the UX-family machine's processor board in cold-booting the FEP. The contents of UX-family machine's main memory are preserved, and the FEP software should be able to warm boot Genera if it was running prior to SYSreset.

You can initiate SYSreset on an XL1200 by using the function **cli::merlin-ii-sysreset**. This is equivalent to pressing the RESET button on the front panel.

Examples of Using the VMEbus Interface

This section shows several different ways to perform a simple VMEbus data transfer operation in which the goal is to copy a contiguous block of 32-bit words from one VMEbus address to another, reversing the 4 8-bit bytes with each word.

```
;;; Given an A32 D32 slave, use polled transfers to copy each
;;; word.  The bytes are shuffled by the interface hardware as
;;; each word is read from the source.  Simple but slow.
(defun copy-VME-memory-shuffling (source-bus-address
                                  destination-bus-address words)
  (loop repeat words
        for s from source-bus-address
        for d from destination-bus-address
        do
    (sys:bus-write d (sys:bus-read s :shuffle :byte))))

;;; Given an A32 D16 slave, use polled transfers to copy each
;;; 32-bit word in two halves.  The bytes within each 16-bit word
;;; are shuffled by the interface hardware as each word is read
;;; from the source, but we have to manually interchange the two
;;; halves of each 32-bit word.
(defun copy-VME-memory-shuffling (source-bus-address
                                  destination-bus-address words)
  (loop repeat words
        for s from source-bus-address
        for d from destination-bus-address
        do
    (let ((v (sys:bus-read s :shuffle :byte :byte-size 2
                             :byte-offset 0)))
      (sys:bus-write d v :byte-size 2 :byte-offset 2))
    (let ((v (sys:bus-read s :shuffle :byte :byte-size 2
                             :byte-offset 2)))
      (sys:bus-write d v :byte-size 2 :byte-offset 0))))
```

```
;;; Given an A16 D8 slave, use polled transfers to copy each
;;; 32-bit word in four separate bytes.  We have to do the byte
;;; swapping manually.  We have to use the :address-modifier
;;; option to specify short (A16) addresses.
(defun copy-VME-memory-shuffling (source-bus-address
                                  destination-bus-address words)
  ;; This with-bus-mode isn't actually required, we could instead
  ;; specify an :address-modifier option to every bus-read and
  ;; bus-write.  But the options for those operations take their
  ;; defaults from the ambient with-bus-mode, so this is
  ;; syntactically cleaner.
  (sys:with-bus-mode (:address-modifier #x29)
    (loop repeat words
          for s from source-bus-address
          for d from destination-bus-address
          do
      (let ((v (sys:bus-read s :byte-size 1 :byte-offset 0)))
        (sys:bus-write d v :byte-size 1 :byte-offset 3))
      (let ((v (sys:bus-read s :byte-size 1 :byte-offset 1)))
        (sys:bus-write d v :byte-size 1 :byte-offset 2))
      (let ((v (sys:bus-read s :byte-size 1 :byte-offset 2)))
        (sys:bus-write d v :byte-size 1 :byte-offset 1))
      (let ((v (sys:bus-read s :byte-size 1 :byte-offset 3)))
        (sys:bus-write d v :byte-size 1 :byte-offset 0)))))
```

The remaining examples use direct transfers to perform this same operation, and therefore work for D32 slaves only.

```
;;; Map the VMEbus addresses into Lisp arrays, then use a Common
;;; Lisp sequence operator to do the copying. The bytes are
;;; shuffled by the interface hardware as each word is read from
;;; the source. Simple and reasonably efficient for large
;;; transfers, although the setup overhead is fairly high.

(defun copy-VME-memory-shuffling (source-bus-address
                                  destination-bus-address words)
  ;; with-bus-mode must be wrapped around all uses of
  ;; direct-transfer addresses.
  (sys:with-bus-mode ()
    (stack-let ((s (sys:make-bus-array source-bus-address words
                                       :shuffle :byte))
                (d (sys:make-bus-array destination-bus-address words)))
      (replace d s))))
```

```
;;; Map the VMEbus addresses into physical addresses and use
;;; simple memory subprimitives to do the copying.  Efficient for
;;; short transfers because of the low setup overhead, but low
;;; level and error prone.
(defun copy-VME-memory-shuffling (source-bus-address
                                  destination-bus-address words)
  ;; with-bus-mode must be wrapped around all uses of
  ;; direct-transfer addresses.
  (sys:with-bus-mode ()
    (loop repeat words
          for s first (sys:make-bus-address source-bus-address words
                                            :shuffle :byte)
                then (sys:%pointer-plus s 1)
          for d first (sys:make-bus-address destination-bus-address words)
                then (sys:%pointer-plus d 1)
          do
      (sys:%memory-write d (sys:%memory-read s)))))

;;; Map the VMEbus addresses into physical addresses and use
;;; block memory operations to do the copying.  This is the most
;;; efficient way to do bulk transfers.
(defun copy-VME-memory-shuffling (source-bus-address
                                  destination-bus-address words)
  ;; with-bus-mode must be wrapped around all uses of
  ;; direct-transfer addresses.  Direct transfers will work for
  ;; A24 and A16 slaves, using the :address-modifier option to
  ;; with-bus-modes as follows.  If we're really trying to be fast
  ;; and don't mind being nasty, we can do the entire transfer
  ;; without ever relinquishing the bus to another master, using
  ;; the :ownership option.
  (sys:with-bus-mode (:address-modifier #x39 :ownership :bus-hog)
    ;; with-block-registers must be wrapped around all uses of
    ;; block registers.
    (sys:with-block-registers (1 2)
      ;; Use block register 1 to address the source
      (setf (sys:%block-register 1)
            (sys:make-bus-address source-bus-address words
                                  :shuffle :byte))
      ;; Use block register 2 to address the destination
      (setf (sys:%block-register 2)
            (sys:make-bus-address destination-bus-address words))
      ;; Use an unrolled loop to copy the words, which makes the
      ;; memory pipeline operate most efficiently.
      (sys:unroll-block-forms (words 4)
        (sys:%block-write 2 (sys:%block-read 1))))))
```

Dictionary of VMEbus Functions

sys:allocate-slave-buffer-memory *name words* &key *:from-end* *Function*

Returns a starting and ending address in words. There is no enforcement, but this a simple check-out scheme for slave buffer memory so you do not accidentally use memory allocated by the system (as on the UX-family machine or by another application (like FrameThrower). Specific allocations can be added to an initialization list. System allocations take place on the system initialization list.

sys:bus-error *Flavor*

This condition is signalled if there is a VMEbus error such as a request timeout. Errors are signalled only on read operations; the XL400 processor stores errors that occur on write operations to be signalled by a future read operation.

sys:bus-read *bus-address* &rest *options* *Function*

Reads the location specified by *bus-address* using a polled transfer. All options default to those specified by the ambient bus mode.

See the section "Summary of VMEbus Transfer Options" for a description of the applicable bus options.

sys:bus-store-conditional *bus-address old new* &rest *options* *Function*

Checks to see whether the specified bus location contains *old*, and, if so, stores *new* in that location. The test and set are done as a single atomic operation; no other bus operations are allowed between the two. Both the read and the write are performed using the specified bus options, if any, which default to those specified by the ambient bus mode, if any. **sys:bus-store-conditional** returns **t** if the test succeeded and **nil** if the test failed. See the function **store-conditional**.

See the section "Summary of VMEbus Transfer Options" for a description of the applicable bus options.

sys:bus-write *bus-address value* &rest *options* *Function*

Stores the specified *value* into the location specified by *bus-address* using a polled transfer. All options default to those specified by the ambient bus mode.

See the section "Summary of VMEbus Transfer Options" for a description of the applicable bus options.

sys:deallocate-slave-buffer-memory *name* *Function*

name represents the portion of the slave buffer that was allocated by **sys:allocate-slave-buffer-memory**.

sys:disable-bus-interrupt *level* *Function*

Disables VMEbus interrupts at *level*. *level* can be between 1 and 7.

sys:enable-bus-interrupt *level* *Function*

Enables VMEbus interrupts at *level*. *level* can be between 1 and 7.

sys:install-bus-interrupt-handler *function status-id* *Function*

Installs *function* as the interrupt handler for interrupts within the specified *status-id*. When an interrupt is detected at that interrupt level, and that interrupt level is enabled in the interrupt mask, *function* will be called with one argument, the status/id byte supplied by the interrupter. The handler will be called in a simple process, and therefore must not depend on the dynamic environment (special variable bindings, catch tags, and so on.

Note that if *function* is redefined, the handler must be installed again for the new definition to take effect.

sys:logand-bus-interrupt-mask *mask* *Function*

Atomically reads the VMEbus interrupt enable mask register, logands it with the *mask* argument, and stores the result back in the register. This function is useful for disabling particular interrupts. It returns the new value, so the current state of the interrupt mask can be read as follows:

```
(sys:logand-bus-interrupt-mask -1)
```

sys:logior-bus-interrupt-mask *mask* *Function*

Atomically reads the VMEbus interrupt enable mask register, logiors it with the *mask* argument, and stores the result back in the register. This function is useful for enabling particular interrupts. It returns the new value, so the current state of the interrupt mask can be read as follows:

```
(sys:logior-bus-interrupt-mask 0)
```

sys:make-bus-address *bus-address size* &rest *options* *Function*

Returns an Ivory physical address usable to access the specified location on the VMEbus. This address is usable within only the ambient **sys:with-bus-mode**. All options default to those specified by the ambient bus mode. An error is signalled if there are any conflicts between the specified options and the hardware configuration specified by the ambient bus mode, or if the desired address range is not supported by the hardware. The first call to **sys:make-bus-address** or **sys:make-bus-array** within a **sys:with-bus-mode** will set up any necessary address window; if a subsequent call specifies an address range outside that window an error will be signalled.

See the section "Summary of VMEbus Transfer Options" for a description of the applicable bus options.

sys:make-bus-array *bus-address dimensions* &rest *options* *Function*

Returns an indirect array pointing to the specified VMEbus address, using the direct transfer facility. This array is usable only within the ambient **sys:with-bus-mode**. The options include all the **make-array** options, but note that the array cannot contain arbitrary Lisp objects, only integers and single-precision floating point numbers; see the section "Keyword Options for **make-array**". The options may also include any applicable bus options, which default to those specified by the ambient bus mode. An error is signalled if there are any conflicts between the specified options and the hardware configuration specified by the ambient bus mode, or if the desired address range is not supported by the hardware. The first call to **sys:make-bus-address** or **sys:make-bus-array** within a **sys:with-bus-mode** will set up any necessary address window; if a subsequent call specifies an address range outside that window an error will be signalled.

See the section "Summary of VMEbus Transfer Options" for a description of the applicable bus options.

sys:post-bus-interrupt &optional *(level 0) (status 0)* *Function*

Issues an interrupt request for the specified level on the bus, waits for the interrupt acknowledge cycle, then transmits the specified status/id byte to the interrupt handler.

sys:*vme-slave-buffer-base* *Variable*

The starting address for the slave buffer, in words.

sys:*vme-slave-buffer-end* *Variable*

The ending address for the slave buffer, in words.

sys:with-bus-mode *(*&rest *options)* &body *body* *Macro*

Establishes a context within which VMEbus addresses may be generated and used; it is illegal to use a VMEbus address returned by **sys:make-bus-address** or **sys:make-bus-array** outside the dynamic scope of the **sys:with-bus-mode** in which it was created. **sys:with-bus-mode** programs the VMEbus interface according to the specified options, and guarantees that those parameters will be maintained throughout the dynamic extent of the macro, even if some other process is trying to use the VMEbus simultaneously in a completely different manner.

See the section "Summary of VMEbus Transfer Options" for a description of the applicable bus options.