3.12.17 mrapi_timeout_t

The `mrapi_timeout_t` type is an unsigned scalar type used to indicate the duration that an `mrapi_wait()` or `mrapi_wait_any()` API call will block before reporting a timeout. The units of the `mrapi_timeout_t` datatype are implementation defined since mechanisms for time keeping vary from system to system. Applications should not rely on this feature for satisfaction of real time constraints as its usage will not guarantee application portability across MRAPI implementations. The `mrapi_timeout_t` datatype is intended only to allow for error detection and recovery. The `mrapi_timeout_t` has an `mca_timeout_t` equivalent. The reserved values are 0 for do not block at all, and MAX(unsigned 32-bit) for `MRAPI_INFINITE`.

3.12.18 Other MRAPI data types

MRAPI also defines its own integer, Boolean and other types, some of which have MCA equivalents. See the header files in this document for specifics on these data types.

3.12.19 mrapi_msg_t (ABB extension)

The `mrapi_msg_t` type must be the first member of a structure exchanged between nodes running in different processes on non-Windows operating systems. For array buffers used in data exchange, each array element has the type as its first data member.

3.12.20 mrapi_atomic_barrier_t (ABB extension)

The `mrapi_atomic_barrier_t` type is a synchronization descriptor for non-Windows cross-process shared memory synchronization. It is initialized by a call to either `mrapi_barrier_init()` or `mrapi_exchange_init()`. The call sets the barrier properties to control the policies used for cross-process data exchange. If the destination exchange participants are in the same process the the barrier has no effect.

3.13 MRAPI Compatibility with MCAPI

The MRAPI working group is following in the footsteps of the MCAPI working group. Therefore, this specification has adopted similar philosophies, and the same style for the API, datatypes, etc. Furthermore, since MRAPI and MCAPI are part of the larger Multicore Association Roadmap, the working group expended great effort to ensure that MRAPI functionality is orthogonal to MCAPI functionality while making sure they are interoperable (for example discussions around shared memory for MRAPI and zero copy messaging for MCAPI.)

3.14 Application Portability Concerns

The MRAPI working groups desires to enable application portability but cannot guarantee it. The guiding principles that should be used by application writers are (i) to write as much of the application in as portable a fashion as possible and (ii) encapsulating optimizations for efficiency or to take advantage of specialized dedicated hardware acceleration where possible and necessary. The end result of this approach should be that from a given MRAPI node's perspective it should not be possible nor required for that node to know whether it is interacting with another node within the same process, on the same processor, or even on the same chip. Furthermore, it should not be transparent to a given node whether it is interacting with another node that is implemented in hardware or software. The MRAPI working group believes that this approach will allow portability of software to be maintained at the interface level (e.g., the functional interface between nodes). However, the software implementation of a particular node cannot (and often should not) necessarily be preserved across a multicore SoC product line (or across product lines from different silicon providers) because a given node's functionality may be provided in different ways -- depending on the chosen multicore SoC. For more discussion on MRAPI nodes see Section 3.3.

3.15 MRAPI Implementation Concerns

This section provides guidance to implementers of MRAPI.

3.15.1 Thread Safe Implementations

MRAPI implementations are assumed to be reentrant (thread safe). Essentially, if an MRAPI implementation is available in a threaded environment, then it must be thread safe. MRAPI implementations can also be available in non-threaded environments. The provider of such implementations will need to clearly indicate that the implementation is not thread safe.