Q: Will MCAPI require modification of existing operating systems?

A: MCAPI is a communication API and does not define requirements for implementation. It is possible to implement MCAPI on top of a commercial off-the-shelf OS with no OS modifications, however an efficient implementation of MCAPI may require changes to the operating system.

Q: Can MCAPI be implemented in hardware?

A: The goal of MCAPI is to make possible efficient implementation in hardware.

Q: If MCAPI is oriented towards static connection topologies, why support methods like mcapi_endpoint_delete()?

A: The ability to release any resources allocated by an endpoint may be particularly important in hardware implementations where resources are strictly limited. Supporting the deletion of endpoints allows the application programmer to dispose of endpoints that are only necessary for a portion of the application's life.

Q: What facilities are provided for debugging MCAPI programs?

A: The MCAPI API is designed to be implemented as a C library, using standard tools and enabling the use of standard debuggers and profilers. The MCAPI error codes should also help give some visibility into what sort of program errors have occurred. The MCAPI channel-connection mechanism is designed to work with static layout tools, so that a channel topology can be defined and debugged outside of the program itself. We also expect that some implementations will provide tools for dynamically collecting information like which channels have been connected and how many packets have been transferred.

Q: Why doesn't the MCAPI API support one to multiple, multiple to one and barrier type communications?

A: To keep the MCAPI API simple and allow for efficient implementations more complex functions that can be layered on top of MCAPI are not part of the MCAPI specification.

Q: Why doesn't MCAPI provide name service functionality?

A: The MCAPI API is meant for multicore chips or multi-chip boards which are static environments in the sense that the number of cores is always the same. A naming service is therefore not needed and would add unnecessary complexity. The initialization functionality added in version 2.000 provides for basic topology discovery. A naming service could be implemented on top of MCAPI.

Q: What happens if my cores have different byte order (endian-ness)?

A: The MCAPI API specification doesn't state how to implement an MCAPI API. The specific implementation would have to address different byte order.

Q: How do I use my MCAPI calls so that the source code is portable between implementations with different node numbering policies?

A: To make code portable between implementations that allow only one node number (and mcapi_initialize call) per process, and those that allow one node number (and mcapi_initialize call) per thread, the code should only make one mcapi_initialize call per process. If there are multiple threads within the process, they should use different port numbers
(endpoints) for communication.

Similarly, to make code portable between implementations that allow only one node number (and mcapi_init call) per processor, and those that allow one node number (and mcapi_init call) per task, the code should only make one mcapi_init call per processor. If there are multiple tasks running on the processor, they should use different port numbers (endpoints) for communication.