Q: Does the MCAPI spec provide for multiple readers or multiple writers on an endpoint?

A: No the spec does not specifically provide for this. However, it does not prevent it either. An MCAPI implementation can choose to provide a mechanism whereby multiple endpoints resolve to essentially the same destination. For example, if an SoC contained a hardware device which managed multiple hardware queues used for moving messages around the SoC, an MCAPI implementation could allow multiple writers to a single hardware queue by simply mapping a set of distinct send endpoints to that single hardware queue. The implementation might choose to do this by assigning a range of port_ids to map to the single hardware device. For example, assume the MCAPI implementation identifies the hardware device that provided the hardware managed queues as node_id 10, and the range of ports 0..16 is assigned to map to hardware queue zero within that device. Then a send to endpoint <0,10,0> as well as a send to endpoint <0,10,1> would both map to a send to queue zero in the hardware device. Different nodes within the system wishing to send to the queue would have to allocate a unique send endpoint, but the sends would all end up in queue zero in the hardware accelerator. Similarly, an MCAPI implementation could map multiple receive endpoints to a single hardware queue. In this example any reads from multiple endpoints mapped to the single hardware queue would be serviced on a first come, first served basis. This would be suitable for a load balancing application, but would not be sufficient to serve as a multicast operation. Support for multicast operations will be considered in future MCAPI specification releases.

Q: How can I implement callback capability with MCAPI?

A: Although MCAPI does not directly provide a callback capability, MCAPI is designed to have many kinds of application services layered on top of it, and callbacks can be done this way. This approach would require you to use the MCAPI non-blocking send or receive functions, and to write a new function which would take the mcapi_request_t returned from calling these MCAPI functions along with a pointer to your callback function as parameters. This new function could use mcapi_test, mcapi_wait, or mcapi_wait_any to determine when the outstanding MCAPI request has completed and then call your callback function. This might be implemented by having a separate thread monitoring a list of outstanding requests and providing callbacks on completion. If you cannot create multithreaded applications your callback solution may be more difficult to implement.

Q: I'd like to have a name service for looking up MCAPI endpoints. How can I achieve this functionality?

The MCAPI standard defines a communication layer that allows easy creation of a name server. The application can choose a statically determined (domain, node, port) tuple where the name service will run. For example, domain, node 0, port 0 might be a good choice. The name server process can be started by calling mcapi_initialize(0,0, …) to bind itself to domain 0, node 0 and then calling mcapi_endpoint_create(0, &status). The name server can then use message send and receive operations to implement its service. Clients can discover the name server by invoking mcapi_endpoint_get(0, 0, 0, &status) to get the its endpoint_t. Given the endpoint_t, each client can send and receive messages as required by your name service protocol.