3.6 MRAPI Metadata

MRAPI provides a set of API calls designed to allow access to information regarding the underlying hardware context an application is running on. These capabilities are described in more detail in the following sections.

3.6.1 Metadata Resource Data Structure

A call to `mrapi_resources_get()` returns a data structure of type `mrapi_resource_t` , see Section 3.12.4. This data structure is provided in the form of a tree containing the set of resources that are visible to the calling MRAPI node. Each node in the tree represents a resource in the system, and each node contains attributes that provide additional information about a given resource. The resource tree may be optionally filtered by the `subsystem_filter` input parameter. Examples of such filters include CPU, cache, and hardware accelerators. An MRAPI implementation must define what filters it can provide as an enumerated type.

The resource data structure can contain hierarchical nodes in addition to the resource nodes themselves. For example, the concept of a core complex, which could contain multiple cores, would be represented as a parent node to the core nodes in the resource tree.

During initialization MRAPI may read in the system resources from a data file which may have a tree structure, such as XML or a device tree, so it is convenient to represent the resource data structure as a tree. Alternatively, the resources could be statically compiled in to the MRAPI implementation.

See section 6.1 for a use case and example code for getting and navigating a resource tree.

3.7 Attributes

Attributes are provided as a means of extension for the API. Different implementations may define and support additional attributes beyond those pre-defined by the API. In order to promote portability and implementation flexibility attributes are maintained in an opaque data structure, which may not be directly examined by the user. Each resource (e.g., mutex, semaphore, etc.) has an attributes data structure associated with it, and many attributes have a small set of predefined values which must be supported by MRAPI implementations The user may initialize, get and set these attributes.

If the user wants default behavior, then the intention is that they should not have to call the init/get/set attribute functions. However, if the user wants non-default behavior then the sequence of events should be as follows:

1) `mrapi_<`*`resource`*`>_init_attributes()`: returns an attributes structure with all attributes set to their default values.

2) `mrapi_<`*`resource`*`>_set_attribute():` (repeat for all attributes to be set): sets the given attribute in the attributes structure parameter to the given value.

3) `mrapi_<`*`resource`*`>_create():` you would pass in the attributes structure modified in the previous step as a parameter to this function.

Once a resource has been created, its attributes may not be changed.

At any time, the user can call `mrapi_<`*`resource`*`>_get_attribute()` to query the value of an attribute.

For a use case where attributes are customized, see section: 6.1.