

are interpreted as different numbers and/or different types of parameters. The precise manner in which they are interpreted depends upon the logical scheme in question. Sometimes a portion of these opaque bits are interpreted based on the specific physical description of the scheme in question. **FIG. 2** illustrates an example of the manner in which these block of bits can be supported according to embodiments of the present invention. Those skilled in the art will realize that here are many possible ways to implement this invention. Different programming language run-time facilities will tend to favor different implementations. **FIG. 2** illustrates one way to implement the common prototypes within a (possibly fixed-size) operational block, using the C programming language.

[0058] Although some microprocessors directly support 64-bit machine words, the exact number of opaque bits used is arbitrary. This depends upon the system scheme to which the invention is applied. The approach works equally well with any number of bits. As needs arise, the number of opaque bits passed (to all described facets in the schema used for any particular embedded software/firmware part) can be expanded. Those skilled in the art will realize that the block of bits is preferably not typed.

[0059] Regardless of the number of bits used, for preferred embodiments of the present invention, the number should be a constant for a given part. In some less preferred embodiments of the present invention, varying number of bits may be used, although this will add to the system's complexity. With a fixed number of opaque bits in the block of bits, each and every behavior facet can always take a single, standard, block-of-bits argument. This is a unique mechanism, and method, for implementing a behavioral facet (or slot). In part, this distinguishes a descriptive facet from an ordinary function, an ordinary object method, etc.

[0060] Returning to the power supply example, the following table (Table 1) depicts an example mapping the described Power Supply (logical) concept into the block of (opaque) bits.

TABLE 1

Logical Power Supply Description (Block of bits only)			
Section/Bit Offset	# bits	Purpose	Sample Value
1	16	Packed Bit Field	0x0404
15:8	8	Logical Power On (GPIO) Signal	0x04
7:0	8	Logical Power Off (GPIO) Signal	0x04
2	16	Packed Bit Field	0x0605
15:8	8	Logical Power Status (GPIO) Signal	0x06
7:0	8	Logical Power Reset (GPIO) Signal	0x05
3	32	Packed Bit Field	0x053205FF
31:24	8	Power On Signal Pulse Width	0x05
23:16	8	Power Off Signal Pulse Width	0x32
15:8	8	Power Reset Pulse Width	0x05
7:7	1	Use IRQ instead of signal for power status	1b
6:6	1	External (true) or Internal (false) IRQ type	1b
5:2	1	Interrupt Request (IRQ) number	0x4
1:0	2	Trigger edges (0 = front, 1 = back, 2 = both)	106

Flexible Description Facets

[0061] To optimize existing implementations, a number of techniques may be used to compress varying numbers, and varying types, of arguments/parameters into a sequence of bit fields packed (tightly/efficiently) into the block of bits. To completely standardize the unique interface facet signatures, some embodiments of the present invention adopt a certain universal enumeration of interface facet that returns values (referred to as "status" values). In addition, systems according to embodiments of the present invention preferably use a conformant function to implement both state/attribute/property access facets (e.g., so-called "getters" and "setters") and behavioral facets.

[0062] Thus far, a logical description approach (via a specific scheme conforming to some specific schema conventions) has been described. This supports instances of various prototypes. Each such instance can represent one peripheral hardware device.

[0063] While this novel mechanism/method can be applied in any number of ways, the power supply example shows how it supports the representation of a very large number of Power Supply types. The logical interfaces of the Power Supply type have been parameterized in a single, uniform and consistent way, e.g., using an opaque block of bits. The present invention makes it possible for a fixed software/firmware image to use any interface facet that is associated with any scheme instance. Diverse schemes can represent a very wide range of peripheral devices (within a peripheral device schema). These mechanisms, and the corresponding methods, provide a constant and a consistent way to bind an implementation to the described facets.

Declarative Description Facet Signatures

[0064] Up to now, the block of bits have been described as opaque. The power supply example is now used to illustrate how other mechanisms, and methods, in the present invention support real-world power supply idiosyncrasies.

[0065] The block-of-bits argument/parameter is associated with the specific set of facets that meet the stated/implied requirements/intentions of some given logical/physical scheme (for Power Supply). Focus first on the logical facets required by the power supply scheme. This logical description outlines the set of Power supply facets/interfaces that power supplies must somehow support. It is the associated Power Supply scheme that determines how the block of bits will be interpreted. Once it is known that we are dealing with an instance of a Power Supply description, the block of bits can be interpreted. The block of bits is not opaque to logic that knows the Power Supply context (for some given block of bits instance).

[0066] The interpretation of various, nested sections of the block of bits can build up across the generalization/specialization type hierarchy. To keep the example simple, assume that each specialized (leaf) type (like Power Supply) can treat the entire block of bits as its own, as is often the case. For instance, the even more general notion of a peripheral (kind of type) might not impose any additional conventions upon the block of bits interpretation.

[0067] In the example, the Power Supply type declares and imposes its own special block of bits interpretation (as was detailed in Table 1 (above)). The actual values used