valstat - call/response protocol

Document Number:	**DBJ0001**
Date	2021-01-01
Audience	
Author	Dusan B. Jovanovic (dbj@dbj.org)

Simplicity is the ultimate sophistication -- Leonardo Da Vinci

Table of Contents

Revision history

R1: created

1. Abstract

This is a proposal about logical, feasible, lightweight and effective protocol for handling the call/response activity, decoupled from both platforms and languages.

This paper describes an software protocol, not language specific implementations or platform specific solutions.

2. Motivation

Economy of software production is an ever more delicate balancing act. Because of its prominent costs, feasible integration is a common recurring theme in a software development. It is complex and difficult to predict and solve in a feasible level of detail. Thus it contributes to the raising costs of software.

Levels of Integration, ordered by the scope

scope	level
widest	universal
	domain
	system
	application
	component
narrowest	code

Calling, creating a response, returning a response and handling the response, is activity permeating all levels.

Feasibility of the integration requires a high level of resilience. Integration resilience requires a common and simple guidances. VALSTAT is attempt to provide on of those guidances, in the format of a standard protocol.

3. The Protocol

By "call" in here we mean the "call" in its most overarching definition. This protocol is applicable regardless of the call/response category. Remote, Local, Synchronous, asynchronous, direct, message based, RPC, HTTP, or whatever.

A call/response, software activity guided by a protocol is a paradigm shift.

"A paradigm is a standard, perspective, or set of ideas. A paradigm is a way of looking at something ... When you change paradigms, you're changing how you think about something..." vocabulary.com

3.1. Building Blocks

VALSTAT protocol central theme is a lightweight structure returned by "Responder" to the "Caller", as a consequence of a call.

- `CALLER` is software entity calling the local or remote function or component.
- `RESPONDER` is a software entity creating a VALSTAT structure to be returned to a `CALLER`

```
CALLER --> (local/remote call) --> RESPONDER

CALLER <-- (VALSTAT structure) <-- RESPONDER
```

Locality: `CALLER` or `RESPONDER` can reside inside the same or different, application or system domains. Local or remote to each other.

Response using logic is divided in two steps

As part of response handling activity, regardless of platform or language, the two steps logic is always present:

- step one -- Is something returned?
- step two -- Can I use it?

Conceptually valstat protocol is the "two steps" return processing:

1. use the structure returned to determine the step two
   - not using the type system or actual values returned
2. use the content returned
   - using the type system and values returned

That is a two step logic, that can be applied across many (if not all) software development platforms and run-time domains.

3.2. VALSTAT structure

VALSTAT structure is an record made of two fields:

- VALSTAT record
  - VALUE
  - STATUS

3.3. Field

VALSTAT "field" is analogous to the database field. "field" is the name for an "single piece of information".

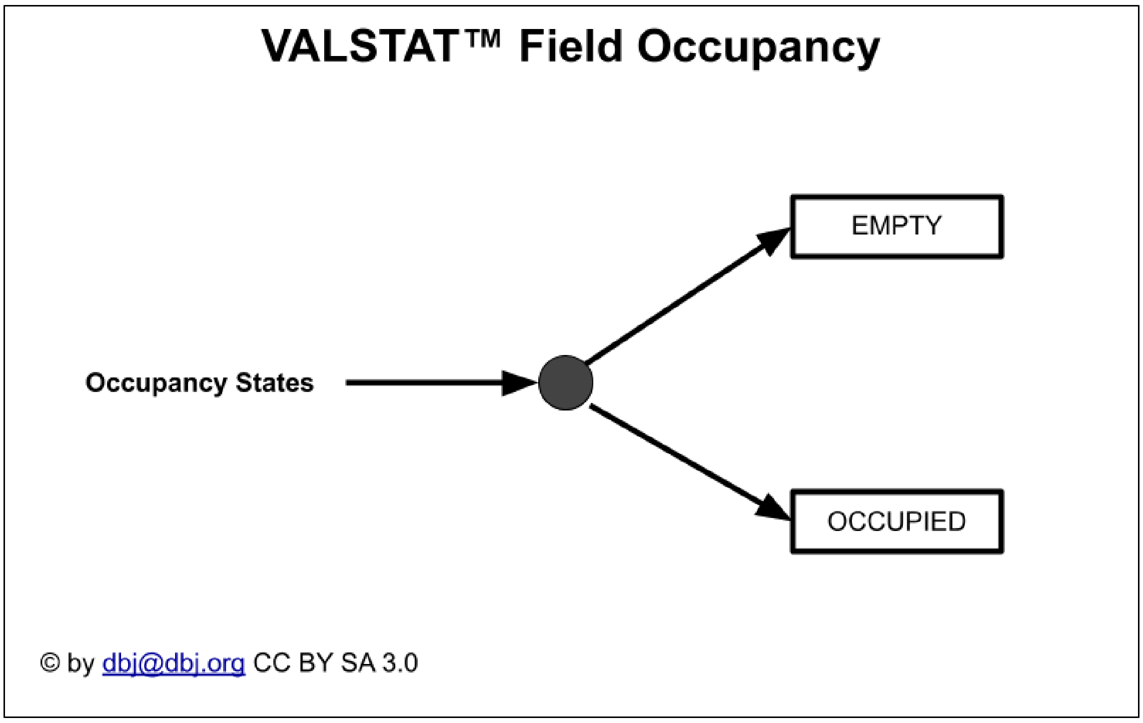
In database theory "field" is: *"a particular piece of data encapsulated within a class or object"*.

Key attributes of a field are:

1. field always exists
2. field can be empty or "contain a value"

3.3.1. Occupancy states

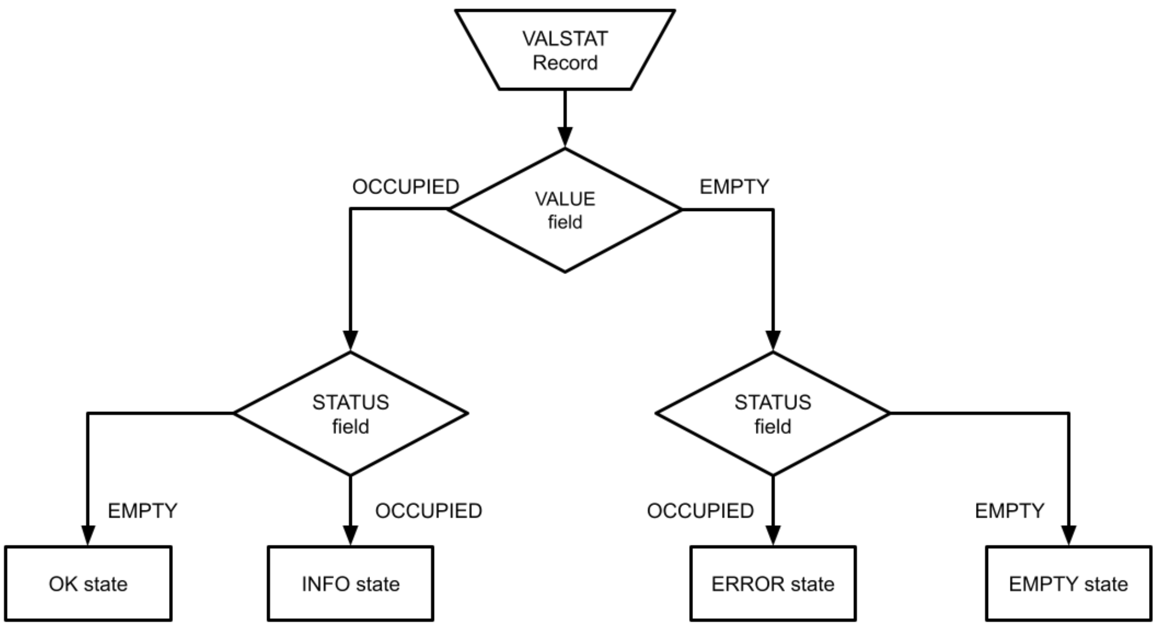
Field can be in two "occupancy states" (authors term) . We will call them : "empty" and "occupied".



In software development terms field is an object **potentially** holding only one value. It might be tested if it is "empty" , or occupied aka "holding a value".

3.4. VALSTAT State

Combination of two fields value *and* status occupancies is giving four possible states.



Valstat Record state decoding

State Tag	Value occupancy	op	Status occupancy
Info	Has value	AND	Has value
OK	Has value	AND	Empty
Error	Empty	AND	Has value
Empty	Empty	AND	Empty

VALSTAT state tags are just that: tags. Not mandating but just indicating the behavior.

In adopting the returned VALSTAT structure two step handling we do not immediately inspect returned values. In step one we inspect the relationship of two fields returned first.

In the most (all?) programing languages, there is no need for existence of some dedicated VALSTAT types. Returned structure handling, step one is only about the relationship between two fields states of occupancy.

VALSTAT state decoding is the act of decoding the relationship between occupancy states of its two fields.

Following is synopsis of decoding one of the four possible VALSTAT states:

```
// (c) by dbj@dbj.org
// pseudo code
// step one: capturing one of four possible VALSTAT states
```

In step one types or values of the content returned are not used just the occupancy states

```
// empty AND empty
  if (   is_empty( value ) &&   is_empty( status )  { /* state: empty */ }

// empty AND occupied
  if (   is_empty( value ) && ! is_empty( status )  { /* state: error   */ }

// occupied AND empty
  if ( ! is_empty( value ) &&   is_empty( status )  { /* state: ok */ }

// empty AND empty
  if ( ! is_empty( value ) && ! is_empty( status )  { /* state: info*/ }
```

On the code level, that synopsis can be implemented, almost as it is, in many languages: C, JavaScript, Python, GO, Java, C# etc.

NOTE: in particular language definition of the VALSTAT we very often do not need `is_empty()` function implemented. Here is canonical C++ as an example.

```
// calling
auto [value, status] = valstat_enabled_function ();

// step one: capturing the one
// of four possible states
  if (   value &&   status )  { /* info */ }
  if (   value && ! status )  { /* ok   */ }
  if ( ! value &&   status )  { /* error*/ }
  if ( ! value && ! status )  { /* empty*/ }
```

VALSTAT Protocol return handling code is neatly divided in two steps. VALSTAT handling logic serves well for arriving to cleaner idioms for complex call consumption code.

3.5. The VALSTAT Responder Behavior

From the callable entity, an VALSTAT Responder is "signaling back" following the encapsulated application logic; creating and returning the appropriate VALSTAT structure.

The VALSTAT structure pseudo code declaration.

```
begin record VALSTAT
     field VALUE
     field STATUS
eof record
```

RESPONDER behavior consists of creating the appropriate VALSTAT record. RESPONDER signaling the "OK" state:

```
// pseudo code
return VALSTAT
// occupied value field and empty status field
     VALUE = (value), STATUS = (NULL)
eof return
```

Both `value` and `NULL` are meaningful software entities, as defined by their application domain. RESPONDER signaling the "INFO" state

```
return VALSTAT
// occupied value field and occupied status field
     VALUE = (value), STATUS = (information)
eof return
```

RESPONDER signaling the "ERROR" state

```
return VALSTAT
// empty value field and occupied status field
     VALUE = (NULL), STATUS = (information)
eof return
```

RESPONDER signaling the "EMPTY" state

```
return VALSTAT
// empty value field and empty status field
     VALUE = (NULL), STATUS = (NULL)
eof return
```

4. The Logic

4.1. Why the "two"?

Context depends on the domain. As ever in the information systems, the meaning of the information is context specific.

VALSTAT is a protocol for light **and** efficient information passing.

VALSTAT is not a messaging protocol. Contrast it to some protocol using (for example) XML (or JSON) documents for functions calling and function return values. Author is not aware of such a protocol. But anything between that and VALSTAT record of two fields, is not as light and not as efficient as VALSTAT structure.

And going bellow the two, degenerates back to the "single error value return" anti-pattern.

4.2. IT landscape matters

VALSTAT protocol value lies in it's deliberate simplicity, still capable aiding in solving the software operational and interoperability issues.

Modern software architectures are clusters of inter-operating but separated components and even sub-systems. One thorny big infrastructure issue, is solving universally applicable returns handling, across language and system barriers. And this is where VALSTAT as a protocol might help. (think JSON format for VALSTAT protocol)

Universal adoption of the VALSTAT requires no changes in any of the software development languages. It just has to be universally adopted and used.

5. Conclusions

Hopefully proving the benefits of evolution of error code handling into returns handling protocol does not need much convincing.

There are many real situations where the VALSTAT protocol can be successfully used. From a micro to the macro level.

As an common returns protocol, VALSTAT state requires to be ubiquitously adopted to become truly an instrumental to widespread interoperability. From micro to macro levels. From inside the code to inter component calls. From inside an project to inter systems situations.

VALSTAT protocol is multilingual in nature. Thus adopters from any imperative language are free to implement it in any way they wish too. For non coherent groups of adopters, the key requirement is: interoperability.

Authors primary aim is to suggest widespread adoption of this paradigm. As shown VALSTAT is more than just solving the "error-signalling problem". It is an paradigm, instrumental in solving the often hard and orthogonal set of run-time operational requirements.

VALSTAT aims high. And it's proposed scope is rather wide. But it is a humble protocol. It is just an simple and effective way of building bridges over deeply fragmented parts of the vast global IT infrastructure. While in the same time imposing extremely little on adopters implementations and leaving the non-adopters to "proceed as before" if they wish too.

Obstacles to VALSTAT adoption are far from just technical. But here is at least an immediately usable attempt to chart the way out.

---

6. References

- [FIELD] Field (computer science), https://en.wikipedia.org/wiki/Field_(computer_science)
- [DATA_FIELD] Data Fields, http://www.sliccware.com/WebHelp/Load_Definition/Definitions/Data_Fields/Data_Fields.htm
- [STROUSTRUP] B. Stroustrup (2018) **P0976: The Evils of Paradigms Or Beware of one-solution-fits-all thinking**, https://www.stroustrup.com/P0976-the-evils-of-paradigms.pdf
- [EMPTY_STRING] Wikipedia **Empty String**, https://en.wikipedia.org/wiki/Empty_string

- [EMPTY] "Your Dictionary" **Definition of empty**, https://www.yourdictionary.com/empty

- [EXCEPTIONS] Exception handling, https://en.wikipedia.org/wiki/Exception_handling