OCPP 2.0

Part 1 - Errata, v1.0

Table of Contents

1. Disclaimer

Version History

Version	Date	Author	Description
1.0	2019-12-20	Milan Jansen, Franc Buve and Robert de Leeuw	Version 1.0
Release Candidate	2019-12-01	Milan Jansen, Franc Buve and Robert de Leeuw	Release Candidate.

2. Scope

This document contains erratas on the OCPP 2.0 specification.

2.1. Terminology and Conventions

Bold: when needed to clarify differences, bold text might be used.

3. Errata

3.1. Page 4, Section 3. Information Model, Unclear what the Common: prefix means in part 2 - Specification, chapter Datatypes

The text below needs to be added to section 3.

In part 2 - Specification, chapter Datatypes, some DataTypes have the Common: prefix. This originates from the Information Model. It means that the DataType is able to be shared among other DataTypes and Messages. This has no impact on the OCPP implementation of a device.

3.2. Page 6, Section 4.3. Characteristics and Attributes, Unclear how to use Variable attributes

Section 4.3 describes what Variable attributes are. The below text and example needs to be included to the end of the section.

There is a difference between how to implement (physical) devices and (virtual) controller components, using the DeviceModel. A (virtual) controller component has to be implementing as described in part 2 chapter the "Referenced Components and Variables". These kind of components/variables are only using the variableAttribute type 'Actual'. Depending on if this variableAttribute is writable, the CSMS can use this to set a new value. (Physical) devices are a bit more complex to implement. For example, there is a fan with a fan speed, that has a (physical) limit with a range of 0 - 1000. But it should not be allowed to set the value below 200, because the fan can stop functioning. And it should not be set above 500, because that would be bad for the fan on the long run. When implementing this device using the DeviceModel, it can be defined as follows:

Component	**name**	Fan	
Variable	**name**	FanSpeed	
	variableAttribute 1	**type**	Actual
		value	<The current fan speed value of the fan.>
		mutability	ReadOnly
	variableAttribute 2	**type**	Target
		value	<The CSMS can use this value to adjust the fan speed. The Charging Station SHALL try to keep the actual value at the target value.>
		mutability	ReadWrite
	variableAttribute 3	**type**	MaxSet
		value	<The value '500' from the example. The target may not be set above this value.>
	variableAttribute 4	**type**	MinSet
		value	<The value '200' from the example. The target may not be set below this value.>
	variableCharacteristics	**maxLimit**	<The value '1000' from the example. This could be the physical max limit of the fan.>
		minLimit	<The value '0' from the example. This could be the physical min limit of the fan. This could also be -1000, if the fan is also able to rotate in the other direction.>
Description	This is an example of how a fan could be defined using the DeviceModel.		

When trying to set the target with value 600, the Charging Station will first check the allowed min and max values/limits and reject the set. If the target value is set to 500, the value is within range and the Charging Station will allow the set and start to adjust the actual fan speed. If the actual fan speed is measured to be 502, it's out of range. But it should be reported to the CSMS, so the actual value of a physical component should be updated without checking the min and max values/limits.

3.3. Page 7, Section 4.4. Monitoring, It is not clear which monitorTypes can be set on certain dataTypes

Section 4.4 describes the Monitoring feature. To make more clear what MonitorType/dataType combinations are possible, below matrix and text needs to be added to the section.

	string	**decimal**	**integer**	**dateTime**	**boolean**	**OptionList**	**SequenceList**	**MemberList**
UpperThreshold		X	X					
LowerThreshold		X	X					
Delta	X	X	X	X	X	X	X	X
Periodic	X	X	X		X	X	X	X
PeriodicClockAligned	X	X	X		X	X	X	X

For *UpperThreshold* and *LowerThreshold* this value represents the to be exceeded value by the actual value of the variable.

For *Delta* this value represents the change in value comparing with the actual value from the moment the monitor was set. - When the dataType of the variable is integer or decimal, this value represents the difference to be reached to trigger the monitor. - When the dataType of the variable is dateTime the unit of measure will be in seconds. - When the dataType of the variable is string, boolean, OptionList, SequenceList or MemberList, this value is ignored. The monitor will be triggered by every change in the actual value. When a delta monitor is triggered OR when the Charging Station has rebooted, the Charging Station shall set a new momentary value.

For *Periodic* and *PeriodicClockAligned* this value represents the interval in seconds.