Implementation Guide

for Minimum Required Error Codes in Electric Vehicle Charging Infrastructure

SEPTEMBER 2023

Working Group 3: Solutions for Scaling Reliability
Diagnostics Taskforce

CHARGEX consortium

Argonne NATIONAL LABORATORY

INL Idaho National Laboratory

NREL NATIONAL RENEWABLE ENERGY LABORATORY

INL/RPT-23-74833

This report was prepared for the U.S. Department of Energy under DOE Idaho Operations Office contract no. AC07-05ID14517. Funding was provided by the Joint Office of Energy and Transportation.

Revision Log:

Version Date	Description	Author
09.25.2023	Preliminary version released to members of the ChargeX WG3	All authors and contributors
09.28.2023	Updates to Tables 1,2, and 3	Mayuresh Savargaonkar, Kaleb Houck
09.29.2023	Public release version	All authors and contributors

Table of Contents

List of Abbreviations

Abbreviation	Description
CS	Charging Station, can consist of one or more electric vehicle supply equipment
CSO	Charging Station Operator, also referred to as a Charge Point Operator
CSMS	Charging Station Management System
EV	Electric Vehicle
EVSE	Electric Vehicle Supply Equipment
JSON	JavaScript Object Notation
MREC	Minimum Required Error Code
OCPP	Open Charge Point Protocol

Authors:

Mayuresh Savargaonkar[1] , Kaleb Houck, Benny Varghese (Idaho National Laboratory), and Bryan Nystrom (Argonne National Laboratory)

[1] mayuresh@inl.gov

1. Introduction

A. Objective

With the growing adoption of electric vehicles (EVs), there is an increased need for a reliable EV charging infrastructure. To help meet this need, the ChargeX report "Recommendations for Minimum Required Error Codes for Electric Vehicle Charging Infrastructure"[2] recommends a set of minimum required error codes (MRECs) and their functional and responsibility classifications. EV supply equipment (EVSE) manufacturers, charging station operators (CSOs), EV manufacturers, and other stakeholders in the North American market are encouraged to uniformly adopt the MRECs to enhance EV charging error reporting, interpretation, and diagnostics. This document serves as a guide to enable uniform implementation of the MRECs using the open charge point protocol (OCPP).

B. Scope

This implementation guide offers an in-depth analysis and association of the MRECs defined in "Recommendations for Minimum Required Error Codes for Electric Vehicle Charging Infrastructure" with specific messages in OCPP versions 1.6J[3] and 2.0.1.[4] This guide dives deeper into the MRECs proposing *fault codes* while providing examples for the structure, essential fields, and other pertinent details as dictated by OCPP. This report also includes sample JSON packets, demonstrating how MRECs can be transmitted in illustrative scenarios.

The objective of this report is to provide a guide that accelerates the MREC adoption proposed by the ChargeX consortium, ensuring consistent error reporting, interpretation, and diagnostics across the North American EV charging landscape. This guide builds upon the MRECs report to improve EV charging reliability by streamlining error reporting and transmission processes via OCPP.

2. Implementation of MRECs

Considering the character restrictions in OCPP message data fields, this report recommends the use of specific *fault codes* for the transmission of each MREC to ensure clarity and precision. Uniquely, each *fault code* mentioned in this report begins with 'CX,' signifying its association to the set of MRECs proposed by the ChargeX consortium. The subsequent three digits uniquely identify each error code. As an illustration, the fault code 'CX001' is designated

[2] https://inl.gov/content/uploads/2023/07/ChargeX_MREC_Rev5_09.12.23.pdf
[3] https://www.openchargealliance.org/protocols/ocpp-16/
[4] https://www.openchargealliance.org/protocols/ocpp-201/

for the first error code (ConnectorLockFailure) within the set of MRECs proposed by the ChargeX consortium.

Table 1 can be used to map the 26 MRECs and their descriptions given in "Recommendations for Minimum Required Error Codes for Electric Vehicle Charging Infrastructure" to their individual *fault codes*.

Table 1. Fault codes for MRECs.

Number	Error Code Name	Fault Code	Error Code Listed in OCPP 1.6?
1	ConnectorLockFailure	CX001	✓
2	GroundFailure	CX002	✓
3	HighTemperature	CX003	✓
4	OverCurrentFailure	CX004	✓
5	OverVoltage	CX005	✓
6	UnderVoltage	CX006	✓
7	WeakSignal	CX007	✓
8	EmergencyStop	CX008	-
9	AuthorizationTimeout	CX009	-
10	InvalidVehicleMode	CX010	-
11	CableCheckFailure	CX011	-
12	PreChargeFailure	CX012	-
13	NoInternet	CX013	-
14	PilotFault	CX014	-
15	PowerLoss	CX015	-
16	EVContactorFault	CX016	-
17	EVSEContactorFault	CX017	-
18	CableOverTempDerate	CX018	-
19	CableOverTempStop	CX019	-
20	PartialInsertion	CX020	-
21	CapacitanceFault	CX021	-
22	ResistanceFault	CX022	-
23	ProximityFault	CX023	-
24	ConnectorVoltageHigh	CX024	-
25	BrokenLatch	CX025	-
26	CutCable	CX026	-

The utilization of *fault codes,* as outlined in Table 1, brings several benefits:

1. With their unique format, these *fault codes* (and consequently MRECs) seamlessly blend into any pre-existing list of custom error codes a stakeholder may already use.
2. Each *fault code*, limited to just five characters, allows for the efficient and compact transmission of multiple MRECs.
3. Future adjustments and expansions to this list are easily managed. A particular *fault code*, if needed, can be designated as 'deprecated,' ensuring no disruption to existing configurations.
4. Each *fault code* is directly linked to its likely error source and the impacted functionality, aligning with the responsibility and functional classifications given in the MRECs report.

A. OCPP 1.6J

OCPP 1.6J is a commonly implemented version of the OCPP, designed to facilitate the management and operation of a charging station (CS). It employs two primary mechanisms to ensure CS's availability: (1) the *websocket ping* mechanism, and (2) the configurable *heartbeat interval*. These mechanisms are crucial for the CSMS to ascertain the operational status and availability of the CS. In addition to these, OCPP 1.6 enables the charging station to actively report errors as they occur using the *StatusNotification* message. A notable aspect of OCPP 1.6 is its ability to support customization in error reporting. This is achieved by incorporating custom error codes within the *vendorErrorCode* field of the *StatusNotification.req* message, which enables a more detailed error reporting process. It is recommended to transmit MRECs using the *vendorErrorCode* field in the *StatusNotification.req* message housed within OCPP 1.6J. Table 2 outlines all the data fields housed within the *StatusNotification.req* message that need to be modified to transmit MRECs using OCPP 1.6J. The *Suggested Data* and *Description* fields provide additional guidance for implementing the MRECs. OCPP 1.6J documentation contains additional details on the *StatusNotification.req* message, its JSON schema, and other data fields and their data housed within it.

Table 2. Data fields within the *StatusNotification.req* that need to be modified for the transmission of MRECs via OCPP 1.6J.

Data Field	Suggested Data	Description
errorCode	ChargePointErrorCode	If available, suitable *ChargePointErrorCode* in OCPP 1.6 should be reported here. If not available, 'OtherError' should be used.
info	Actual Value	This includes additional information related to the error. The actual or observed value is recommended to be reported here and can be left blank if no information is available. More

		details of the error can be included in this field separated by the ";" delimiter.
vendorId	{vendor specific information}; https://chargex.inl.gov	This identifies the vendor-specific implementation. It is highly recommended to use https://chargex.inl.gov to help provide the end-user easier access to the details of the reported MREC, including its description and responsibility and functional classifications using the ";" delimiter.
vendorErrorCode	Fault code	*Fault code* for a specified MREC as given in Table 1. Multiple errors can be separated by the ";" delimiter.

An example *StatusNotification.req* message for a *ChargePointErrorCode* *outlined in the OCPP 1.6* is as follows:

{
"connectorId": 1,
"errorCode": "HighTemperature",
"info": "50",
"status": "Finishing",
"timestamp": "2023-09-06T00-08-09Z",
"vendorId": "https://chargex.inl.gov",
"vendorErrorCode": "CX003"
}

The above example showcases a hypothetical "HighTemperature" (fault code: CX003) scenario where the temperature inside the CS is 50 degrees Celsius. Since the *vendorErrorCode* is given as 'CX003' along with the *vendorId* as 'https://chargex.inl.gov', the end user should use the ChargeX documentation to identify and interpret the error. All MRECs in Table 1 that are cross listed in OCPP 1.6 should be reported in this way. Using this method allows the stakeholder to use ChargeX's MRECs without interfering with OCPP's certification process.

It is important to note that the thresholds for reporting these errors and their actual values along with their units may be configured by individual stakeholders based on their implementation or existing standards.

An example of the *StatusNotification.req* message for a *ChargePointErrorCode* *not outlined in OCPP 1.6* is as follows:

{

"connectorId": 1,
"errorCode": "OtherError",
"info": "105",
"status": "Faulted",
"timestamp": "2023-09-06T00-08-09Z",
"vendorId": "https://chargex.inl.gov",
"vendorErrorCode": "CX019"
}

The above example showcases a hypothetical “CableOverTempStop” (fault code: CX019) scenario where the cable temperature is 105 degrees Celsius. Since the *vendorErrorCode* is given as ‘CX019’ along with the *vendorId* as ‘https://chargex.inl.gov’, the end user should use the ChargeX documentation to identify and interpret the error.

Next, there may be some cases where no information is available. In this case, we recommend leaving the *info* data field blank.

An example *StatusNotification.req* message for a *fault without info* is as follows:

{
"connectorId": 1,
"errorCode": "OtherError",
"info": "",
"status": "Faulted",
"timestamp": "2023-09-06T00-08-09Z",
"vendorId": "https://chargex.inl.gov",
"vendorErrorCode": "CX026"
}

The above example showcases a hypothetical “CableCut” (fault code: CX026) scenario where the charging cable has been severed. Since there is no actual value to be reported, in this case, the *info* data field is left blank. Since the *vendorErrorCode* is given as ‘CX026’ along with the *vendorId* as ‘https://chargex.inl.gov’, the end user should use the ChargeX documentation to identify and interpret the error.

In many practical cases, it is possible that multiple faults need to be reported simultaneously. When reporting multiple simultaneous faults, it is recommended that the *info* data field and the *vendorErrorCode* data field report multiple values in the same sequence separated by the “;” delimiter. An example *StatusNotification.req* message for *multiple faults* is as follows:

{
"connectorId": 1,
"errorCode": "OtherError",
"info": "50;105",
"status": "Faulted",
"timestamp": "2023-09-06T00-08-09Z",
"vendorId": "www.vendor.com;https://chargex.inl.gov",
"vendorErrorCode": "CustomError;CX019"
}

As shown in the example above, it is possible to report the MRECs in combination with an existing list of vendor-specific error codes. Therefore, while the "CustomError" points to a vendor-specific error code with a value of "50" units, *fault code* "CX019" points to the "CableOverTempStop" error with a value of "105" degrees Celsius. We recommend relaying multiple *vendorIds* if the errors are mapped to different error lists. With the 'CX' prefix and 'https://chargex.inl.gov' included in the *vendorId*, the end user should be well-informed to use the ChargeX documentation to identify and interpret this error.

Finally, it should be noted that the {vendor specific information} field(s) in Table 2 may vary and should be populated by the individual stakeholder (CSO or EVSE manufacturer), as mandated by the OCPP 1.6J standard.

B. OCPP 2.0.1

OCPP 1.6 has some limitations in error reporting and diagnostics. While the core functionalities defined in OCPP 1.6 persist in the latest OCPP version (OCPP 2.0.1), the *websocket ping* and *heartbeat interval* mechanisms still serve as vital checks for a CS's availability. However, the error reporting mechanism has been updated. OCPP 2.0.1 relies on a device model that presents a holistic and granular view of the structure and functionality of a CS. Unlike the more straightforward structure in OCPP 1.6, the device model in OCPP 2.0.1 is hierarchical. This means that the CS is visualized not only as a monolithic entity but also as an assembly of interrelated components. This representation mirrors the physical and functional structure of the CS, making error identification, reporting, and resolution more intuitive. Each component, whether it is a connector, a meter, or a temperature sensor, has associated variables. These variables capture the operational data and states of the respective components. When a fault occurs, the specific variable associated with a component can be identified, thereby enabling a direct correlation between the error and its source. This component-variable structure permits more robust monitoring capabilities. With the ability to pin errors down to specific components and variables, diagnostics become significantly more detailed.

Given these advancements, OCPP 2.0.1 uses the *NotifyEvent* message as the main conduit for transmitting errors from the CS to the CSMS. Therefore, this implementation guide recommends using the *techCode* data field in the *NotifyEventRequest* message of OCPP 2.0.1 for transmitting the MRECs. Table 3 highlights the data fields within the *NotifyEventRequest* message that need to be modified for transmitting MRECs. For an exhaustive understanding of the *NotifyEventRequest* message, including its JSON schema and associated data fields, please refer to the OCPP 2.0.1 documentation.

Table 3. Data fields within the *NotifyEventRequest* message that need to be modified for transmission of MRECs via OCPP 2.0.1

Data Field	Suggested Data	Description
techCode	Fault code	*Fault code* as defined in Table 1.

OCPP 2.0.1 adaptability ensures that it can be customized to fit vendor-specific requirements and their unique device model implementations. Therefore, *we do not mandate any specific device model* or *component-variable-monitor combination* in this report. The examples provided below draw from a conceptual device and information model and are intended to provide example component-variable-monitor pairings. It is important to note that while some of these sample messages mention monitors, a majority of the 26 MRECs can be reported without configuring monitors, which expedites their way into real-world implementations of OCPP 2.0.1.

An example of the *NotifyEventRequest* message for a “ConnectorLockFailure” (fault code: CX001) *fault without a monitor* is as follows:


```
{
 "generatedAt": "2023-09-06T00-08-09Z",
 "tbc": false,
 "seqNo": 0,
 "eventData": [
    {
        "eventId": 1,
        "timestamp": "2023-09-06T00-08-09Z",
        "trigger": "Delta",
        "actualValue": "true",
        "cause": "",
        "techCode": "CX001",
        "techInfo": "Additional information",
```

```
        "cleared": false,
        "transactionId": "12345",
        "variableMonitoringId": "",
        "eventNotificationType": "HardWiredNotification",
        "component": {
              "name": "EVRetentionLock",
              "instance": "Main",
              "evse": {
                  "id": 1,
                  "connectorId": 1}
      },
        "variable": {
        "name": "Problem",
        "instance": "Main"}
   }
 ]
}
```


A sample *NotifyEventRequest* message for “HighTemperature” (fault code: CX001) *fault with a monitor* is as follows:


```
{
 "generatedAt": "2023-09-06T00-08-09Z",
 "tbc": false,
 "seqNo": 0,
 "eventData": [
  {
   "eventId": 1,
   "timestamp": "2023-09-06T00-08-09Z",
   "trigger": "Alerting",
   "actualValue": "50",
   "cause": "",
   "techCode": "CX003",
   "techInfo": "Additional information",
   "cleared": false,
   "transactionId": "12345",
   "variableMonitoringId": 1,
   "eventNotificationType": "HardWiredMonitor",
   "component": {
    "name": "TemperatureSensor",
```

```
      "instance": "Main",
      "evse": {
       "id": 1,
       "connectorId": 1
      }
     },
     "variable": {
      "name": "Temperature",
      "instance": "Main"
     }
    }
  ]
}
```


The example above shows a “HighTemperature” (fault code: CX003) error while reporting that the temperature inside the CS is 50 degrees Celsius. In this case, the monitor set on the *Main TemperatureSensor* located within the CS is triggered due to the high temperature inside the CS. In this illustrative example, we assume that a few *NotifyEventRequest* messages have been sent to notify the CSMS that report the temperature inside the CS has been rising from various instances (or sensing points) of *TemperatureSensor* within the CS.

The *NotifyEventRequest* message within OCPP 2.0.1 is well-equipped to simultaneously handle multiple error codes. Below is an example *NotifyEventRequest* message with *multiple faults*:


```
{
 "generatedAt": "2023-09-06T00-08-09Z",
 "tbc": false,
 "seqNo": 0,
 "eventData": [
  {
   "eventId": 100,
   "timestamp": "2023-09-06T00-08-09Z",
   "trigger": "Delta",
   "actualValue": "true",
   "cause": "",
   "techCode": "CX026",
   "techInfo": "Additional information",
   "cleared": false,
```

```
  "transactionId": "12345",
  "variableMonitoringId": "",
  "eventNotificationType": "HardWiredNotification",
  "component": {
   "name": "CableBreakawaySensor",
   "instance": "Main",
   "evse": {
    "id": 1,
    "connectorId": 1
   }
  },
  "variable": {
   "name": "Tripped",
   "instance": "Main"
  }
 },
 {
  "eventId": 101,
  "timestamp": "2023-09-06T00-08-10Z",
  "trigger": "Delta",
  "actualValue": "120",
  "cause": 100,
  "techCode": "CX019",
  "techInfo": "Additional information",
  "cleared": false,
  "transactionId": "12345",
  "variableMonitoringId": 2,
  "eventNotificationType": "HardWiredMonitor",
  "component": {
   "name": "CableTemperatureSensor",
   "instance": "Main",
   "evse": {
    "id": 1,
    "connectorId": 1
   }
  },
  "variable": {
   "name": "Active",
   "instance": "Main"
  }
```

```
    }
  ]
}
```

In the example above, it appears that a severed cable (fault code: CX026) caused the *CableTemperatureSensor* in the cable to exceed the upper threshold of a monitored variable in the connector cable, resulting in a "CableOverTempStop" error (fault code: CX019). This can be inferred from the hierarchical order in which the example message above has been relayed with cause in the second *eventData,* pointing to the *eventId* of the first *eventData*.

3. Conclusion

In recent years, the transition to EVs has underscored the critical importance of a robust charging infrastructure. "Recommendations for Minimum Required Error Codes for Electric Vehicle Charging Infrastructure" provided essential groundwork by identifying a minimum set of error codes along with their responsibility and functional classifications to enhance the reliability of this infrastructure. Building on that foundation, this report presents a detailed framework for integrating and transmitting MRECs in the form of *fault codes* via existing OCPP implementations. The practical insights and illustrative examples in this report are intended to ease MRECs incorporation into any pre-existing, vendor-specific OCPP implementations.