OCPP & California Pricing Requirements

v3.1, 13-09-2024

Table of Contents

OCA Application Note

Relevant for OCPP version: 1.6 and 2.0.1.

Version History

Version	Date	Author	Description
1.0	2020-07-14	Franc Buve *(OCA)*	1.0 version
2.0	2021-09-17	Franc Buve *(OCA)*	Extended with support for real-time cost calculation on the charger.
2.1	2023-06-05	Franc Buve *(OCA)*	Fixed minor typos/inconsistencies in JSON examples
3.0	2024-02-22	Franc Buve *(OCA)*	Added multi-language support
3.1	2024-09-13	Franc Buve *(OCA)*	Fixed error in par 3.2 where a configuration variable was referred to as DefaultPrice instead of DefaultPriceText. Fixed fields in TransactionEventResponse (UpdatedPersonalMessage) Removed "idlePrice" from DefaultPrice in sequence diagrams.

1. Introduction

The National Conference on Weights and Measures in the USA has issued a document [1] with checklists and test procedures for electric vehicle fueling systems, which has been adopted by the California Division of Measurement Standards (DMS). Many requirements in the checklists apply to capabilities of the charging station (e.g. having a display or how to display prices), but in some cases it poses requirements for information that is being exchanged between charging station and CSMS (back-office).

This document explains how price and cost information can be communicated to the customer with OCPP 1.6 by creating a few custom extensions. It continues to show how this can be achieved in OCPP 2.0.1 with standard messages. A customization of a single message is required if real-time cost calculation on the charger is needed.

Section Checklist Items Requiring Attention summarizes the items for which no direct support exists in OCPP and offers alternative solutions. Finally, the Appendix: Relevant Requirements per Section lists all requirements from the checklist that are relevant to OCPP.

Based on experiences from first implementations in the field this document has been updated. The running cost information is now sent in a structured JSON-format, which will result in more standardized implementations. The new format also enables the charging station to calculate and display a running total in between the periodic cost updates that it receives from the back-end.

2. Cost Calculation in OCPP

Since tariffs can potentially be quite complex with prices depending on things like time of day, power, amount of energy and type of contract, OCPP has adopted the approach of letting the CSMS calculate the cost and communicate this to the charging station. In a situation where CPO and eMSP are not the same party, we have to deal with two different prices: a wholesale price that CPO is charging to the eMSP and a retail price of eMPS to the customer. It is the retail price that has to be communicated on the charging station. Note, that the eMSP retail price can vary from a simple surcharge on the wholesale price to something completely unrelated, like a flat rate. This is something that can only be provided by the CSMS, because the charging station does not have access to this data.

Upon each meter value that the charging station sends, CSMS will send back the calculated running cost of the transaction. This means that the running cost on the display will only be updated after each meter interval. Typically, meter values are sent every 30 to 60 seconds on a fast charger (level 3) and every 5 to 15 minutes on a regular (level 2) charging station. This update frequency is too low to pass CTEP [2] certification. To remedy this situation we provide the charging station with the unit prices that are needed to calculate and show the running cost locally until the next cost update from CSMS arrives.

3. Tariff and Cost Features in OCPP 1.6

OCPP 1.6 does not provide any support for the communication of cost or prices. Therefore, extensive customization is needed on both Central System and charge point [3] to transfer the required data. These customizations on both ends need to be perfectly aligned, or it will not work.

3.1. Displaying Price and Cost in OCPP 1.6

It is possible to use a ChangeConfiguration message to set a default price to be displayed at a charge point. This mechanism is, however, not suitable to display a running cost or final cost message, since these messages depend on the transaction in progress and must only be shown to the user of the transaction and at the right connector.

A better method is therefore to use DataTransfer messages to convey the pricing information to the charge point. A way to implement this, is to use a configuration key for the default price and create a custom SetUserPrice and a RunningCost and FinalCost message in OCPP 1.6 using DataTransfer. This makes it possible to link the pricing information to the user or the transaction.

It is good practice to define a boolean configuration key ("CustomDisplayCostAndPrice"), that a Central System can query to find out if this customization is present and optionally use it to enable or disable the customization:

```
ChangeConfiguration.req( "CustomDisplayCostAndPrice", "true" )
```

DataTransfer messages need to have a vendor ID to identify the customization. We advise to use the value "org.openchargealliance.costmsg" to identify the customizations that are described in this document.

The pricing messages are described in the following sections.

NOTE	Backslashes that are required to escape quotation marks within a field have been omitted from the command examples for readability.

3.1.1. Default Price

The default price is displayed when the user has not yet been identified, because authorization has not taken place. Setting of the default price is only needed once, as long as it does not change, and can therefore be done via a configuration variable. This variable is set with a JSON string with three data fields.

The field **priceText** holds a text to show on the display. The optional field **priceTextOffline** can be used to display a different pricing message when the charge point is offline. An optional third field **chargingPrice** can contain the pricing components that the charge point should use in case a transaction is started while the charge point is offline and cannot receive RunningCost messages. (See Sequence diagrams for (partly) offline situations).

There is no "idlePrice" component in the default price, because when the charge point is offline, the Central System cannot notify the charge point that it should start using idle price. See Idle Time During Transaction for more information on idle time.

An example message for DefaultPrice with pricing information to use when offline:

```
 ChangeConfiguration.req( "DefaultPrice", "{
    "priceText": "0.15 $/kWh, idle fee after charging: 1 $/hr",
    "priceTextOffline": "The station is offline. Charging is possible for
0.15 $/kWh.",
    "chargingPrice": { "kWhPrice": 0.15, "hourPrice": 0.00, "flatFee": 0.00 }
    }" )
```

An example message for DefaultPrice when no cost is calculated when offline:

```
 ChangeConfiguration.req( "DefaultPrice", "{
    "priceText": "0.15 $/kWh, idle fee after charging: 1 $/hr",
    "priceTextOffline": "The station is offline. The charging is free-of-
charge.",
    }" )
```

Table 1. data fields in ChangeConfiguration.req for DefaultPrice

Field	Type	Card.	Description
priceText	string	1..1	Text for display of price information.
priceTextOffline	string	0..1	Alternative text for display when charge point is offline.
chargingPrice	ChargingPrice	0..1	Structure with price components to use when starting a session while offline. Not needed if offline sessions are not allowed or not charged.

3.1.2. User-specific Price

When the user has been identified, Central System can send the user-specific price. It is linked to the authorization token. This may be a different price than the default price. The user-specific price message is for display only. It has no "chargingPrice" and "idlePrice" fields, because Central System will send a RunningCost message immediately after the StartTransaction message.

```
 DataTransfer.req( "vendorId": "org.openchargealliance.costmsg",
 "messageId": "SetUserPrice",
 "data": "{
    "idToken": "12345678",
    "priceText": "$0.12/kWh, no idle fee"
    }" )
```

Table 2. data fields in DataTransfer.req for SetUserPrice

Field	Type	Card.	Description
idToken	string	1..1	idToken of the user to which this price applies.
priceText	string	1..1	Text for display of price and price components.

3.1.3. Final Cost

The following message is used to send the final cost. It is linked to the transaction and sent at the end of the transaction. The **cost** field contains the total cost, which the charge point might use to send to an integrated payment terminal if it is equipped with one. The **priceText** field contains the message for display at the charge point, which shows the price and its components. It can also contain a URL that points to a location where the user can retrieve an invoice. For convenience this URL can also be provided in the optional **qrCodeText** field. A charge point that supports it, can then display the URL as a QR code for easy scanning by the user.

```
DataTransfer.req( "vendorId": "org.openchargealliance.costmsg",
"messageId": "FinalCost",
"data": "{
   "transactionId": 98765,
   "cost": 3.31,
   "priceText": "$2.81 @ $0.12/kWh, $0.50 @ $1/h, TOTAL KWH: 23.4
       TIME: 03.50 COST: $3.31.
       Visit www.cpo.com/invoices/13546 for an invoice of your session.",
   "qrCodeText": "https://www.cpo.com/invoices/13546"
   }" )
```

Table 3. data fields in DataTransfer.req for FinalCost

Field	Type	Card.	Description
transactionId	integer	1..1	Transaction to which this applies.
cost	decimal	1..1	Calculated total final cost.
priceText	string	1..1	Text for display of price and price components.
qrCodeText	string	0..1	Optional URL to display as QR code.

3.1.4. RunningCost Message

The RunningCost message is used to display the price and cost during the transaction.

The Central System sends a DataTransfer message, called "RunningCost", immediately after sending the StartTransaction.conf response. This message also has the unit prices that the charging point needs to calculate the running cost locally, so that a real-time display of transaction cost can be shown to the customer.

Upon each meter value it receives from the charge point the Central System updates the running cost of the transaction and sends a RunningCost message, that holds the cost of the transaction up to the meter value that it just received.

The message has unit prices to be used when the EV is charging and optionally a unit price for an idle fee. It is up the Central System to decide if and when an idle fee shall be charged. This is indicated by the field "state", which can be "Charging" or "Idle". The price for charging is sent in the **chargingPrice** field; the price for idle in the **idlePrice** field.

If the prices change at a certain time of the day, then these can be added in the optional **nextPeriod** field. There may be other situations when the price might change, however. For that purpose a field **triggerMeterValue** can be added, which describes when, in addition to the regular sampled meter values, the charging point should

send a meter value. Upon receiving a meter value Central System will send a new RunningCost message which contains the updated price information. A **triggerMeterValue** can be set to a time of day, an energy amount or a power amount.

A full description of the JSON structure for the RunningCost message is shown below:

Table 4. data fields in DataTransfer.req for RunningCost

Field	Type	Card.	Description
transactionId	integer	1..1	Transaction to which this applies.
timestamp	dateTime	1..1	Timestamp of the meter value upon which this cost is based.
meterValue	integer	1..1	Meter value (Wh) upon which this cost is based.
cost	decimal	1..1	Calculated total running cost.
state	string	1..1	"Charging" or "Idle". Determines which pricing components are to be used.
chargingPrice	ChargingPrice	1..1	Price components while charging.
idlePrice	IdlePrice	0..1	Price components while not charging. Optional if no idle fee is charged.
nextPeriod	NextPeriod	0..1	Pricing for next period.
triggerMeterValue	Triggers	0..1	Triggers to request a new meter value.

Table 5. ChargingPrice

Field	Type	Card.	Description
kWhPrice	decimal	0..1	Price per kWh.
hourPrice	decimal	0..1	Price per hour of charging.
flatFee	decimal	0..1	Flat fee for (part of) charging session.

Table 6. IdlePrice

Field	Type	Card.	Description
graceMinutes	integer	0..1	Grace period in minutes before idle time is charged. Grace minutes start counting from the *timestamp* of the message in which *state* changed from "Charging" to "Idle".
hourPrice	decimal	0..1	Price per hour while idle.

Table 7. NextPeriod

Field	Type	Card.	Description
atTime	dateTime	1..1	Time when these prices become active.
chargingPrice	ChargingPrice	1..1	Price components while charging.
idlePrice	IdlePrice	0..1	Price components while idle. Optional if no idle fee charged.

Table 8. Triggers

Field	Type	Card.	Description
atTime	dateTime	0..1	Time when a meter value must be sent.
atEnergykWh	decimal	0..1	Consumed energy amount in kWh upon which a meter value must be sent.
atPowerkW	decimal	0..1	Power threshold in kW when meter value must be sent when crossing in downward or upward direction. Can either be used to trigger a meter value when vehicle stops charging or when vehicle charges at a high power that requires a different tariff. It is recommended to implement a hysteresis around this value to avoid repetitive triggers when the power fluctuates around this level.
atCPStatus	string	0..6	ChargePointStatus upon which a meter value must be sent. Values matching ChargePointStatus enumeration: *Available, Preparing, Charging, SuspendedEVSE, SuspendedEV, Finishing*

RunningCost (full example)

```
DataTransfer.req("vendorId": "org.openchargealliance.costmsg",
"messageId": "RunningCost",
"data": "{
   "transactionId": 12345,
   "timestamp": "2021-03-19T12:00:00Z", "meterValue": 1234000,
   "cost": 1.00,
   "state": "Charging",
   "chargingPrice": {
       "kWhPrice": 0.123, "hourPrice": 0.00, "flatFee": 0.00 },
   "idlePrice": { "graceMinutes": 30, "hourPrice": 1.00 },
   "nextPeriod: {
       "atTime": "2021-03-19T19:00:00Z",
       "chargingPrice": {
           "kWhPrice": 0.100, "hourPrice": 0.00, flatFee": 0.00 },
       "idlePrice": { "hourPrice": 0.00 }
   }
   "triggerMeterValue": {
       "atTime": "2021-03-19T23:00:00Z",
       "atEnergykWh": 50.0,
       "atPowerkW": 0.1,
       "atCPStatus": [ "SuspendedEV", "SuspendedEVSE" ]
   }
}" )
```

RunningCost (minimal example, only kWh price)

```
DataTransfer.req("vendorId": "org.openchargealliance.costmsg",
"messageId": "RunningCost",
"data": "{
   "transactionId": 12345,
   "timestamp": "2021-03-19T12:00:00Z", "meterValue": 1234000,
   "cost": 1.00,
   "state": "Charging"
   "chargingPrice": { "kWhPrice": 0.123 }
}" )
```

RunningCost (realistic example, kWh price, idle fee, future price change)

```
DataTransfer.req("vendorId": "org.openchargealliance.costmsg",
"messageId": "RunningCost",
"data": "{
   "transactionId": 12345,
   "timestamp": "2021-03-19T14:00:00Z", "meterValue": 1234567,
   "cost": 1.00,
   "state": "Idle",
   "chargingPrice": { "kWhPrice": 0.123 },
   "idlePrice": { "graceMinutes": 30, "hourPrice": 1.00 },
   "nextPeriod: {
       "atTime": "2021-03-19T19:00:00Z",
       "chargingPrice": { "kWhPrice": 0.100 },
       "idlePrice": { "graceMinutes": 30, "hourPrice": 1.00 }
   }
}" )
```

3.2. Multi-language support for tariffs (new in v3)

NOTE A charge point that provides multi-language support for tariffs will report the configuration key "CustomMultiLanguageMessages" as true.

The Central System can check if a charge point provides multi-language support as follows:

```
GetConfiguration.req( "CustomMultiLanguageMessages" )
```

This will return a value "true" when this customization is supported.

In order to minimize the impact of adding the multi-language customization, the existing **priceText** field remains valid, and a new field **priceTextExtra** is added to support different languages. The CTEP customization allows for a maximum of 4 languages to be sent in addition to the default language.

The Central System needs to be able to retrieve the list of languages that are supported by the UI of the charging station. To allow for this, a configuration keys is added through which a charging station conveys how many and which languages it can support. (Shown values are only an example).

Language codes shall be specified as IETF RFC5646 (as in OCPP 2.0.1).

```
ChangeConfiguration.req( "SupportedLanguages", "en,de,fr,nl,sv" )
```

This leads to the following overview of configuration variables for the multi-language support.

Table 9. Overview configuration keys

Configuration key	Type	ReadWrite/ ReadOnly	Description
CustomMultiLanguageMessages	boolean	RO	True when charging station supports multiple languages as per this document.
Language	string	RW	Default language code for the stations UI. Can be changed by CSMS.
SupportedLanguages	CSL	RO	Comma separated list of supported language codes, per RFC5646.

3.2.1. Multi-language default tariffs/pricing information

The default pricing is set as follows:

```
ChangeConfiguration.req( "DefaultPrice", "{
   "priceText": "0.15 $/kWh, idle fee after charging: 1 $/hr",
   "priceTextOffline": "The station is offline. Charging is possible for 0.15
$/kWh.",
   "chargingPrice": { "kWhPrice": 0.15, "hourPrice": 0.00, "flatFee": 0.00 }
   }" )
```

Fields **priceTextOffline** and **chargingPrice** have a cardinality [0..1], as they are only needed when supporting offline operations.

There are limitations on how long the value part of a configuration key can be, so we need multiple keys for multi-lingual support, one per language. The name of the key is "DefaultPriceText,<language code>". The standard configuration key "DefaultPrice" is still used to define the **chargingPrice** to ensure that the same pricing will be used for all languages.

```
ChangeConfiguration.req( "DefaultPriceText,<language code>", "{
   "priceText": "0.15 $/kWh, idle fee after charging: 1 $/hr",
   "priceTextOffline": "The station is offline. Charging is possible for
0.15$/kWh."
   }" )
```

where <language code> is the language code per RFC5646. Thus, we could for instance have the following configuration keys:

```
DefaultPrice                <-- text in default language and chargingPrice
DefaultPriceText,en-US      <-- text for English language in US
DefaultPriceText,fr-CA      <-- text for French language in Canada
DefaultPriceText,sv         <-- text for Swedish language
```

All depends on which languages we need to support, and if multiple variants of the same language are needed.

3.2.2. Multi-language driver/user-specific pricing

Driver/user specific pricing in the default language is sent as follows:

```
DataTransfer.req( "org.openchargealliance.costmsg", "SetUserPrice", "{
   "idToken": "12345678",
   "priceText": "€0.12/kWh, no idle fee"
   }" )
```

Since DataTransfer message does not have the size limitations of a configuration key, we can add multiple languages in the same message. When the configuration "CustomMultiLanguageMessages" was reported as "true", the DataTransfer.req will add the field **priceTextExtra**, which is a variant of *priceText* with *format*, *language* and *content* fields. **priceTextExtra** is used for text in other languages than the default language.

```
DataTransfer.req( "org.openchargealliance.costmsg", "SetUserPrice", "{
   "idToken": "12345678",
   "priceText": "GBP 0.12/kWh, no idle fee",
   "priceTextExtra": [{"format": "UTF8", "language": "nl",
                  "content": "€0.12/kWh, geen idle fee"},
                 {"format": "UTF8", "language": "de",
                  "content": "€0,12/kWh, keine Leerlaufgebühr"},
                  ...]
   }" )
```

The field **priceTextExtra** has the same format as MessageContentType in OCPP 2.0.1:

Field	Type	Card.	Description
format	string	1..1	One of "ASCII", "HTML", "UTF8"
language	string	1..1	Message language identifier. Contains a language code as defined in RFC5646.
content	string	1..1	Message text

The supported languages are constrained by the configuration keys described in Multi-language support for tariffs (new in v3).

3.2.3. Multi-language final cost

The **priceTextExtra** field for FinalCost is treated in exactly the same way as above.

```
DataTransfer.req( "org.openchargealliance.costmsg", "FinalCost", "{
   "transactionId": 98765,
   "cost": 3.31,
   "priceText": "GBP 2.81 @ 0.12/kWh, GBP 0.50 @ 1/h, TOTAL KWH: 23.4
        TIME: 03.50 COST: GBP 3.31.
        Visit www.cpo.com/invoices/13546 for an invoice of your session.",
   "priceTextExtra": [
        {"format": "UTF8", "language": "nl",
            "content": "€2.81 @ €0.12/kWh, €0.50 @ €1/h, TOTAL KWH: 23.4
            TIME: 03.50 COST: €3.31.
            Bezoek www.cpo.com/invoices/13546 voor een factuur van
            uw laadsessie."},
        {"format": "UTF8", "language": "de",
            "content": "€2,81 @ €0,12/kWh, €0,50 @ €1/h, GESAMT-KWH: 23,4
            ZEIT: 03:50 KOSTEN: €3,31.
            Besuchen Sie www.cpo.com/invoices/13546 um eine Rechnung
            für Ihren Ladevorgang zu erhalten.“}
        ]
   "qrCodeText": "https://www.cpo.com/invoices/13546"
   }" )
```

3.3. Timezone for Display

OCPP 1.6 does not have built-in support for timezones. It may be desirable to set a timezone when displaying time related to pricing information. The timezone of the charge point can be set using the following configuration variables:

TimeOffset:

```
ChangeConfiguration.req( "TimeOffset", "-05:00" )
```

NextTimeOffsetTransitionDateTime:

When to change to summer or winter time.

```
ChangeConfiguration.req( "NextTimeOffsetTransitionDateTime",
    "2021-03-28T02:00:00+01:00"
```

TimeOffsetNextTransition:

```
ChangeConfiguration.req( "TimeOffsetNextTransition", "-04:00" )
```

This is similar to how time offsets are set as device model variables in OCPP 2.0.1.

3.4. Idle Fees

NOTE	Idle fee calculation is not a DMS requirement.

If the pricing model needs to calculate an idle fee, i.e. charge a fee for being connected after charging has completed, then there are two different scenarios that can occur. There is idle time within the transaction and

idle time after the transaction has ended while the vehicle remains connected to the charge point. The first situation can easily be detected, because the received meter values do not change. Whereas the second situation can easily be handled in OCPP 2.0.1, this is not the case for OCPP 1.6, which was never designed to deal with this situation.

3.4.1. Idle Time During Transaction

DMS or CTEP do not define how idle time should be measured if an idle fee is charged. It is up to Central System and not the charge point, to decide when a transaction is considered to be in idle time. The reason for this is, that charging may have been suspended on request of Central System due to a demand response event or a smart charging algorithm. In that case it would not be fair to charge an idle fee to the customer.

Central System can detect that an EV is no longer charging when it sees that meter values stay the same. Depending on the meter value interval, it can take many minutes before it is noticed, unless the charge point sends a status notification *SuspendedEV* or *SuspendedEVSE*.

NOTE It is up to the CSO to decide whether both *SuspendedEV* and *SuspendedEVSE* are counted as idle time, or only *SuspendedEV* is considered idle time.

Detect idle by low power usage

Central System can request the charge point to send a meter value whenever the vehicle charges with less than a certain power by setting the **atPowerkW** trigger (e.g. at a value of 0.1 kW). If the charge point has been configured to send not only the measurand "Energy.Active.Import.Register", but also "Power.Active.Import", then Central System will immediately know from the power reading, that the EV has stopped charging. If only the register reading is provided, then Central System will have to wait until the next meter value to know that the EV has stopped charging. This is still quicker than via the regular meter intervals, though.

NOTE A meter value is triggered to be sent everytime the **atPowerkW** threshold is crossed in upward or downward direction. Some logic in the Central System is required to determine the direction in which the threshold is crossed.

If a charging suspension is caused by local load-balancing and not initiated by the Central System, then the Central System will not know, based on power consumption alone, whether the suspension is caused by EV or EVSE, unless it also monitors the StatusNotifications for *SuspendedEV* or *SuspendedEVSE*. In that situation it is convenient to use the **atCPStatus** trigger.

Detect idle by charge point status change

When a charge point goes into a *SuspendedEV* or *SuspendedEVSE* state, then Central System knows that charging has paused when it receives a StatusNotification.req. If the Central System needs to know the meter value at start of pausing, then the RunningCost messages should contain the **atCPStatus** trigger with the values *SuspendedEV* and *SuspendedEVSE*, in order to trigger the charge point to send a meter value at that moment.

Switch between Idle and Charging

When Central System detects that charging has paused, it will send a new RunningCost message with **state** set to "Idle". From that moment on, the charge point will charge the idle time according to the parameters in

idlePrice.

When the EV resumes charging, the charge point may send a status notification "Charging" or it will after some time send a meter value. (Sending of the meter value can be forced by setting the **atPowerkW** or **atCPStatus** triggers). Central System will then send a new RunningCost message with **state** set to "Charging" and the charge point will use the **chargingPrice** information to calculate cost to show on the display.

3.4.2. Idle Time When Transaction Has Ended

It is more difficult to detect idle time that occurs after the transaction has finished and the EV remains connected. A tempting solution in OCPP 1.6 would be to wait for a StatusNotification "Available" to signal that the vehicle has been unplugged. This is, however, not reliable. In the event that the charge point is offline when the transaction is stopped, the StatusNotifications may never reach Central System, since they are not required to be queued by the charge point.

3.4.3. Reliable Connector Unplug Message

Using the StatusNotification to detect unplugging is not always reliable, since this message is not guaranteed to be queued when the charge point is offline. A work-around for this is to define a DataTransfer message "ConnectorUnplugged", which the charge point sends to signal that the connector has been unplugged and the idle fee calculation can be stopped.

Such a DataTransfer message looks like this:

```
 DataTransfer.req( "vendorId": "org.openchargealliance.costmsg",
    "messageId": "ConnectorUnplugged",
    "data": "{"transactionId": 123456, "timestamp": "2020-06-01T12:34:00Z"}"
)
```

IMPORTANT	This DataTransfer "ConnectorUnplugged" message does not replace the existing StatusNotification.req message. The DataTransfer "ConnectorUnplugged" is sent as an additional reliable message that is queued as long as the charge point is offline, like other transaction-related messages.

A charge point still needs to send the StatusNotification.req when the connector becomes available, since a Central System might depend on it to report accurate connector status. It also ensures maximum compatibility in case a Central System does not support the new DataTransfer message.

A Central System normally ends a transaction upon receiving the StopTransaction.req message. If the customization to calculate an idle fee after a transaction has stopped (until connector is unplugged) has been implemented, then this must be reported in a boolean configuration key "CustomIdleFeeAfterStop" that reports "true" to the Central System, so that it knows, that it has to extend the transaction until receipt of the DataTransfer "ConnectorUnplugged".

```
 ChangeConfiguration.req( "CustomIdleFeeAfterStop", "true" )
```

This same configuration key can be used by the Central System to switch this behavior off, when it does not

support the extending of a transaction after stop.

3.5. Direct Payment

Normally, it is the eMSP that invoices the customer, but in the event of direct payment with credit or debit card or other means, then the CPO acts as an eMSP. In that case there must be a way to make a printable receipt available to the customer upon request. (See [EVSE_S.2.6])

Since a charge point is not normally equipped with a printer, the solution is to display a message that directs the customer to a website where the receipt can be downloaded.

3.6. Sequence diagrams showing the OCPP 1.6 customizations

The following diagram shows the sequence of events to report price and cost during a transaction.

3.6.1. Price calculation during transaction

Central System sends calculated cost and price for current and optionally next period. Charge point uses this to calculate intermediate cost values.

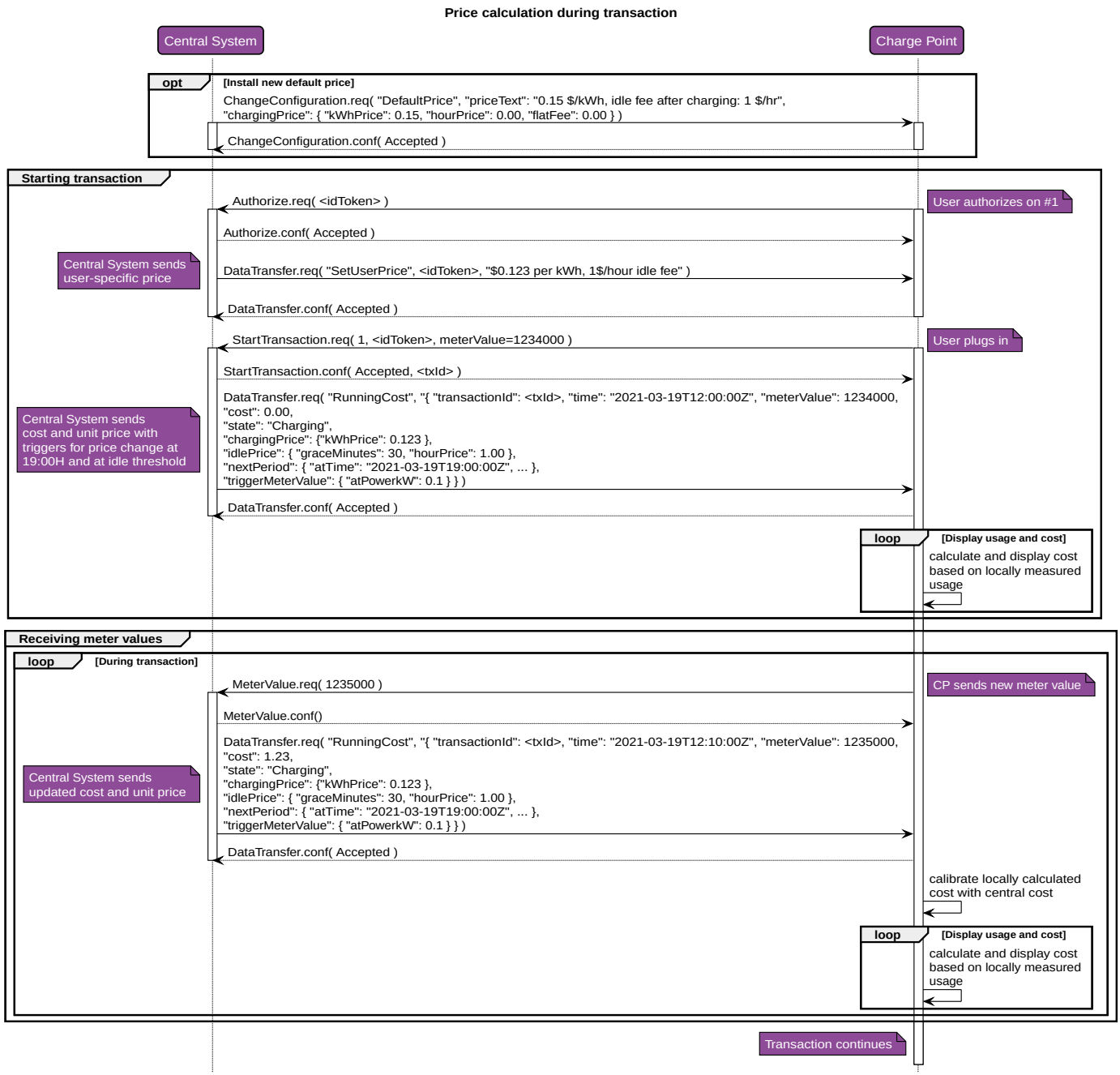

Figure 1. Price calculation during transaction

3.6.2. Price calculation when EV stops and resumes

When Central System discovers that EV has stopped charging, it sends a RunningCost for state 'Idle' with grace period, idle costs and charging costs. Charging costs are included so that Charge point can start counting again as soon as EV resumes charging.

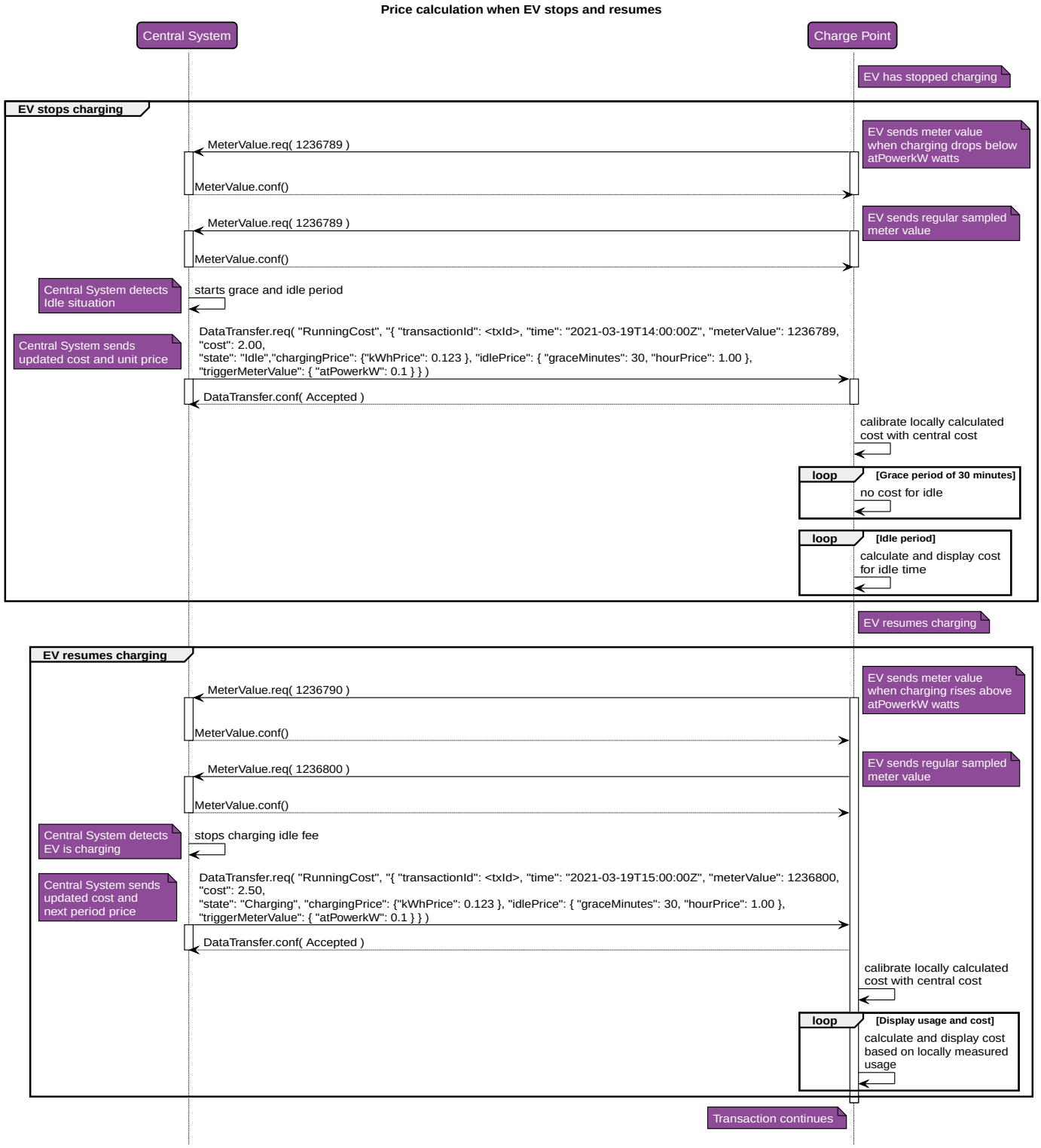

Figure 2. Price calculation when EV stops and resumes

3.6.3. Price calculation at stop of transaction

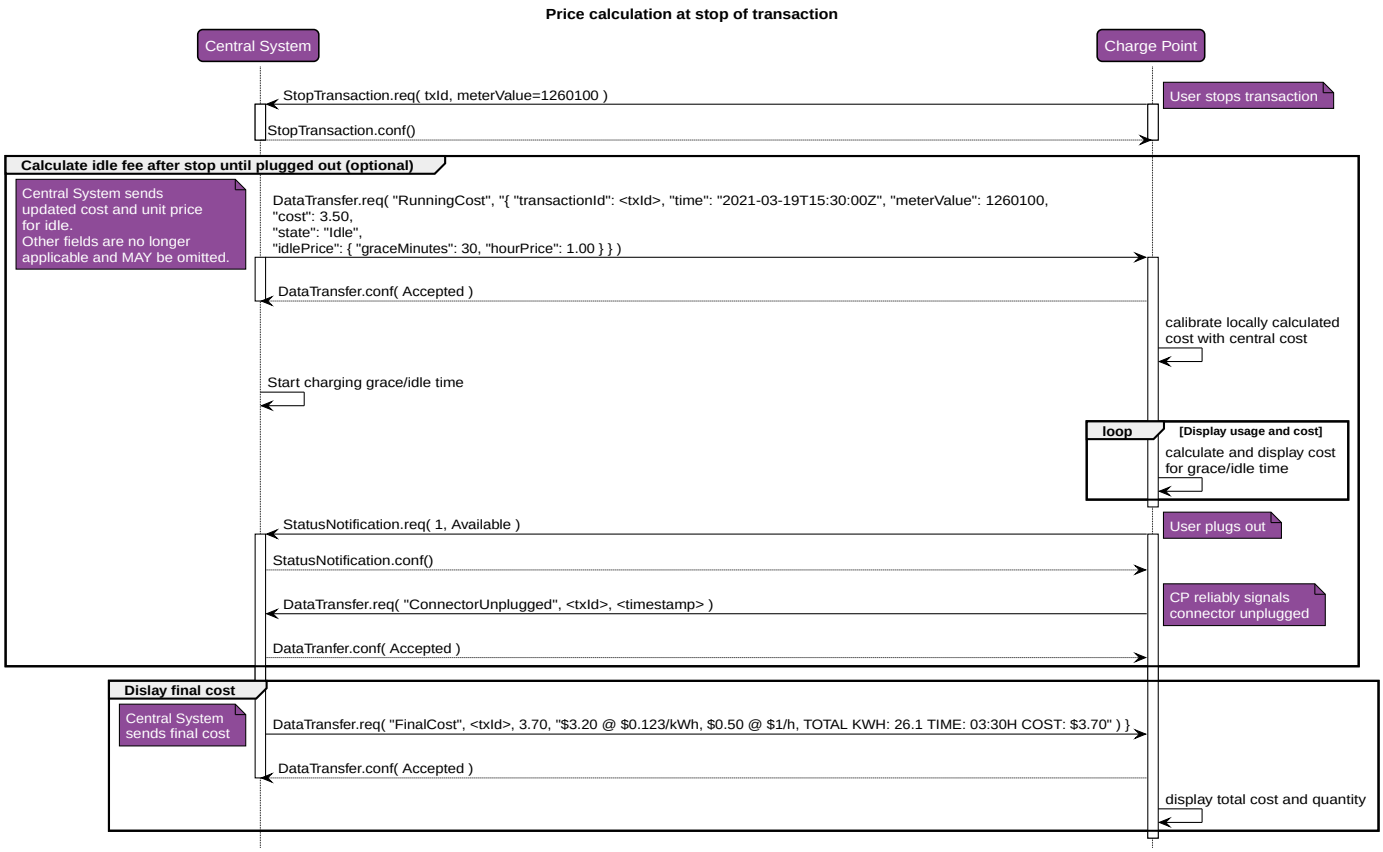

Figure 3. Price calculation at stop of transaction

3.7. Sequence diagrams for (partly) offline situations

Different operators may implement different behavior for offline situations. This may also be governed by local legislation. The sequence diagrams shown in this chapter are meant to illustrate which messages can be sent in partly offline situations. These are not normative.

The final cost is calculated by Central System. Without a connection the charge point can only show the cost that it has calculated locally. This should be the same as the cost from Central System, but that is not guaranteed. Several approaches are possible:

No charge

When the final cost cannot be displayed, nothing is charged for the transaction.

Partial charge

When the final cost cannot be displayed, only charge up to the last meter value received during the transaction.

Regular charge

When the final cost cannot be displayed, show a message to the customer, that the cost shown on the display may deviate slightly from the final cost on the invoice, because the system currently has no connection with the back-office. Or alternatively, do not show any cost at all due to lack of connectivity.

3.7.1. Price calculation when temporarily offline

The charge point can continue with its calculations for current and next period, and final cost can be displayed, because it is back online before the transaction finishes.

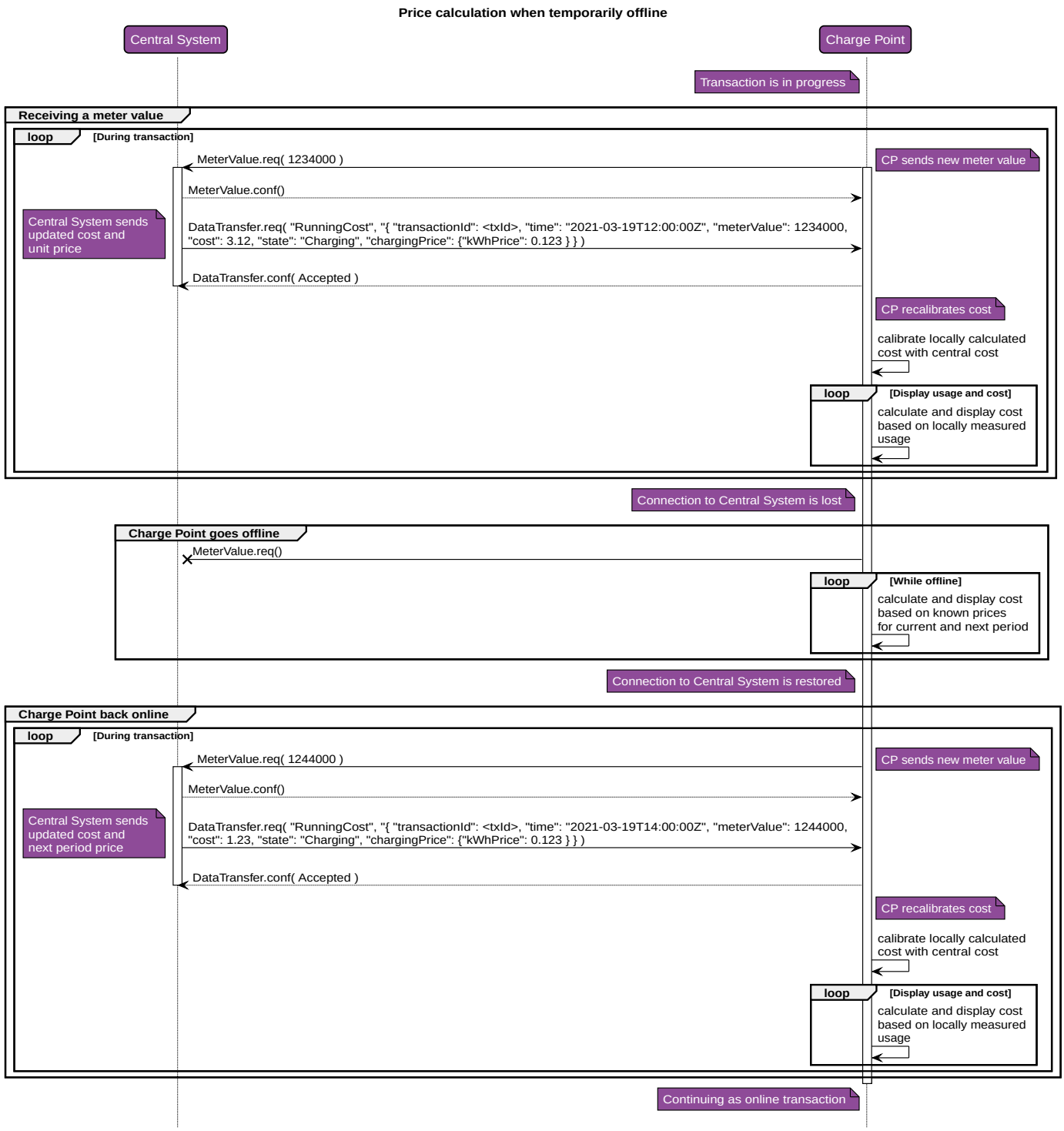

Figure 4. Price calculation when temporarily offline

3.7.2. Price calculation for offline stopped transaction

The charge point has calculated the price, but final cost cannot be displayed until the charge point is online again.

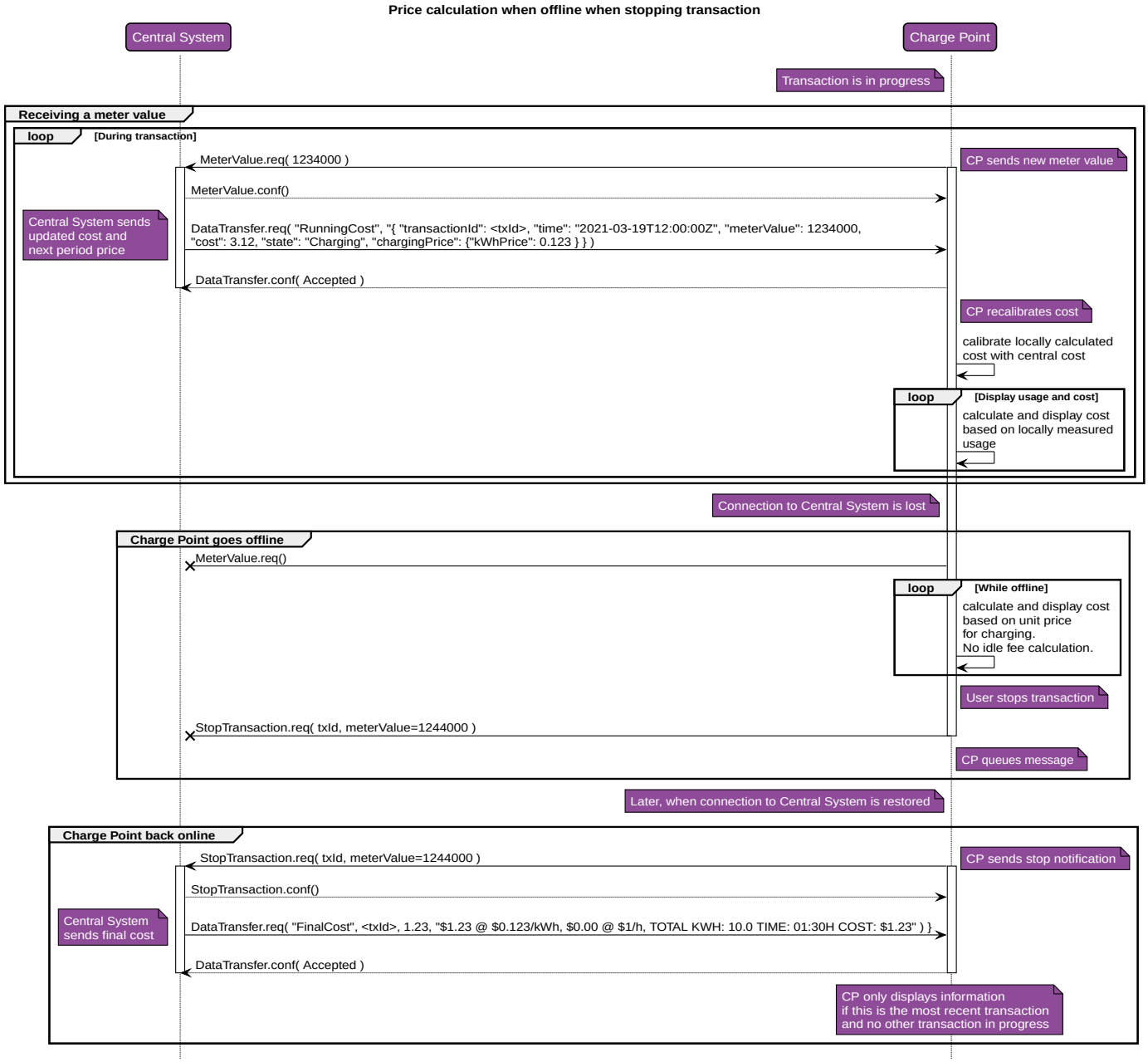

Figure 5. Price calculation for offline stopped transaction

3.7.3. Price calculation for offline started transaction

User-specific price is not known. Charge point cannot show running costs, unless it has been configured with unit prices for the default pricing. The user agrees to use default pricing. If the charge point gets back online, Central System continues to use the default price scheme.

NOTE Alternatively, charging could be disallowed or be free of charge while offline.

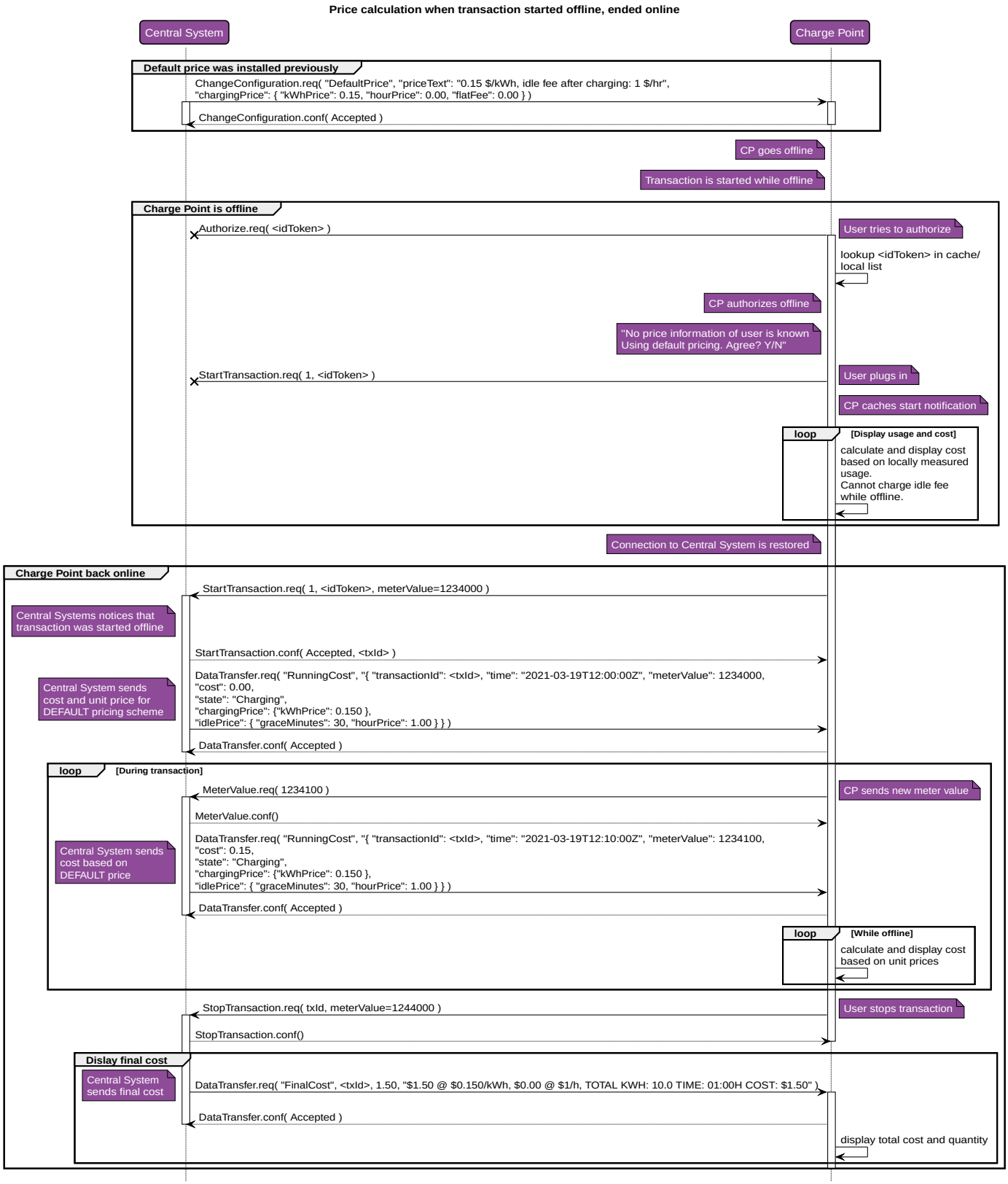

Figure 6. Price calculation for offline started transaction

3.7.4. Price calculation for completely offline transaction

User-specific price is not known. Charge point cannot show running costs, unless it has been configured with unit prices for the default pricing. User agrees to use default pricing. When Central System later receives the offline start/stop transaction messages, it calculates cost using the default price scheme.

NOTE Alternatively, charging could be disallowed or be free of charge while offline.

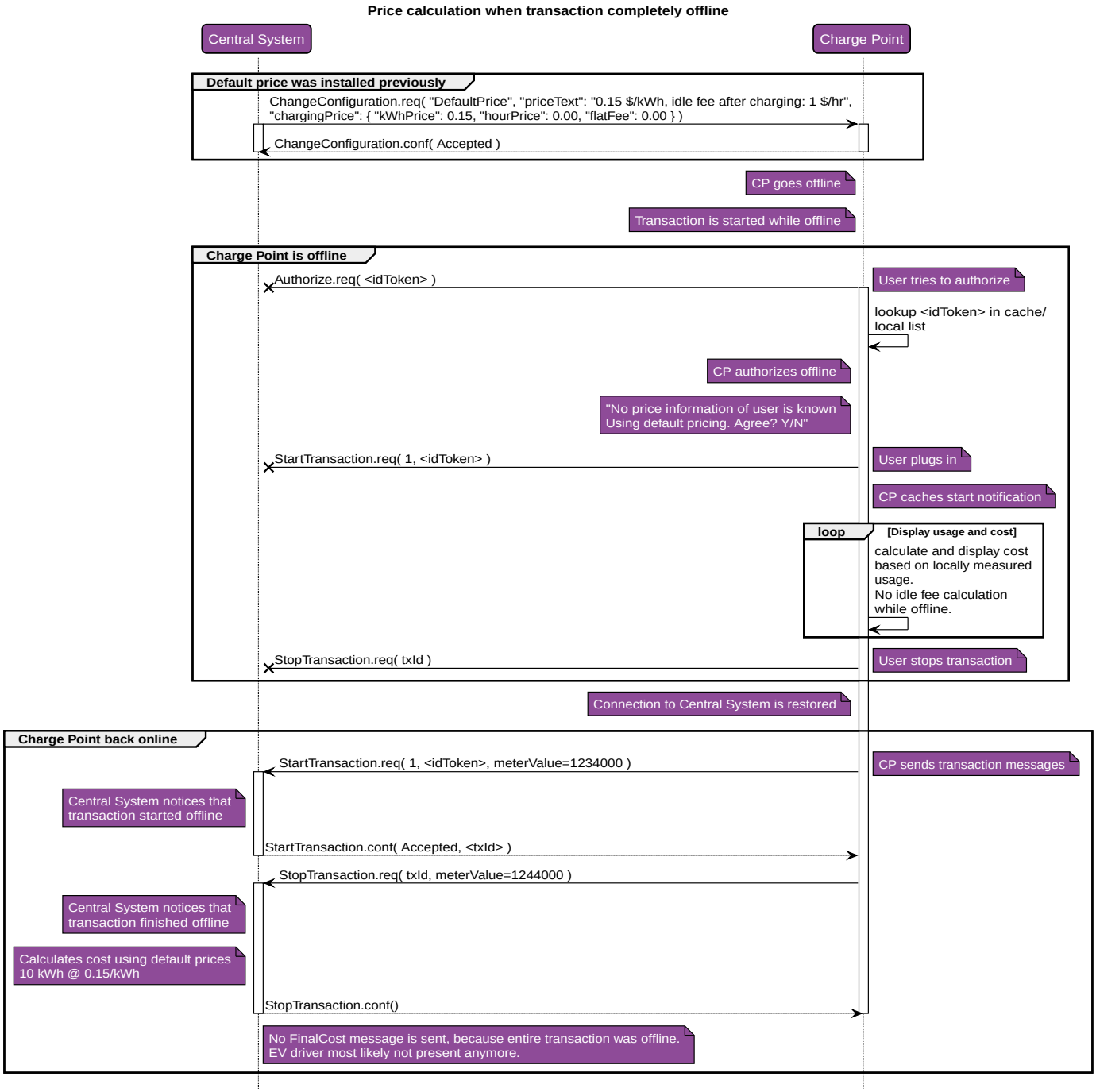

Figure 7. Price calculation for completely offline transaction

4. Tariff and Cost Features in OCPP 2.0.1

4.1. Displaying Price and Cost in OCPP 2.0.1

OCPP 2.0.1 offers the following standard messages to communicate pricing information to the charging station:

SetDisplayMessageRequest

OCPP provides a mechanism to display messages on a charging station. This can be used to display generic (i.e. not customer-specific) pricing information. Alternatively, or when it is offline, the charging station can display the price message that is configured in the TariffFallbackMessage of component `TariffCostCtrlr`.

AuthorizeResponse

Customer-specific pricing information can be returned upon successful authorization. CSMS requests this information at the eMSP of the associated charging contract that has just been authorized. It then formats this a 512 character message for display at the charging station.

CostUpdatedRequest

CSMS sends periodic updates of the running total cost of a session with the CostUpdated message, which is a single decimal value.
We extend this message with *customData* fields that provide the unit price information that is required to do local running cost calculations. Details in CustomData Extension of CostUpdatedRequest.

TransactionEventResponse (eventType Update)

When the tariff changes during the transaction, for example, in case of a 'time of use' based price, then CSMS sends the updated price in the *updatedPersonalMessage* of TransactionEventResponse.

TransactionEventResponse (eventType Ended)

CSMS shows the final total cost of a transaction in *totalCost* in the TransactionEvent message that marks the end of the transaction. The field *updatedPersonalMessage* contains the text for the display that shows the final price and its price components. It might also contain a URL that points to a location where the user can retrieve an invoice. For convenience this URL can optionally be provided in a customData *qrCodeText* field. A charging station that supports it, can then display the URL as a QR code for easy scanning by the user.
A TransactionEventResponse with such a customData extension looks as follows:

```
 TransactionEventResponse( "totalCost": 3.31,
    "updatedPersonalMessage": {"format": "UTF8", "language": "en",
    "content": "$2.81 @ $0.12/kWh, $0.50 @ $1/h,TOTAL KWH: 23.4 TIME: 03.50
COST: $3.31.
        Visit www.cpo.com/invoices/13546 for an invoice of your session."},
    "customData": {
        "vendorId": "org.openchargealliance.org.qrcode",
        "qrCodeText": "https://www.cpo.com/invoices/13546"
    }
 )
```

NOTE	We recommend using a boolean device model variable `QRCodeDisplayCapable` on the `DisplayMessageCtrlr` component to tell CSMS whether the station can display QR codes or not.

4.1.1. CustomData Extension of CostUpdatedRequest

NOTE	A charging station that supports this customization will report the device model variable `CustomizationCtrlr.CustomImplementationEnabled` with instance "org.openchargealliance.costmsg" as true.

The CostUpdatedRequest is extended with *customData* fields. These are the same fields as are used in the DataTransfer customization for OCPP 1.6, with the exception of the fields for cost and transaction id, which are already part of the message, and without the trigger **atCPStatus**, because a charging station with OCPP 2.0.1 already sends a TransactionEventRequest with a meter value when the charging state changes.

The CSMS will send a CostUpdatedRequest as soon as the driver is authorized and the EV is connected to an EVSE. At that point the tariff for the user and EVSE in use are known and a CostUpdatedRequest with **chargingPrice** and other fields can be sent.

If all optional fields are used, it looks as follows:

CostUpdatedRequest (full example)

```
CostUpdatedRequest( "totalCost": 1.00, "transactionId": "12345",
   "customData": {
       "vendorId": "org.openchargealliance.costmsg",
       "timestamp": "2021-03-19T12:00:00Z", "meterValue": 1234000,
       "state": "Charging",
       "chargingPrice": {
           "kWhPrice": 0.123, "hourPrice": 0.00, "flatFee": 0.00 },
       "idlePrice": { "graceMinutes": 30, "hourPrice": 1.00 },
       "nextPeriod: {
           "atTime": "2021-03-19T19:00:00Z",
           "chargingPrice": {
               "kWhPrice": 0.100, "hourPrice": 0.00, "flatFee": 0.00 },
           "idlePrice": { "graceMinutes": 30, "hourPrice": 1.00 }
       }
       "triggerMeterValue": {
           "atTime": "2021-03-19T23:00:00Z",
           "atEnergykWh": 50.0,
           "atPowerkW": 0.1
       }
    }
)
```

The JSON in **customData** has the following structure:

Table 10. customData fields

Field	Type	Card.	Description
vendorId	string	1..1	Fixed value of "org.openchargealliance.costmsg" to identify this customization.
timestamp	dateTime	1..1	Timestamp of the meter value upon which this cost is based.
meterValue	integer	1..1	Meter value (Wh) upon which this cost is based.
state	string	1..1	"Charging" or "Idle". Determines which pricing components are to be used.
chargingPrice	ChargingPrice	1..1	Price components while charging.
idlePrice	IdlePrice	0..1	Price components while not charging. Optional if no idle fee is charged.
nextPeriod	NextPeriod	0..1	Pricing for next period.
triggerMeterValue	Triggers	0..1	Triggers to request a new TransactionEvent with meter value.

Table 11. ChargingPrice

Field	Type	Card.	Description
kWhPrice	decimal	0..1	Price per kWh.
hourPrice	decimal	0..1	Price per hour of charging.
flatFee	decimal	0..1	Flat fee for (part of) charging session.

Table 12. IdlePrice

Field	Type	Card.	Description
graceMinutes	integer	0..1	Grace period in minutes before idle time is charged. Grace minutes start counting from the *timestamp* of the message in which *state* changed from "Charging" to "Idle".
hourPrice	decimal	0..1	Price per hour while idle.

Table 13. NextPeriod

Field	Type	Card.	Description
atTime	dateTime	1..1	Time when these prices become active.
chargingPrice	ChargingPrice	1..1	Price components while charging.
idlePrice	IdlePrice	0..1	Price components while idle. Optional if no idle fee charged.

Table 14. Triggers

Field	Type	Card.	Description
atTime	dateTime	0..1	Time when a meter value must be sent.
atEnergykWh	decimal	0..1	Consumed energy amount in kWh upon which a meter value must be sent.

Field	Type	Card.	Description
atPowerkW	decimal	0..1	Power threshold in kW when meter value must be sent when crossing in downward or upward direction. Can either be used to trigger a meter value when vehicle stops charging or when vehicle charges at a high power that requires a different tariff. It is recommended to implement a hysteresis around this value to avoid repetitive triggers when the power fluctuates around this level.

4.2. Device Model Settings for Tariff and Cost

The following table shows the device model variables on the component `TariffCostCtrlr` that need to be set in order to activate the showing of tariff and cost on the charging station.

Table 15. Standard variables for TariffCostCtrlr in OCPP 2.0.1

Variable	Instance	Description
Enabled	"Tariff"	When true this enables showing of tariffs.
Enabled	"Cost"	When true this enables showing of cost.
TariffFallbackMessage		Message (and/or tariff information) to be shown to an EV Driver when there is no driver specific tariff information.
TotalCostFallbackMessage		Message to be shown to an EV Driver when the Charging Station cannot retrieve the cost for a transaction at the end of the transaction.
Currency		Currency used for tariff and cost information.

Some additional information is needed to let the charging station know which prices to use when a transaction is started while the charging station is offline and cannot receive RunningCost messages. (See Sequence diagrams for (partly) offline situations).

An advantage of the device model is, that new variables can easily be introduced. Rather than using a JSON structure to communicate charging and idle prices for use when the charging station is offline, like we do with DefaultPrice in OCPP 1.6, we can use variables of the `TariffCostCtrlr` component for this. The offline energy and time prices are presented as instances of the OfflineChargingPrice variable. These are set by the CSMS and are usually not changed often.

There is no "OfflineIdlePrice" variable, because when the charge station is offline, the CSMS cannot notify the charge station that it should start using the idle price. See Idle Time During Transaction for more information on idle time.

Table 16. New variables for TariffCostCtrlr for offline support

Variable	Instance	Description
TariffFallbackMessage	"Offline"	Optional alternative message to be shown to an EV Driver when the charging station is offline.

Variable	Instance	Description
OfflineChargingPrice	"kWhPrice"	The energy (kWh) price for transactions started while offline.
OfflineChargingPrice	"hourPrice"	The time (hour) price for transactions started while offline.

4.3. Multi-language support for tariffs (new in v3)

NOTE

A charging station that provides multi-language support for tariffs will not only report the device model variable `CustomizationCtrlr.CustomImplementationEnabled` with instance "org.openchargealliance.costmsg" as true, but also the same variable with instance "org.openchargealliance.multilanguage" as true.

4.3.1. Device Model Variables

In order to support multiple languages for TariffFallbackMessage and TotalCostfallbackMessage an instance per message is created. Since the instance is used to designate the language, we can no longer use the instance "Offline" to set the message to display when the charging station is offline. A new variable is introduced to hold the offline fallback message.

Table 17. Changed TariffCostCtrlr variables for multi-language support

Variables	Instance	Type	Description
TariffFallbackMessage	"<language code>"	string	Message (and/or tariff information) to be shown to an EV Driver when there is no driver specific tariff information available.
OfflineTariffFallbackMessage	"<language code>"	string	Message (and/or tariff information) to be shown to an EV Driver when Charging Station is offline.
TotalCostFallbackMessage	"<language code>"	string	Message to be shown to an EV Driver when the Charging Station cannot retrieve the cost for a transaction at the end of the transaction.

E.g.:

Variables	Instance	Description
TariffFallbackMessage	"en-US"	"$0.12/kWh, no idle fee"
TariffFallbackMessage	"es-MX"	"$0.12/kWh, sin tarifa de inactividad"
TariffFallbackMessage	"fr-CA"	"$0,12/kWh, pas de frais d'inactivité"

Language codes shall be specified as RFC5646 (per OCPP 2.0.1). This allows eg `en`, `es`, and `fr` to be sufficient if no regional variant is needed.

As instances aren't dynamic in the device model, the station will need to expose all languages that it supports.

4.3.2. Multi-language driver/user specific tariffs

Driver specific tariffs / pricing information can be returned in the `AuthorizeResponse` message. The standard message includes a field *idTokenInfo* of type IdTokenInfoType. That type contains the preferred language of this customer and a personal message.

Table 18. IdTokenInfoType

Field Name	Field Type	Card.	Description
status	AuthorizationStatusEnumType	1..1	Authorization status of idToken
...			<other fields>
language1	string[0-8]	0..1	Optional. Preferred user interface language of identified user. Contains a language code as defined in RFC5646.
language2	string[0-8]	0..1	Optional. Second preferred user interface language...
personalMessage	MessageContentType	0..1	Optional. Personal message that can be shown to the EV Driver and can be used for tariff information, user greetings etc.

personalMessage is the field that is suitable for sending tariff/pricing information to be displayed to the EV Driver. Note that there is also a field, *updatedPersonalMessage*, in *TransactionEventResponse* of the same type that can be used to send updated tariffs/pricing information.

Table 19. MessageContentType

Field Name	Field Type	Card.	Description
format	MessageFormatEnumType	1..1	Required. Format of the message.
language	string[0-8]	0..1	Optional. Message language identifier. Contains a language code as defined in RFC5646.
content	string[0-512]	1..1	Required. Message contents.

In order to support multiple languages, the following *customData* extension is added to the idTokenInfoType in the AuthorizeResponse:

Table 20. IdTokenInfoType customData extension

	Field Name	Field Type	Card.	Description
customData				
	vendorId	string	1..1	Id of our extension. "org.openchargealliance.multilanguage"
	personalMessageExtra	MessageContentType	0..4	Personal messages in the extra languages.

4.3.3. Multi-language final cost

The TransactionEventResponse allows the CSMS to send the final cost and information about that. In the same way as described above, the TransactionEventResponse will be extended with a customData extension, as follows:

Table 21. TransactionEventResponse customData extension

	Field Name	Field Type	Card.	Description
customData				
	vendorId	string	1..1	Id of our extension. "org.openchargealliance.multilanguage"
	updatedPersonalMessageExtra	MessageContentType	0..4	Personal messages in the extra languages.

4.3.4. Default UI language

The DisplayMessageCtrlr component is extended with a "Language" variable to report the languages supported by the charging station UI.

DisplayMessageCtrlr.Language holds the default language. This can be changed by CSMS. The *variableCharacteristics.valuesList* of DisplayMessageCtrlr.Language holds the list of supported languages. This cannot be changed by CSMS.

A maximum of 4 additional messages in other languages can be provided in a *personalMessage* of an AuthorizationResponse or an *updatedPersonalMessage* of a TransactionEventResponse.

The standard field *idTokenInfo.personalMessage* holds the message text in the default language. All other languages are part of the custom field *idTokenInfo.customdata.personalMessageExtra[]*.

Table 22. DisplayMessageCtrlr.Language configuration variable

Required	yes		
Component	**componentName**	DisplayMessageCtrlr	
Variable	**variableName**	Language	
	variableAttributes	**mutability**	ReadWrite
	variableCharacteristics	**dataType**	OptionList
Description	Default language code, per RFC 5646, of this Charging Station. Value must be one of *valuesList*. The set of language codes in *valuesList* is the full set of languages that is supported by the Charging Station.		

4.4. Idle Fees

NOTE	Idle fee calculation is not a DMS requirement.

If the pricing model needs to calculate an idle fee, i.e. charge a fee for being connected after charging has completed, then there are two different scenarios that can occur. There is idle time within the transaction and idle time after the transaction has ended while the vehicle remains connected to the charging station.

4.4.1. Idle Time During Transaction

DMS or CTEP do not define how idle time should be measured if an idle fee is charged. It is up to the CSMS and not the charging station, to decide when a transaction is considered to be in idle time. The reason for this is, that charging may have been suspended on request of the CSMS due to a demand response event or a smart charging algorithm. In that case it would not be fair to charge an idle fee to the customer.

CSMS can detect that an EV is no longer charging when it sees that meter values stay the same. Depending on the meter value interval, it can take many minutes before it is noticed, unless the charging station sends a TransactionEventRequest with a **chargingState** of *SuspendedEV* or *SuspendedEVSE*.

Detect idle by low power usage

CSMS can request the charging station to send a meter value whenever the vehicle charges with less than a certain power by setting the **atPowerkW** trigger (e.g. at a value of 0.1 kW). If the charging station has been configured to send not only the measurand "Energy.Active.Import.Register", but also "Power.Active.Import", then the CSMS will immediately know from the power reading, that the EV has stopped charging. If only the register reading is provided, then the CSMS will have to wait until the next meter value to know that the EV has stopped charging. This is still quicker than via the regular meter intervals, though.

NOTE	A meter value is triggered to be sent everytime the **atPowerkW** threshold is crossed in upward or downward direction. Some logic in the CSMS is required to determine the direction in which the threshold is crossed.

Detect idle by chargingState change

When a charging station goes into a *SuspendedEV* or *SuspendedEVSE* state, then CSMS knows that charging has paused when it receives a TransactionEventRequest with that **chargingState**. This message also contains the current meter value.

Switch between Idle and Charging

When CSMS detects that charging has paused, it will send a new RunningCost message with **state** set to "Idle". From that moment on, the charging station will charge the idle time according to the parameters in **idlePrice**.

When the EV resumes charging, the charging station may send a TransactionEventRequest with **chargingState** "Charging" if it was in a suspended state before, or it will after some time send a TransactionEventRequest with a new sampled meter value. (Sending of the TransacationEvent message with meter value can be forced by setting the **atPowerkW** trigger). The CSMS will then send a new CostUpdatedRequest message with **state** set to "Charging" and the charging station will use the **chargingPrice** information to calculate cost to show on the display.

4.4.2. Idle Time When Transaction Has Ended

A major difference between OCPP 2.0.1 and 1.6 is, that OCPP 2.0.1 allows the CSO to configure the start and stop moments of a transaction. When a CSO wishes to charge an idle fee for the period that an EV is not charging, but remains connected to the charging station, then it suffices to set the configuration variable `TxCtrlr::TxStopPoint` to "EVConnected". This will cause the transaction to remain active, until the EV disconnects. For a charging station that has parking bay occupancy detection, this can even be set to *ParkingBayOccupancy*, such that a fee can be charged for as long as the parking spot is occupied.

4.5. Direct Payment

Normally, it is the eMSP that invoices the customer, but in the event of direct payment with credit or debit card or other means, then the CPO acts as an eMSP. In that case there must be a way to make a printable receipt available to the customer upon request. (See [EVSE_S.2.6])

Since a charging station is not normally equipped with a printer, the solution is to display a message that directs the customer to a website where the receipt can be downloaded.

The field **updatedPersonalMessage** in the TransactionEventResponse at end of the transaction contains a text field to display a cost-breakdown and information on how to retrieve an invoice. Optionally, a customData field **qrCodeText** can be added, that contains a URL for the invoice as a QR code that can be scanned by the user.

4.6. Sequence diagram showing OCPP 2.0.1 messages

The following diagram shows the sequence of events to report price and cost during a transaction, including the optional part to record an idle fee. This assumes that the `TxStartPoint` and `TxStopPoint` have both been set to "EVConnected".

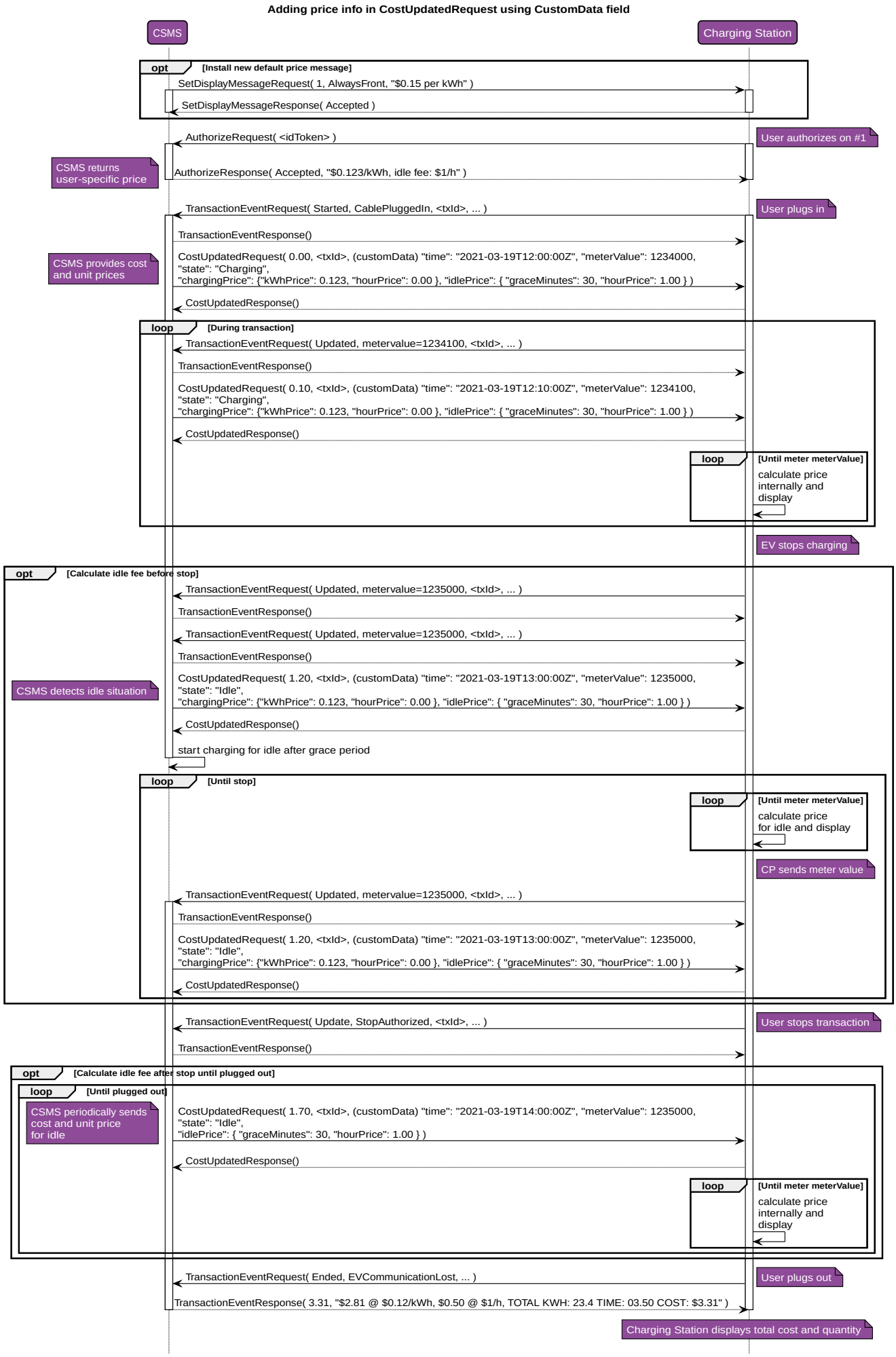

Figure 8. Using CustomData in CostUpdatedRequest

4.7. Offline behavior

Behavior in offline situations is the same as shown in Sequence diagrams for (partly) offline situations for OCPP 1.6.

5. Checklist Items Requiring Attention

Analysis of the checklist learns that almost all relevant information can be provided using the above-mentioned messages of OCPP.

The following checklist items require special attention, since they refer to data that cannot be transferred between charging station and CSMS using OCPP. The data can be communicated using other means, however.

Checklist item 1.7: Selection of Unit Price

There is no message in OCPP to communicate a choice of unit price by the user, if the charging station supports a choice between multiple unit prices. However, it is not required to be able to show multiple prices. In fact this is not needed for the most common situation, namely to distinguish credit card payment and eMSP charge card payment. The credit card tariff (for which the CPO acts as eMSP) can be shown as the default tariff on the charging station. Only when a customer authorizes using an eMSP charge card will CSMS retrieve the tariff that applies for that eMSP and communicate it on the charging station.

If there is a need to select a price from multiple unit prices, for example, to distinguish between regular charging and smart charging, then the following solutions are possible:

- Provide a mobile phone app to display and select the price to use. This app communicates the selected price with its own protocol to the CSMS.
- Create a customization using OCPP device model or DataTransfer to communicate the user's choice.

Checklist item 1.8: In case TOU unit price modifications are allowed

In case TOU unit price modifications are allowed, the following customer options shall be offered before a charging session could be started:

1. Customer accepts TOU based rate increases
2. Customer elects to terminate charge session when a TOU rate increase occurs.
3. Customer does not accept any TOU option.

There is no message in OCPP to communicate whether the customer accepts TOU based rate increases. This is, of course, only an issue when TOU based rate increases are present on the charging station. If that is the case, then one of the above-mentioned solutions, like a smartphone app or an OCPP customization, is needed.

Checklist item 1.31: The system must provide a receipt

For transactions conducted with point-of-sale systems or devices activated by credit cards, debit cards, electronic payment (ApplePay) or other electronic payment method recorded representation containing information about the transaction shall be available to the customer as outlined in the following items.

Printable receipt must always be available to the customer upon request. The system must provide a receipt to be made available to the customer at the completion of the transaction through either:

- a built-in recording element OR
- a separate recording element that is part of the system OR
- an electronic device (phone, computer, etc.) accessible by the system.

Since a charging station is not normally equipped with a printer, the solution is to display a message that directs the customer to a website where the receipt can be downloaded.

6. Appendix: Relevant Requirements per Section

This section lists the checklist requirements that are relevant in relation to OCPP. A comment in *italic* font describes how OCPP can help to fulfill this checklist requirement.

Section 1. Primary Indicating and Recording Elements

S.1.2 EVSE indicating Elements

1.1. An electronic digital indicating element shall

- 1.1.2. Displays for a minimum of 15 seconds after activation by the user
- 1.1.3. All indications and representations of energy sold are clearly identified
- 1.1.4. All indications and representations of time-based charges are clearly identified

`OCPP 1.6` *Can be achieved by setting a custom configuration key that holds the price description. During the transaction unit prices are provided in the RunningCost message. See Displaying Price and Cost in OCPP 1.6*
`OCPP 2.0.1` *Can be achieved with DisplayMessageRequest or AuthorizeResponse.PersonalMessage. During the transaction unit prices are provided in the CostUpdated message. See Displaying Price and Cost in OCPP 2.0.1*

S.1.3.2 Value of Smallest Unit

The value of the smallest unit of indicated delivery by an EVSE, and recorded delivery if the EVSE is equipped to record, shall be no greater than 0.0005 MJ or 0.0001 kWh.

`OCPP 1.6` *The meter values in StartTransaction and StopTransaction are integer values in Wh. In order to provide these meter values in 0.1 Wh accuracy, the start and stop meter values must be provided in the transactionData field of StopTransaction.req.*
`OCPP 2.0.1` *Meter values in TransactionEventRequests are decimals and can be provided in 0.1 Wh accuracy.*

S.2.4.3 Selection of Unit Price

1.7. An EVSE may be equipped with means for selecting more than one unit price, provided that the selected unit price cannot be changed after the initial flow begins.

`OCPP 1.6` & `OCPP 2.0.1` *OCPP can provide a message for the selection of multiple prices, however, it does not provide a means for the user to select one price or the other. However, it is not required to be able to show multiple prices. In fact this is not needed for the most common situation to distinguish credit card payment and eMSP charge card payment. The credit card tariff (for which the CPO acts as eMSP) can be shown as the default tariff on the charging station. Only when a customer authorizes using an eMSP charge card will CSMS retrieve the tariff that applies for that eMSP and communicate it on the charging station.*

If there is a need to select a price from multiple unit prices, for example, to distinguish between regular charging and smart charging, then the following solutions are possible:

- *Provide a smartphone app to display and select the price to use. This app communicates the selected price with its own protocol to the CSMS.*

- *Create a customization using OCPP device model or DataTransfer to communicated the user's choice.*

1.8. The selected unit price must be made clearly evident on the EVSE.Once selected the unit price cannot be changed at the point of sales prior to or during the delivery except when the change is triggered by a notified Time Of Use (TOU) modification.

If TOU base rated increases are present on the charging station, then the customer must be allowed to select one of the following options:

1.8.1. Customer accepts TOU based rate increases

1.8.2. Customer elects to terminate charge session when a TOU rate increase occurs.

1.8.3. Customer does not accept any TOU option.

`OCPP 1.6` & `OCPP 2.0.1` *It is not required to support TOU based rate changes, but if they are implemented, then the solutions mentioned in the previous paragraph (1.7) can be used to facilitate this.*

1.10. When a delivery is completed, the total price and quantity for that transaction shall be displayed on the face of the EVSE for at least 5 minutes or until the next transaction is initiated by using controls on the device or other user-activated (e.g., customer-activated) controls.

`OCPP 1.6` *The total price when delivery is complete is the last update of total price, that the charging station received via the custom message that was implemented for that purpose. Quantity in kWh and/or time is known by the charging station. Both can be displayed by charging station. See Displaying Price and Cost in OCPP 1.6*
`OCPP 2.0.1` *The total price has been sent by CSMS as part of the last TransactionEventRequest message. Quantity in kWh and/or time is known by the charging station. Both can be displayed by charging station. See Displaying Price and Cost in OCPP 2.0.1*

S.2.3. Provisions for Power Loss

1.15. The quantity and total sales price shall be recallable for 15 minutes after the power failure.

`OCPP 1.6` & `OCPP 2.0.1` *Since price calculation is done at the CSMS, this information is not lost during a power failure at the charging station.*

1.17. The quantity and total sales price values shall be correct if the delivery is continued after a power failure.

`OCPP 1.6` & `OCPP 2.0.1` *Since price calculation is done at the CSMS, this information is not lost during a power failure at the charging station.*

S.2.4.1. Display of Unit Price

1.25. Means shall be provided to display the unit price on the face of the device.

`OCPP 1.6` *Can be achieved by setting a custom configuration key that holds the price description. See Displaying Price and Cost in OCPP 1.6*
`OCPP 2.0.1` *This can be displayed via the DisplayMessageRequest or*

AuthorizeResponse.PersonalMessage. The display can either display prices for all units as one message or let the use step through the various units. CSMS will send all information in one message, though. See Displaying Price and Cost in OCPP 2.0.1

1.26. The unit price shall be expressed in dollars and decimals of dollars using a dollar sign.

`OCPP 1.6` *Can be achieved by setting a custom configuration key that holds the currency.*
`OCPP 2.0.1` *The currency is configured via the device model variable* `TariffCostCtrlr::Currency`.

S. 2.4.3. Selection of Unit Price

1.29. Prior to delivery using controls on the device. OR

1.30. Through deliberate action of the purchaser using

1. controls on the device;
2. personal or vehicle mounted electronic equipment communicating with the system; or
3. verbal instructions

`OCPP 1.6` & `OCPP 2.0.1` *The same remarks as for checklist item 1.7 apply here.*

S.2.6. Recorded Representations

For transactions conducted with point-of-sale systems or devices activated by credit cards, debit cards, electronic payment (ApplePay) or other electronic payment method recorded representation containing information about the transaction shall be available to the customer as outlined in the following items. Printable receipt must always be available to the customer upon request.

1.31. The system must provide a receipt to be made available to the customer at the completion of the transaction through either

1.31.1. a built-in recording element OR a separate recording element that is part of the system OR an electronic device (phone, computer, etc.) accessible by the system.

`OCPP 1.6` & `OCPP 2.0.1` *Charging station can display a message to point the customer to a website where the receipt can be downloaded.*

Section 2. Computing

No requirements for OCPP.

Section 3. Measuring Elements

No requirements for OCPP.

Section 4. Test of the EVSE System (Hb44 N and T Sections)

No requirements for OCPP.

Annex # EVSEs with parking time charging functionality

1.1 The EVSE shall indicate and record, the time in minutes for time intervals of 60 minutes or less and in hours and minutes for time intervals greater than 60 minutes.

`OCPP 1.6` & `OCPP 2.0.1` *EVSE can display parking time continuously, since start of transaction. CSMS also*

knows when transaction starts and ends and can add this to the receipt.

[1] NCWM Publication No.14, National Type Evaluation Program: Technical Policy Checklists and Test Procedures, draft 201902

[2] California Type Evaluation Program

[3] In line with the terminology of OCPP 1.6 we will use Central System and charge point, instead of CSMS and charging station in this section.