E. Transactions

1. Introduction

This Functional Block describes the OCPP Transaction related functionalities. Transactions are started/stopped on the Charging Station. Note that at most one transaction can be active on an EVSE at any point in time.

1.1. Flexible transaction start/stop

To support as many business cases as possible, and to prevent sending too many messages when not needed for certain business cases, OCPP 2.0.1 supports flexible configuration of the start and stop of a transaction.

For this the following Configuration Variables are defined:

- `TxStartPoint`
- `TxStopPoint`

These 2 Configuration Variables make it possible to define when a transaction should start: TransactionEventRequest (eventType = Started) and when a transaction should stop: TransactionEventRequest (eventType = Ended)

1.1.1. Readonly or Read/Write

OCPP 2.0.1 supports 2 options for the transaction start/stop Configuration Variables. They can either be: RW (read-write) or R (read-only).

When a Charging Station supports RW, the CSO can configure the settings. To support all possible settings, the software in the Charging Station has to be more flexible.

With only R, the settings are fixed in firmware, the CSO can read the settings to learn how a Charging Station will behave, but cannot configure it. This makes for a simpler implementation. When the needs of the target market are well known there might be no need to implement the flexible model.

1.1.2. OCPP 1.6 Transaction compatibility

If transactions similar to OCPP 1.6 are wanted, this section describes how the transaction start and stop point should be configured.

In OCPP 1.x the moment a Charging Station should send StartTransaction.req was not defined very precise, generally this was done when the power path was closed: relay closed. Which should only be done after authorization.

To support similar transaction start behaviour, the value: *PowerPathClosed* is to be used. (and for completeness, also add: *EnergyTransfer*

Table 95. The settings for an OCPP 1.6 compatible transaction

Configuration Variable	Values
`TxStartPoint`	PowerPathClosed,EnergyTransfer
`TxStopPoint`	EVConnected,Authorized,DataSigned,PowerPathClosed

For stop behavior the *ParkingBayOccupancy* should not be added, OCPP 1.6 did not support this, and in case of a dual socket charging station where somebody is using the 'opposite' connector, the transaction would then be stopped, while the EV could still be charging.

1.2. TransactionId generation

New in OCPP 2.0.1: Transaction IDs are now generated by the Charging Station.

In OCPP 1.x this was done by the CSMS. This had some drawbacks. When a Charging Station was offline it had a transaction which did not have a transactionId.

The TransactionId generated by a Charging Station has to be unique for this Charging Station. During the lifetime of a Charging Station it should never use the same TransactionId twice. Also when the Charging Station is rebooted, power cycled, firmware updated, repaired etc.

OCPP does not specify an algorithm to use, but it is RECOMMENDED to use UUIDs.

1.3. Delivering transaction-related messages

The primary purpose of TransactionEventRequest messages is to give the CSMS the information that it will later use to bill the transaction. To be sure that the CSMS receives all the necessary information for billing a transaction, OCPP uses two mechanisms: *retrying* and *sequence numbers*.

1.3.1. Retrying

The Charging Station sends TransactionEventRequest messages to the CSMS System as soon as possible after the events they report on have occurred.

If the Charging Station is offline, or if an error occurs processing the message in transport, the CSMS will be missing billing information. In order to repair the missing information in the CSMS, the Charging Station should retry to deliver this information. When the Charging Station fails to receive a TransactionEventResponse for a TransactionEventRequest message within the message timeout period, the Charging Station should follow the retry procedure described in use case E13 - Transaction-related message not accepted by CSMS.

1.3.2. Sequence numbers

When delivery of TransactionEventRequest messages fails and will be retried later, the result is that TransactionEventRequest messages may arrive in the CSMS in a different order from the one in which the transaction events occurred at the Charging Station. This in turn would make it difficult for the CSMS to know if it received all TransactionEventRequest messages about a transaction, which the CSMS may want to know before it starts billing the transaction.

In order to make it possible to know that all TransactionEventRequest messages about a transaction were received, OCPP uses *sequence numbers* in TransactionEventRequest messages. For every EVSE, the Charging Station maintains a counter of the number of TransactionEventRequest messages generated about that EVSE. When generating a new TransactionEventRequest message, the Charging Station includes the current value of the EVSE's counter in the **seqNo** field of the request, and then increments the counter. With this mechanism, a CSMS can check if it has full information about a transaction by checking that:

- It received a TransactionEventRequest about the start of the transaction, with a **seqNo** *a*
- It received a TransactionEventRequest about the stop of the transaction, with a **seqNo** *o* greater than *a*.
- It received a TransactionEventRequest about the transaction with **seqNo** *n* for every integer *n* between *a* and *o*

1.3.2.1. Sequence number generation

This section is normative.

When a TransactionEventRequest has to be created, the Charging Station SHALL set the message's **seqNo** field to the value of a transaction event request counter maintained for the EVSE on which the transaction is occurring. Immediately after taking the counter value, the Charging Station SHALL update the counter value as follows:

- If the counter's value is smaller than 2147483647, the counter's value is incremented.
- If the counter's value is 2147483647, the counter's value is set to 0.

The counter SHALL be stored persistently across cold boots.

The initial value of an EVSE's transaction event request counter SHALL be 0 (it shall not be reset at the start of a transaction).

An EVSE's transaction event request counter SHOULD NOT be updated by other processes than transaction event message creation.

1.4. Authorization

To simplify the use cases in this functional block, the way an EV Driver is authorized is not part of these use cases. It will simply be called something like: "User authorization successful" or "The EV Driver is authorized by the Charging Station and/or CSMS.". This may be any way of authorizing an EV Driver. See functional block: C Authorization for all the options and requirements for authorization.

2. Use cases & Requirements

2.1. OCPP transaction mechanism

E01 - Start Transaction options

Table 96. E01 - Start Transaction

No.	Type	Description
1	**Name**	Start Transaction options
2	**ID**	E01
	Functional block	E. Transactions
3	**Objective(s)**	To inform the CSMS that a transaction at the Charging Station has started.
4	**Description**	This use case describes the different moments a Charging Station can start a transaction (send TransactionEventRequest with eventType = Started), depending on the configuration of the Charging Station.
5	*Actors*	Charging Station, CSMS, EV Driver
S1	*Scenario objective*	**To start a transaction when a parking bay occupancy detector detects an "EV".**
	Scenario description	**1.** The EV Driver parks his "EV" at a Charging Station with a parking bay occupancy detector, which triggers the detector. **2.** The Charging Station sends a TransactionEventRequest (eventType = Started) notifying the CSMS about a transaction that has started (even when the driver is not yet known. **3.** The CSMS responds with a TransactionEventResponse, confirming that the TransactionEventRequest was received.
	Prerequisite(s)	No transaction is ongoing on the EVSE. Configuration Variable: `TxStartPoint` contains: ParkingBayOccupancy
	Postcondition(s)	**Successful postcondition:** The transaction is ongoing and the CSMS is *Successfully* informed. **Failure postcondition:** The transaction is *not* ongoing, *or* The CSMS is *not* informed.

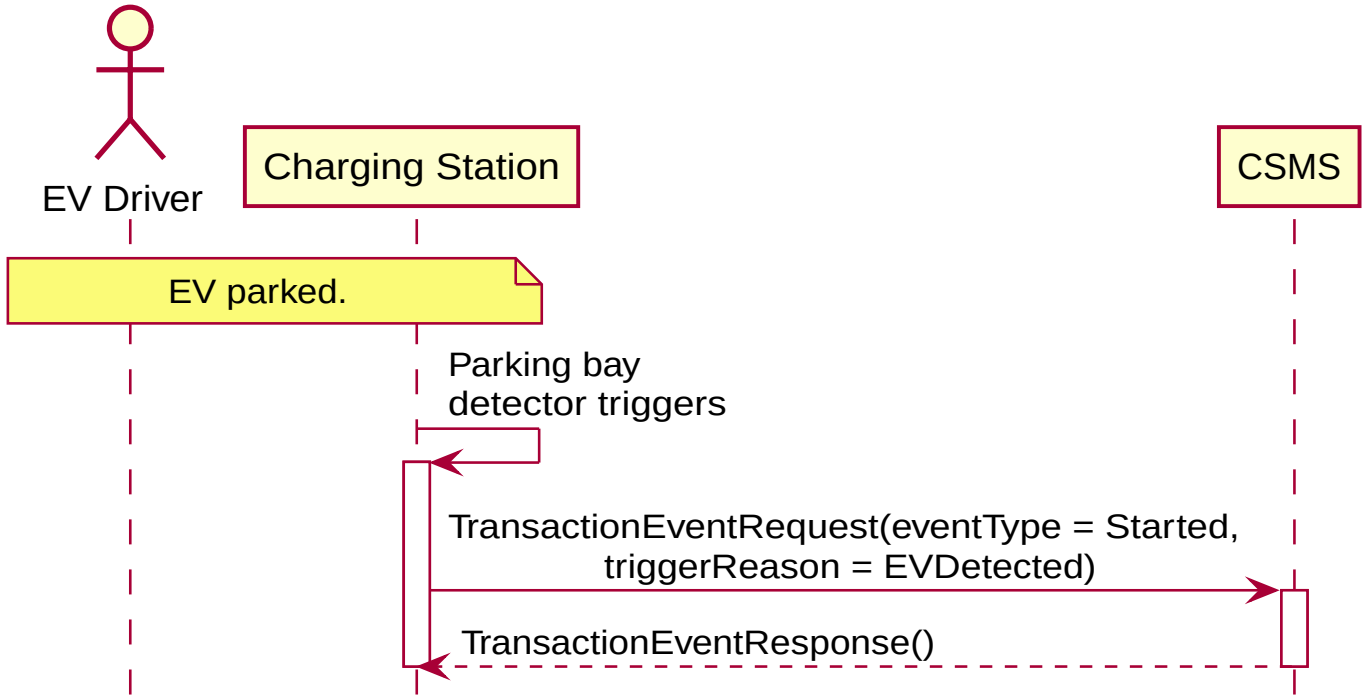

Figure 40. Sequence Diagram: Start Transaction options - ParkingBayOccupancy

S2	*Scenario objective*	**To start a transaction when communication is set up between the Charging Station and an EV (for example: cable plugged in correctly on both sides)**
	Scenario description	**1.** The Charging Station sets up a connection with the EV. **2.** The Charging Station sends a TransactionEventRequest (eventType = Started) notifying the CSMS about a transaction that has started (even when the driver is not yet known). **3.** The CSMS responds with a TransactionEventResponse, confirming that the TransactionEventRequest was received.
	Prerequisite(s)	No transaction is ongoing on the EVSE. Configuration Variable: `TxStartPoint` contains: EVConnected (Not: ParkingBayOccupancy)
	Postcondition(s)	**Successful postcondition:** The transaction is ongoing and the CSMS is *Successfully* informed. **Failure postcondition:** The transaction is *not* ongoing, *or* The CSMS is *not* informed.

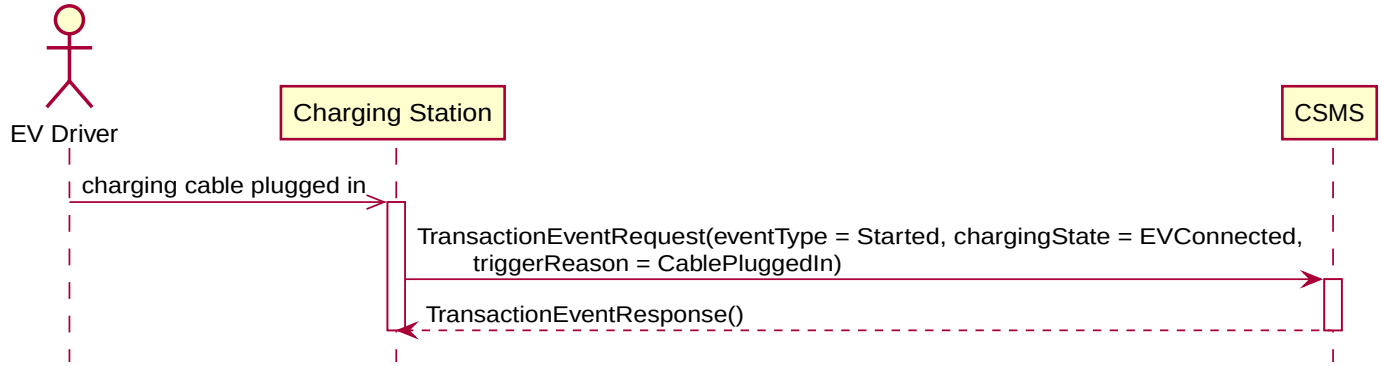

Figure 41. Sequence Diagram: Start Transaction options - EVConnected

S3	*Scenario objective*	**To start a transaction when the EV Driver is authorised to charge.**
	Scenario description	**1.** The EV Driver provides his identification **2.** The Charging Station validates the provided identification (for example via the Authorization Cache or an AuthorizeRequest). **3.** The Charging Station sends a TransactionEventRequest (eventType = Started) notifying the CSMS about a transaction that has started. **4.** The CSMS responds with a TransactionEventResponse, confirming that the TransactionEventRequest was received.
	Prerequisite(s)	No transaction is ongoing on the EVSE. Configuration Variable: `TxStartPoint` contains: Authorized (Not: ParkingBayOccupancy).
	Postcondition(s)	**Successful postcondition:** The transaction is ongoing and the CSMS is *Successfully* informed. **Failure postcondition:** The transaction is *not* ongoing, *or* The CSMS is *not* informed.

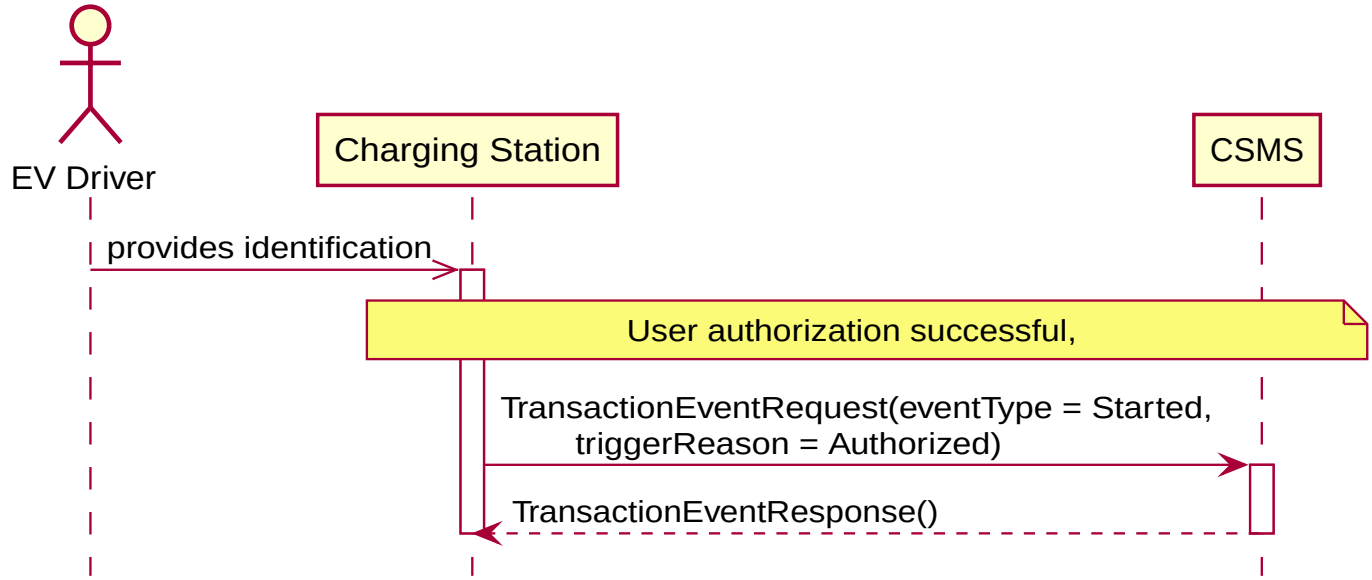

Figure 42. Sequence Diagram: Start Transaction options - Authorized

S4	*Scenario objective*	**To start a transaction when the meter has provided the first signed meter values before starting with charging.**
	Scenario description	**1.** The EV Driver plugs in the cable at the Charging Station and the EV. **2.** The Charging Station request the Meter for a signed value. **3.** The Meter provides a signed value (this might take some time). **4.** The Charging Station sends a TransactionEventRequest (eventType = Started) notifying the CSMS about a transaction that has started. **5.** The CSMS responds with a TransactionEventResponse, confirming that the TransactionEventRequest was received.
	Prerequisite(s)	No transaction is ongoing on the EVSE. Configuration Variable: `TxStartPoint` contains: DataSigned (Not: ParkingBayOccupancy, EVConnected or Authorized). The Charging Station has a meter that can sign measured values Configuration Variable: `AlignedDataSignReadings` set to *true*.
	Postcondition(s)	**Successful postcondition:** The transaction is ongoing and the CSMS is *Successfully* informed. **Failure postcondition:** The transaction is *not* ongoing, *or* The CSMS is *not* informed.

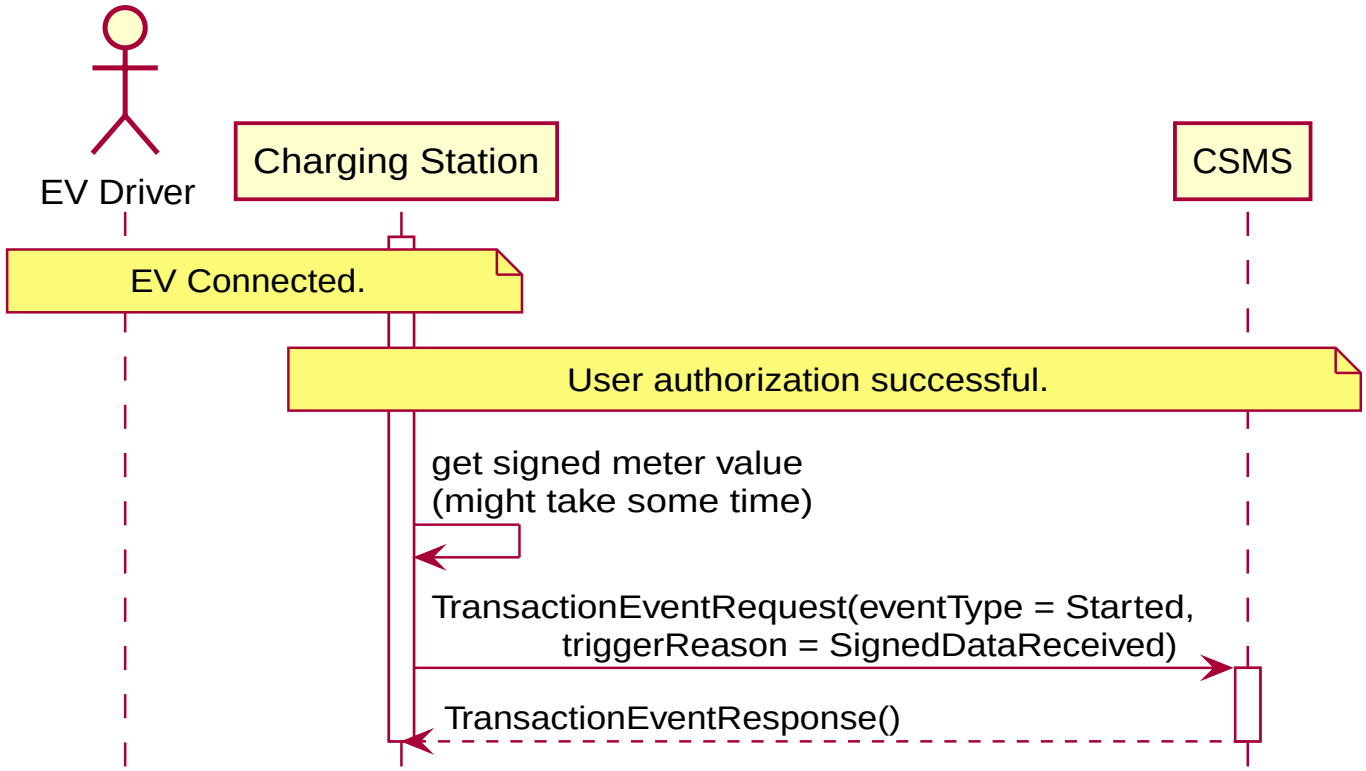

Figure 43. Sequence Diagram: Start Transaction options - DataSigned

S5	*Scenario objective*	**To start a transaction when all preconditions are available to start charging, but energy does not yet have to be transfered (for example: power relay closed).**
	Scenario description	**1.** The EV Driver is authorized by the Charging Station and/or CSMS. **2.** The Charging Station closes the power relay. **3.** The Charging Station sends a TransactionEventRequest (eventType = Started) notifying the CSMS about a transaction that has started. **4.** The CSMS responds with a TransactionEventResponse, confirming that the TransactionEventRequest was received.
	Prerequisite(s)	No transaction is ongoing on the EVSE. Configuration Variable: `TxStartPoint` contains: PowerPathClosed (Not: ParkingBayOccupancy, EVConnected, Authorized or DataSigned). Charging Cable plugged in.
	Postcondition(s)	**Successful postcondition:** The transaction is ongoing and the CSMS is *Successfully* informed. **Failure postcondition:** The transaction is *not* ongoing, *or* The CSMS is *not* informed.

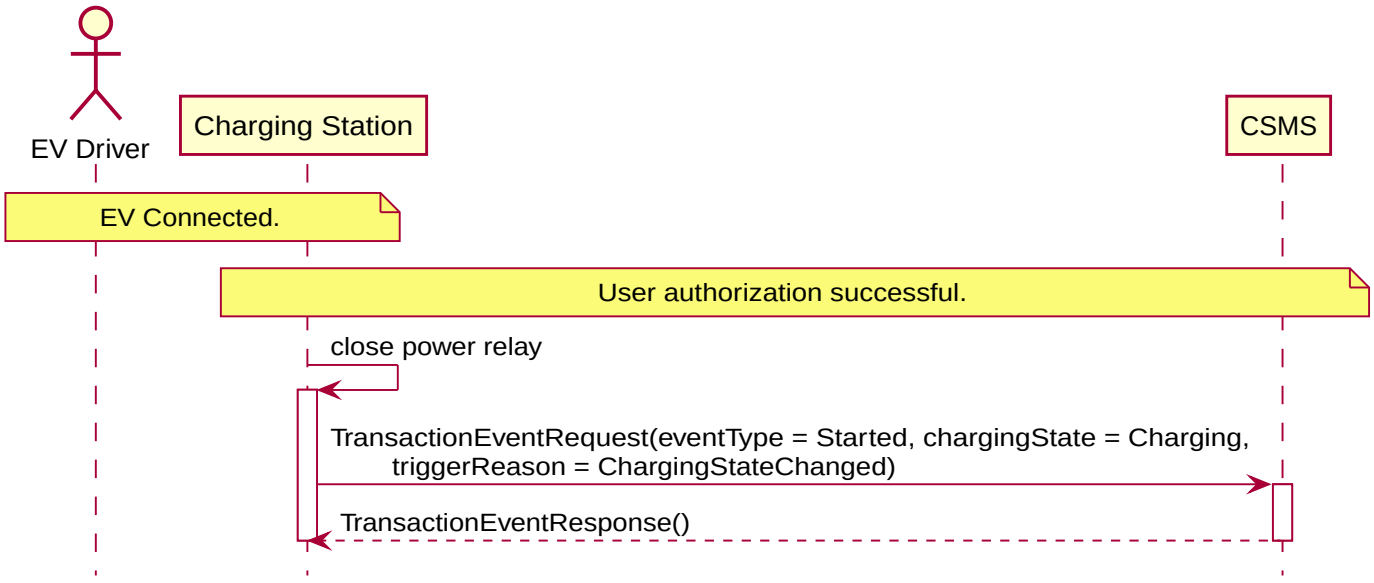

Figure 44. Sequence Diagram: Start Transaction options - PowerPathClosed

S6	*Scenario objective*	**To start a transaction when the energy flow starts.**
	Scenario description	**1.** The EV Driver is authorized by the Charging Station and/or CSMS. **2.** The Charging Station closes the power relay. **3.** The EV starts charging, energy flow starts. **4.** The Charging Station sends a TransactionEventRequest (eventType = Started) notifying the CSMS about a transaction that has started. **5.** The CSMS responds with a TransactionEventResponse, confirming that the TransactionEventRequest was received.
	Prerequisite(s)	Configuration Variable: `TxStartPoint` contains: EnergyTransfer (Not: ParkingBayOccupancy, EVConnected, Authorized, DataSigned or PowerPathClosed).
	Postcondition(s)	**Successful postcondition:** The transaction is ongoing and the CSMS is *Successfully* informed. **Failure postcondition:** The transaction is *not* ongoing, *or* The CSMS is *not* informed.

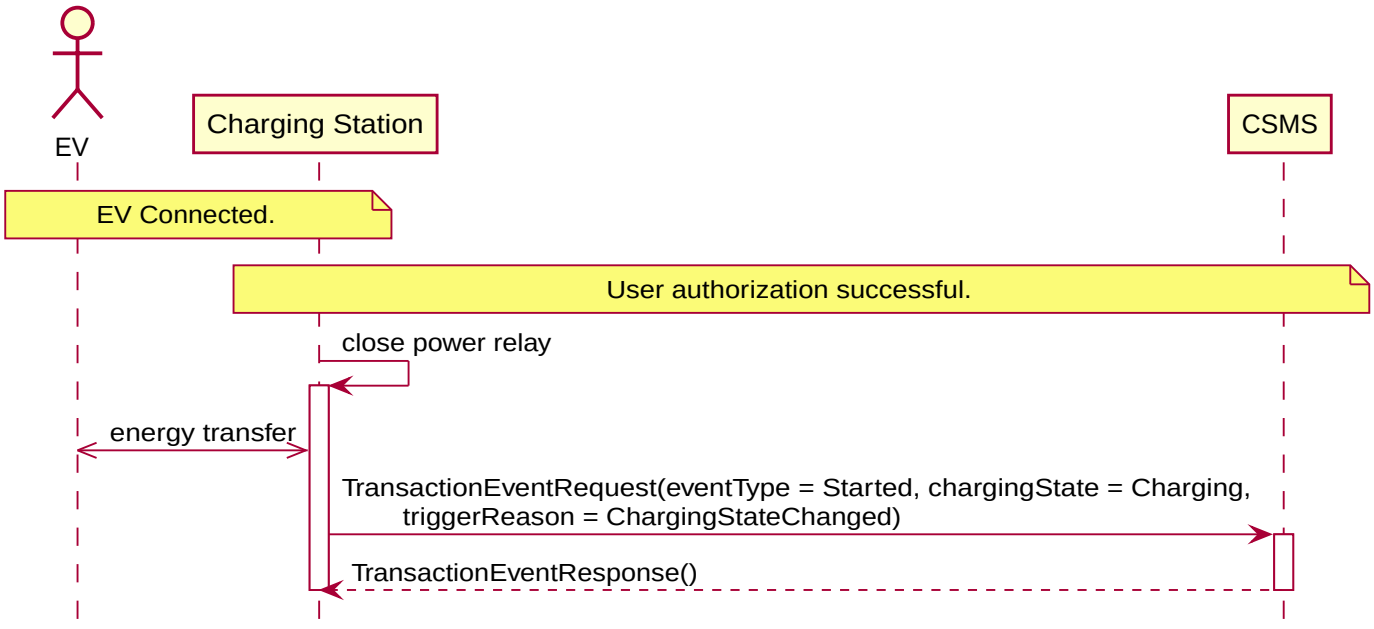

Figure 45. Sequence Diagram: Start Transaction options - EnergyTransfer

7	**Error handling**	n/a
8	**Remark(s)**	n/a

E01 - Start Transaction options - Requirements

Table 97. E01 - Requirements

ID	Precondition	Requirement definition
E01.FR.01	`TxStartPoint` contains: ParkingBayOccupancy AND Parking Bay Detector detects an "EV" AND No transaction has started yet	The Charging Station SHALL start a transaction and send a TransactionEventRequest (eventType = Started) to the CSMS.
E01.FR.02	`TxStartPoint` contains: EVConnected AND The Charging Station has a connection with the EV AND No transaction has started yet on this EVSE	The Charging Station SHALL start a transaction and send a TransactionEventRequest (eventType = Started) to the CSMS.
E01.FR.03	`TxStartPoint` contains: Authorized AND The EV Driver is authorized AND No transaction has started yet	The Charging Station SHALL start a transaction and send a TransactionEventRequest (eventType = Started) to the CSMS.
E01.FR.04	`TxStartPoint` contains: DataSigned AND The Charging Station has a meter that can sign measured values AND Configuration Variable: `AlignedDataSignReadings` set to *true*. AND The Charging Station has retrieved a signed meter value AND No transaction has started yet	The Charging Station SHALL start a transaction and send a TransactionEventRequest (eventType = Started) to the CSMS.
E01.FR.05	`TxStartPoint` contains: PowerPathClosed AND The Charging Station closes the power relay AND No transaction has started yet on this EVSE	The Charging Station SHALL start a transaction and send a TransactionEventRequest (eventType = Started) to the CSMS.
E01.FR.06	`TxStartPoint` contains: EnergyTransfer AND Energy flow starts AND No transaction has started yet on this EVSE	The Charging Station SHALL start a transaction and send a TransactionEventRequest (eventType = Started) to the CSMS.
E01.FR.07	When a TransactionEventRequest has to be created	The Charging Station SHALL set the message's **seqNo** field as specified in Sequence Number Generation.
E01.FR.08		The transactionId generated by the Charging Station MUST be unique for each transaction started by that Charging Station, even when the Charging Station is rebooted, repaired, firmware is updated etc, it SHALL ensure that it never generates the same TransactionId twice.

ID	Precondition	Requirement definition
E01.FR.09	When configured to send meter data in the TransactionEventRequest (eventType = Started), See: Meter Values - Configuration AND EVSE is known at start of transaction	The Charging Station SHALL add the configured measurands to the optional meterValue field with *context* = `Transaction.Begin` in the TransactionEventRequest(eventType = Started) sent to the CSMS to provide more details during the transaction.
E01.FR.10	After the EV Driver is authorized for this transaction	The Charging Station SHALL send a TransactionEventRequest that contains IdTokenType information.
E01.FR.11	E01.FR.10	The CSMS SHALL verify the validity of the identifier in TransactionEventRequest.
E01.FR.12	E01.FR.11	The CSMS SHALL send a TransactionEventResponse that includes an authorization status value.
E01.FR.13	This transaction ends a reservation	The next TransactionEventRequest SHALL contain the reservationId.
E01.FR.14	After TransactionEventRequest(eventType = Started) has been sent for a specific EVSE and Connector	The Charging Station SHALL NOT start another transaction on a different Connector of the same EVSE until this transaction has ended.
E01.FR.15	When sending a TransactionEventRequest	The Charging Station SHALL set the triggerReason to inform the CSMS about what triggered the event. What reason to use is described in the description of TriggerReasonEnumType.
E01.FR.16	After the EV is connected with the Charging Station.	The next TransactionEventRequest SHALL contain *evse.id* AND *evse.connectorId*.
E01.FR.17	When configured to send meter data in the TransactionEventRequest (eventType = Started), See: Meter Values - Configuration AND EVSE is not known at start of transaction	The Charging Station SHALL add the measurands for *eventType* = `Started` to the optional meterValue field with *context* = `Transaction.Begin` in the TransactionEventRequest(eventType = Updated) that occurs when charging starts.

E02 - Start Transaction - Cable Plugin First

Table 98. E02 - Start Transaction - Cable Plugin First

No.	Type	Description
1	**Name**	Start Transaction - Cable Plugin First
2	**ID**	E02
	Functional block	E. Transactions
3	**Objective(s)**	To inform the CSMS that a transaction at the Charging Station has started.
4	**Description**	The EV Driver begins the interaction with the Charging Station by plugging in the charging cable first. The CSMS is notified about this. Then, when the communication between EV and EVSE is established, the transaction is started and the CSMS is notified of this. The EV starts charging.
	Actors	Charging Station, CSMS, EV Driver
	Scenario description	**1.** The EV Driver plugs in the cable at the Charging Station. **2.** The Charging Station sends a StatusNotificationRequest to the CSMS to inform it about a Connector that became *Occupied*. **3.** The Charging Station sends a TransactionEventRequest (eventType = Started) notifying the CSMS about a transaction that has started (even when the driver is not yet known.) **4.** The CSMS responds with a TransactionEventResponse, confirming that the TransactionEventRequest was received. **5.** The EV Driver is authorized by the Charging Station and/or CSMS. **6.** The energy offer starts. **7.** The Charging Station sends a TransactionEventRequest (eventType = Updated) with the authorized idToken information to the CSMS to inform about the charging status and which idToken belongs to the transaction. **8.** The CSMS responds with a TransactionEventResponse to the Charging Station with the IdTokenInfo.status *Accepted*. **9.** During the charging process, the Charging Stations continues to send TransactionEventRequest (Updated) messages for transaction-related notifications.

No.	Type	Description
	Alternative scenario(s)	E02 - Start Transaction - IdToken First E04 - Offline Start Transaction E05 - Start Transaction - Id not Accepted
5	**Prerequisite(s)**	The Charging Cable is plugged in first.
6	**Postcondition(s)**	**Successful postcondition:** The transaction is ongoing and the CSMS is *Successfully* informed. **Failure postcondition:** The transaction is *not* ongoing. *or* The CSMS is *not* informed. *or* Start Transaction - Id not accepted.

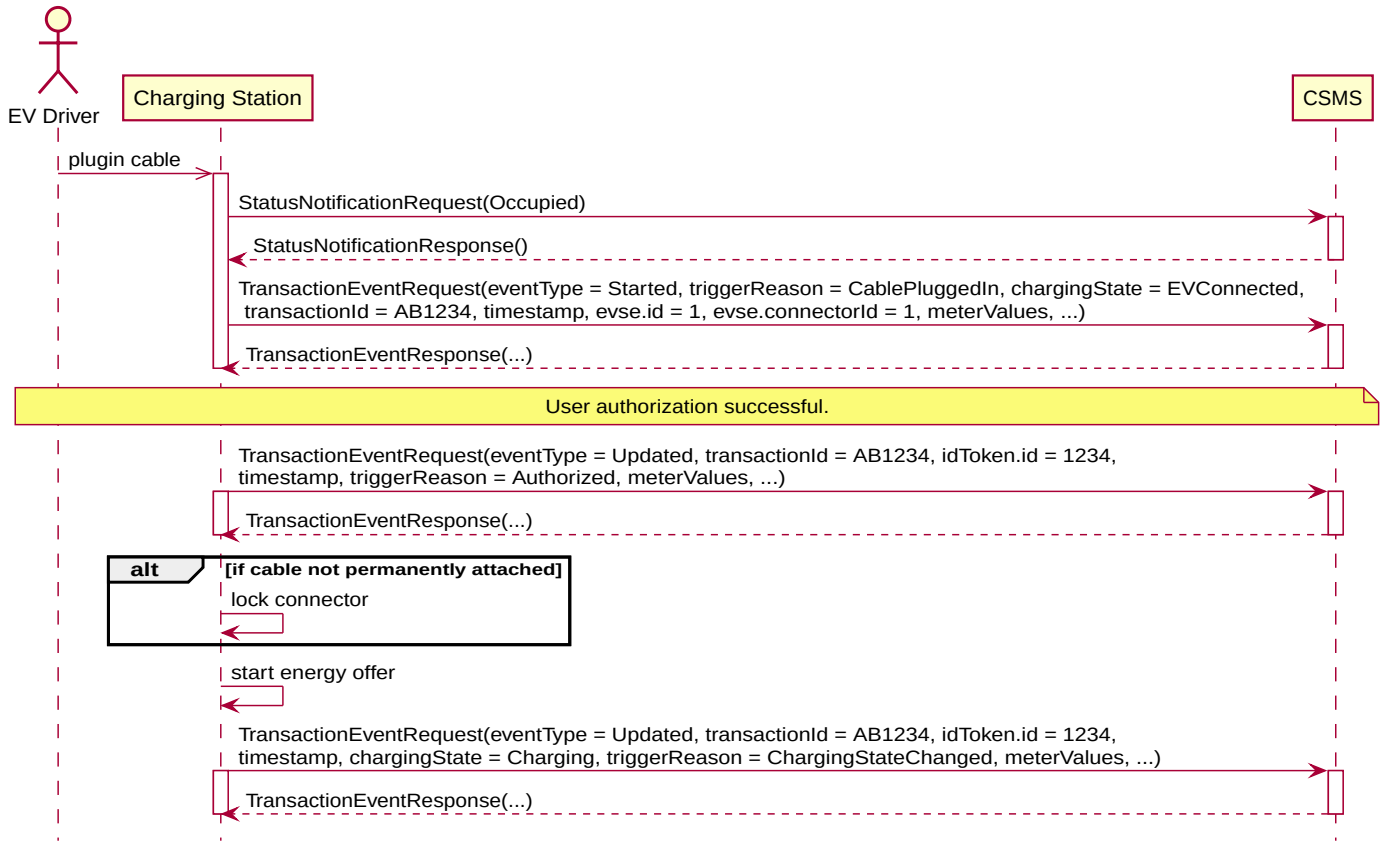

Figure 46. Sequence Diagram: Start Transaction - Cable Plugin First

7	**Error handling**	Failing to respond with TransactionEventResponse will only cause the Charging Station to try the same message again as specified in E12 - Transaction-related message not accepted by CSMS.
8	**Remark(s)**	If the Charging Station has implemented an Authorization Cache, then upon receipt of TransactionEventResponse, the Charging Station updates the cache entry. It is now possible and allowed to send IdTokenType in more than 1 TransactionEventRequest. The CSMS has to be able to handle/process multiple IdTokenType per transaction. It is up to the CSO how they use this information (for billing purposes). The scenario description and sequence diagram above are based on the Configuration Variable for start & stop transaction being configured as follows: `TxStartPoint`: EVConnected, Authorized, DataSigned, PowerPathClosed, EnergyTransfer This use-case is also valid for other configurations, but then the transaction might start at another moment, which might change the sequence in which message are sent. For more details see the use cases: E01 - Start Transaction options and E06 - Stop Transaction options.

E02 - Start Transaction - Cable Plugin First - Requirements

Table 99. E02 - Requirements

ID	Precondition	Requirement definition	Note
E02.FR.01	After the EV Driver is authorized for this transaction.	The next TransactionEventRequest SHALL contain *triggerReason*: Authorized AND IdTokenType information.	
E02.FR.02	E02.FR.01	The CSMS SHALL send a TransactionEventResponse that includes an authorization status value.	
E02.FR.03	This transaction ends a reservation.	The next TransactionEventRequest SHALL contain the reservationId.	See H. Reservation.
E02.FR.04		The CSMS SHALL verify the validity of the identifier in TransactionEventRequest.	Because the identifier might have been authorized locally by the Charging Station using outdated information.

ID	Precondition	Requirement definition	Note
E02.FR.05	When a cable is plugged in	The Charging Station SHALL send a StatusNotificationRequest with status: *Occupied*	
E02.FR.06	When a cable is plugged in	The Charging Station SHALL send a TransactionEventRequest.	
E02.FR.07	When a TransactionEventRequest has to be created	The Charging Station SHALL set the message's **seqNo** field as specified in Sequence Number Generation.	This enables the CSMS to track the completeness of transaction information.
E02.FR.08		The transactionId generated by the Charging Station MUST be unique for each transaction started by that Charging Station, even when the Charging Station is rebooted, repaired, firmware is updated etc, it SHALL ensure that it never generates the same TransactionId twice.	
E02.FR.09	When configured to send meter data in the TransactionEventRequest (eventType = Started), See: Meter Values - Configuration AND EVSE is known at start of transaction	The Charging Station SHALL add the configured measurands to the optional meterValue field with *context* = `Transaction.Begin` in the TransactionEventRequest(eventType = Started) sent to the CSMS to provide more details during the transaction.	
E02.FR.10	When configured to send meter data in the TransactionEventRequest (eventType = Updated), See: Meter Values - Configuration	The Charging Station SHALL add the configured measurands to the optional meterValue field in the TransactionEventRequest(eventType = Updated) sent to the CSMS to provide more details during the transaction.	
E02.FR.11	E02.FR.10 AND Amount of meter data is too much for 1 TransactionEventRequest (eventType = Updated)	The Charging Station MAY split meter data over multiple TransactionEventRequest(eventType = Updated) messages with the same *timestamp*.	
E02.FR.13	If the charging state changes	The Charging Station SHALL send a TransactionEventRequest including the chargingState element.	
E02.FR.14	`AlignedDataSignReadings` is *true*	The Charging Station SHALL retrieve signed meter values and put them in the *signedMeterValue* field of sampledValues.	
E02.FR.15	When sending a TransactionEventRequest	The Charging Station SHALL set the triggerReason to inform the CSMS about what triggered the event. What reason to use is described in the description of TriggerReasonEnumType.	
E02.FR.16	After a transaction has been started	The Charging Station MAY send additional TransactionEventRequest(eventType = Updated) messages during the transaction when a trigger event occurs.	
E02.FR.17	When a transaction-related trigger event occurs, listed in *TriggerReasonEnumType* AND the transaction is ongoing.	The Charging Station SHALL send a TransactionEventRequest with a triggerReason corresponding to the occurred event.	When two trigger reasons overlap, the more specific one should be used. For example, when a cable is plugged in, triggerReason *CablePluggedIn* should be used, not *EVDetected*. When two events occur at the same time, they need transmitted using two separate TransactionEventRequest messages. This is to prevent information loss, when something goes wrong.

ID	Precondition	Requirement definition	Note
E02.FR.18	When the energy transfer starts AND If the Charging Station is able to report the number of phases used	The Charging Station SHALL provide the number of phases used, using the *numberOfPhasesUsed* field.	
E02.FR.19	E02.FR.18 AND during the transaction the number of phases used changes	The Charging Station SHALL provide the adjusted number of phases used, using the *numberOfPhasesUsed* field.	

E03 - Start Transaction - IdToken First

Table 100. E03 - Start Transaction - IdToken First

No.	Type	Description
1	**Name**	Start Transaction - IdToken First
2	**ID**	E03
	Functional block	E. Transactions
3	**Objective(s)**	To enable the EV Driver to start a transaction by first presenting an IdToken at the Charging Station.
4	**Description**	This use case covers how the EV Driver is first authorized by presenting an IdToken before the cable is plugged in and a transaction starts.
	Actors	Charging Station, CSMS, EV Driver
	Scenario description	**1.** The EV Driver is authorized by the Charging Station and/or CSMS. **2.** The Charging Station informs the CSMS that a transaction has started by sending a TransactionEventRequest (eventType = Started). **3.** The EV Driver plugs in the Charging Cable at the Charging Station. **4.** The Charging Station sends StatusNotificationRequest to, and receives StatusNotificationResponse from the CSMS. **5.** The Charging Station informs the CSMS that the EV started charging by sending a TransactionEventRequest (eventType = Updated, chargingState = Charging). **6.** The CSMS responds with TransactionEventResponse, accepting the transaction.
5	**Prerequisite(s)**	IdToken is presented prior to plugin cable.
6	**Postcondition(s)**	**Successful postcondition:** A transaction is started and the ChargingState is *Charging* **Failure postcondition:** No transaction is started

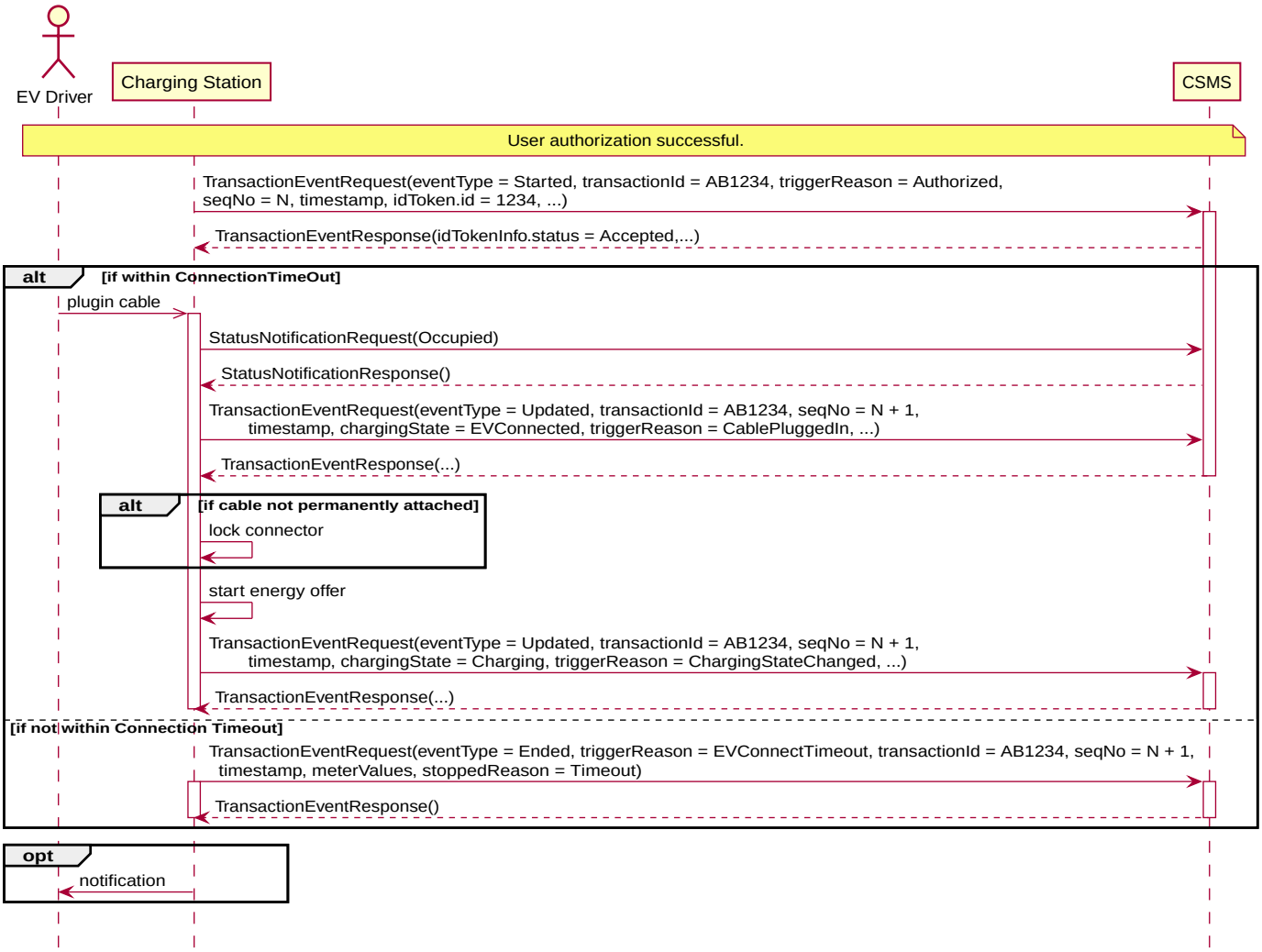

Figure 47. Sequence Diagram: Start Transaction - IdToken First

7	**Error handling**	n/a
8	**Remark(s)**	It is likely that the CSMS applies sanity checks to the data contained in TransactionEventRequest messages it received. The outcome of such sanity checks SHOULD NOT ever cause the CSMS to not respond with a TransactionEventResponse. Failing to do so will only cause the Charging Station to try the same message again as specified in E12 - Transaction-related message not accepted by CSMS. The scenario description and sequence diagram above are based on the Configuration Variable for start transaction being configured as follows: `TxStartPoint`: Authorized, DataSigned, PowerPathClosed, EnergyTransfer This use-case is also valid for other configurations, but then the transaction might start/stop at another moment, which might change the sequence in which message are sent. For more details see the use cases: E01 - Start Transaction options.

E03 - Start Transaction - IdToken First - Requirements

Table 101. E03 - Requirements

ID	Precondition	Requirement definition	Note
E03.FR.01	When the IdToken information is known.	The next TransactionEventRequest SHALL contain IdTokenType information.	
E03.FR.02	E03.FR.01	The CSMS SHALL send a TransactionEventResponse that includes an authorization status.	
E03.FR.03	This transaction ends a reservation for the specific IdToken.	The next TransactionEventRequest SHALL contain the reservationId.	See H. Reservation.

ID	Precondition	Requirement definition	Note
E03.FR.04	When the EV Driver does not plug-in the Charging Cable before the timeout set by the Configuration Variable: `EVConnectionTimeOut`	The Charging Station SHALL send a StatusNotificationRequest with **status** set to **Available**, to the CSMS.	
E03.FR.05	E03.FR.04	The Charging Station SHALL deauthorize the transaction and send a TransactionEventRequest (*triggerReason* = `EVConnectionTimeout`) to the CSMS.	
E03.FR.06	When a TransactionEventRequest has to be created	The Charging Station SHALL set the message's **seqNo** field as specified in Sequence Number Generation.	This enables the CSMS to track the completeness of transaction information
E03.FR.07	When configured to send meter data in the TransactionEventRequest (eventType = Started), See: Meter Values - Configuration AND EVSE is known at start of transaction	The Charging Station SHALL add the configured measurands to the optional meterValue field with *context* = `Transaction.Begin` in the TransactionEventRequest(eventType = Started) sent to the CSMS to provide more details during the transaction.	
E03.FR.08	When configured to send meter data in the TransactionEventRequest (eventType = Updated), See: Meter Values - Configuration	The Charging Station SHALL add the configured measurands to the optional meterValue field in the TransactionEventRequest(eventType = Updated) sent to the CSMS to provide more details during the transaction.	
E03.FR.09	E03.FR.08 AND Amount of meter data is too much for 1 TransactionEventRequest (eventType = Updated)	The Charging Station MAY split meter data over multiple TransactionEventRequest(eventType = Updated) messages with the same *timestamp*.	
E03.FR.10	`AlignedDataSignReadings` is *true*	The Charging Station SHALL retrieve signed meter values and put them in the *signedMeterValue* field of sampledValues.	
E03.FR.11	When configured to send meter data in the TransactionEventRequest (eventType = Started), See: Meter Values - Configuration AND EVSE is not known at start of transaction	The Charging Station SHALL add the measurands for *eventType* = `Started` to the optional meterValue field with *context* = `Transaction.Begin` in the TransactionEventRequest(eventType = Updated) that occurs when charging starts.	
E03.FR.12	When a transaction-related trigger event occurs, listed in *TriggerReasonEnumType* AND the transaction is ongoing.	The Charging Station SHALL send a TransactionEventRequest with a triggerReason corresponding to the occurred event.	When two trigger reasons overlap, the more specific one should be used. For example, when a cable is plugged in, triggerReason *CablePluggedIn* should be used, not *EVDetected*. When two events occur at the same time, they need transmitted using two separate TransactionEventRequest messages. This is to prevent information loss, when something goes wrong.
E03.FR.13	When the energy transfer starts AND If the Charging Station is able to report the number of phases used	The Charging Station SHALL provide the number of phases used, using the *numberOfPhasesUsed* field.	
E03.FR.14	E03.FR.13 AND during the transaction the number of phases used changes	The Charging Station SHALL provide the adjusted number of phases used, using the *numberOfPhasesUsed* field.	

E04 - Transaction started while Charging Station is offline

Table 102. E04 - Transaction started while Charging Station is offline

No.	Type	Description
1	**Name**	Transaction started while Charging Station is offline
2	**ID**	E04
	Functional block	E. Transactions
3	**Objective(s)**	To enable the EV Driver to start a transaction while the Charging Station is *Offline*.
4	**Description**	This use case covers how the Charging Station, while *Offline*, is able to start a transaction using the Local Authorization List or the Authorization Cache.
	Actors	Charging Station, CSMS, EV Driver
	Scenario description	**1.** The transaction starts. **2.** The TransactionEventRequest (eventType = Started) is stored/queued by the Charging Station. **3.** The connection between Charging Station and CSMS is restored. **4.** The Charging Station starts to send queued messages **5.** The stored TransactionEventRequest is sent, notifying the CSMS about the transaction that was started.
	Alternative scenario(s)	E10 - Connection Loss During Transaction
5	**Prerequisite(s)**	The Charging Station is *Offline*. The EV Driver is offline/locally authorized by the Charging Station.
6	**Postcondition(s)**	**Successful postcondition:** The TransactionEventRequest has been responded to by the CSMS AND has been removed from the queue of the Charging Station. **Failure postcondition:** The TransactionEventRequest was NOT responded to by the CSMS AND remains in the queue of the Charging Station.

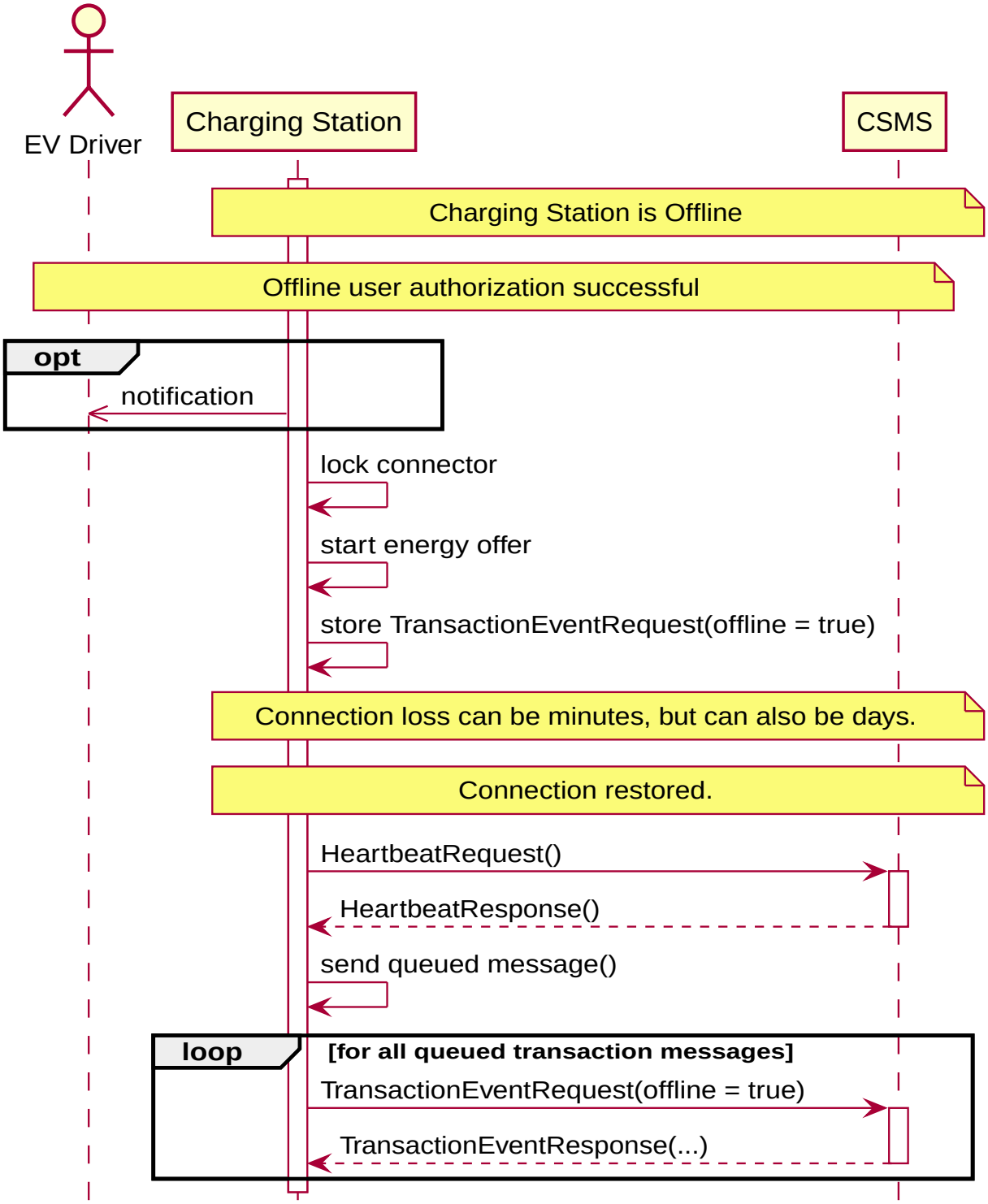

Figure 48. Sequence Diagram: Transaction started while Charging Station is offline

7	**Error handling**	n/a
8	**Remark(s)**	The scenario description and sequence diagram above are based on the Configuration Variable for start transaction being configured as follows: `TxStartPoint`: Authorized, DataSigned, PowerPathClosed, EnergyTransfer This use-case is also valid for other configurations, but then the transaction might start/stop at another moment, which might change the sequence in which message are sent. For more details see the use cases: E01 - Start Transaction options.

E04 - Transaction started while Charging Station is offline - Requirements

Table 103. E04 - Requirements

ID	Precondition	Requirement definition	Note
E04.FR.01	When *Offline*.	The Charging Station MUST queue any TransactionEventRequest messages.	
E04.FR.02	After the connection is restored.	The Charging Station MUST send queued TransactionEventRequest messages.	
E04.FR.03	E04.FR.02	The flag: "offline" SHALL be set to TRUE for any TransactionEventRequest that occurred while the Charging Station was offline.	
E04.FR.04	When a TransactionEventRequest has to be created	The Charging Station SHALL set the message's **seqNo** field as specified in Sequence Number Generation.	This enables the CSMS to track the completeness of transaction information
E04.FR.05	When configured to send meter data in the TransactionEventRequest (eventType = Started), See: Meter Values - Configuration AND EVSE is known at start of transaction	The Charging Station SHALL add the configured measurands to the optional meterValue field with *context* = `Transaction.Begin` in the TransactionEventRequest(eventType = Started) sent to the CSMS to provide more details during the transaction.	
E04.FR.06	When configured to send meter data in the TransactionEventRequest (eventType = Updated), See: Meter Values - Configuration	The Charging Station SHALL add the configured measurands to the optional meterValue field in the TransactionEventRequest(eventType = Updated) sent to the CSMS to provide more details during the transaction.	
E04.FR.07	E04.FR.06 AND *Offline* AND The Charging Station is running low on memory	The Charging Station MAY drop TransactionEventRequest(eventType = Updated) messages.	
E04.FR.08	E04.FR.07	When dropping TransactionEventRequest (eventType = Updated) messages, the Charging Station SHALL drop intermediate messages first (1st message, 3th message, 5th message etc.), not start dropping messages from the start or stop adding messages to the queue.	
E04.FR.09	E04.FR.06 AND Amount of meter data is too much for 1 TransactionEventRequest (eventType = Updated)	The Charging Station MAY split meter data over multiple TransactionEventRequest(eventType = Updated) messages with the same *timestamp*.	
E04.FR.10	`AlignedDataSignReadings` is *true*	The Charging Station SHALL retrieve signed meter values and put them in the *signedMeterValue* field of sampledValues.	

ID	Precondition	Requirement definition	Note
E04.FR.11	When configured to send meter data in the TransactionEventRequest (eventType = Started), See: Meter Values - Configuration AND EVSE is not known at start of transaction	The Charging Station SHALL add the measurands for *eventType* = `Started` to the optional meterValue field with *context* = `Transaction.Begin` in the TransactionEventRequest(eventType = Updated) that occurs when charging starts.	

E05 - Start Transaction - Id not Accepted

Table 104. E05 - Start Transaction - Id not Accepted

No.	Type	Description
1	**Name**	Start Transaction - Id not Accepted
2	**ID**	E05
	Functional block	E. Transactions
3	**Objective(s)**	To enable the Charging Station to suspend a transaction when the IdToken has an AuthorizationStatus that does not allow charging.
4	**Description**	This use case covers how the Charging Station wants to start a transaction while the IdToken is not accepted by the CSMS Because the identifier might have been authorized locally by the Charging Station using outdated information, the CSMS has to validate the IdTokenType in every TransactionEventRequest message it receives that contains an IdTokenType. When receiving a TransactionEventResponse message with **idTokenInfo** field **status** is not **Accepted**, the Charging Station should stop the energy delivery to the EV.
	Actors	Charging Station, CSMS
	Scenario description	**1.** The Charging Station sends TransactionEventRequest (eventType = Started) that contains the IdToken provided by the EV Driver. **2.** The CSMS responds with TransactionEventResponse, with an AuthorizationStatus that does not allow charging. **3.** The Charging Station suspends the energy offer. (Taking into account: `MaxEnergyOnInvalidId`, if supported) **4.** The Charging Station sends TransactionEventRequest (eventType = Updated) with trigger *Deauthorized* and the chargingState *SuspendedEVSE* and receives TransactionEventResponse from the CSMS.
5	**Prerequisite(s)**	The EV Driver is offline/locally authorized by the Charging Station. The IdToken is not allowed to charge by the CSMS.
6	**Postcondition(s)**	**Successful postcondition:** The transaction is kept ongoing, and the cable remains locked, but no energy is delivered. **Failure postcondition:** n/a

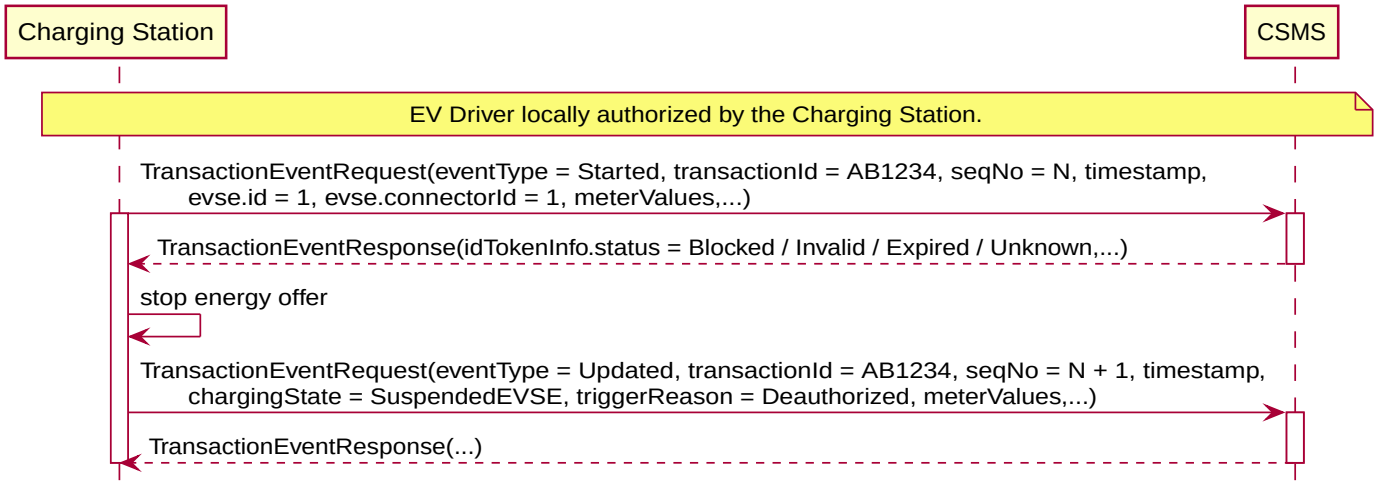

Figure 49. Sequence Diagram: Start Transaction - Id not Accepted

7	**Error handling**	n/a

8	**Remark(s)**	The scenario description and sequence diagram above are based on the Configuration Variable for start & stop transaction being configured as follows: `TxStartPoint`: Authorized, DataSigned, PowerPathClosed, EnergyTransfer `TxStopPoint`: ParkingBayOccupancy, EVConnected This use-case is also valid for other configurations, but then the transaction might start/stop at another moment, which might change the sequence in which message are sent. For more details see the use cases: E01 - Start Transaction options and E06 - Stop Transaction options.

E05 - Start Transaction - Id not Accepted - Requirements

Table 105. E05 - Requirements

ID	Precondition	Requirement definition	Note
E05.FR.01		The CSMS MUST verify validity of the identifier in the TransactionEventRequest message.	The identifier might have been authorized locally by the Charging Station using outdated information. The identifier, for instance, may have been blocked since it was added to the Charging Station's Authorization Cache.
E05.FR.02	E05.FR.01 AND The authorization status in TransactionEventResponse is not *Accepted* AND The transaction is still ongoing AND `StopTxOnInvalidId` is set to *false* AND `MaxEnergyOnInvalidId` is not implemented or has been exceeded. `TxStopPoint` does NOT contain: (PowerPathClosed OR EnergyTransfer)	The Charging Station SHALL stop the energy delivery to the EV immediately and send TransactionEventRequest (eventType = Updated) with *triggerReason* set to *ChargingStateChanged* and chargingState set to *SuspendedEVSE*	
E05.FR.03	E05.FR.01 AND The authorization status in TransactionEventResponse is not *Accepted* AND The transaction is still ongoing AND `StopTxOnInvalidId` is set to *false* AND `MaxEnergyOnInvalidId` is set and has NOT been exceeded.	Energy delivery to the EV SHALL be allowed until the amount of energy specified in `MaxEnergyOnInvalidId` has been reached.	
E05.FR.04	When a TransactionEventRequest has to be created	The Charging Station SHALL set the message's **seqNo** field as specified in Sequence Number Generation.	This enables the CSMS to track the completeness of transaction information.
E05.FR.05	When configured to send meter data in the TransactionEventRequest (eventType = Started), See: Meter Values - Configuration AND EVSE is known at start of transaction	The Charging Station SHALL add the configured measurands to the optional meterValue field with *context* = `Transaction.Begin` in the TransactionEventRequest(eventType = Started) sent to the CSMS to provide more details during the transaction.	
E05.FR.06	`AlignedDataSignReadings` is *true*	The Charging Station SHALL retrieve signed meter values and put them in the *signedMeterValue* field of sampledValues.	
E05.FR.08	When configured to send meter data in the TransactionEventRequest (eventType = Started), See: Meter Values - Configuration AND EVSE is not known at start of transaction	The Charging Station SHALL add the measurands for *eventType* = `Started` to the optional meterValue field with *context* = `Transaction.Begin` in the TransactionEventRequest(eventType = Updated) that occurs when charging starts.	

ID	Precondition	Requirement definition	Note
E05.FR.09	E05.FR.01 AND The authorization status in TransactionEventResponse is not *Accepted* AND The transaction is still ongoing AND `StopTxOnInvalidId` is *true* AND `TxStopPoint` does NOT contain: (Authorized OR PowerPathClosed OR EnergyTransfer)	The Charging Station SHALL stop the energy transfer and send TransactionEventRequest (eventType = Updated) with *triggerReason* set to *Deauthorized* and chargingState set to *SuspendedEVSE*.	
E05.FR.10	E05.FR.01 AND The authorization status in TransactionEventResponse is not *Accepted* AND The transaction is still ongoing AND `StopTxOnInvalidId` is *true* AND `TxStopPoint` does contain: (Authorized OR PowerPathClosed OR EnergyTransfer)	The Charging Station SHALL stop the transaction and send TransactionEventRequest (eventType = Ended) with *triggerReason* set to *Deauthorized* and stoppedReason set to *DeAuthorized*.	
E05.FR.11	E05.FR.10 AND If the Charging Station has the possibility to lock the Charging Cable	The Charging Station SHOULD keep the Charging Cable locked until the owner presents his identifier.	

E06 - Stop Transaction options

Table 106. E06 - Stop Transaction

No.	Type	Description
1	**Name**	Stop Transaction options
2	**ID**	E06
	Functional block	E. Transactions
3	**Objective(s)**	To inform the CSMS that a transaction at the Charging Station has stopped.
4	**Description**	This use case describes the different moment a Charging Station can stop a transaction (send TransactionEventRequest (eventType = Ended), depending on the configuration of the Charging Station.
5	*Actors*	Charging Station, CSMS, EV Driver
S1	*Scenario objective*	Stop a transaction when a parking bay occupancy no longer detector detects the EV.
	Scenario description	**1.** The Charging Stations parking bay occupancy detector stops detecting the EV. **2.** The Charging Station sends a TransactionEventRequest (eventType = Ended) notifying the CSMS about a transaction that has ended. **3.** The CSMS responds with a TransactionEventResponse, confirming that the TransactionEventRequest was received.
	Prerequisite(s)	A transaction is ongoing. Configuration Variable: `TxStopPoint` contains: ParkingBayOccupancy
	Postcondition(s)	**Successful postcondition:** The transaction is ended and the CSMS is *Successfully* informed. **Failure postcondition:** The transaction is still ongoing. *or* The CSMS is *not* informed.

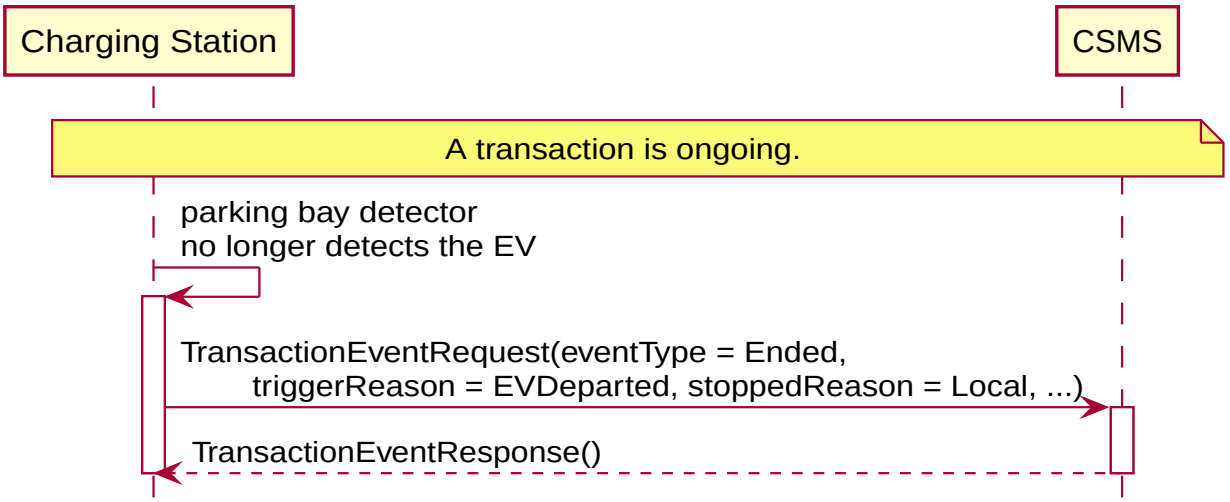

Figure 50. Sequence Diagram: Stop Transaction options - ParkingBayOccupancy

S2	*Scenario objective*	**Stop a transaction when communication between the Charging Station and the EV is lost. (for example: cable unplugged)**
	Scenario description	**1.** Communication between Charging Station and the EV is lost (Charging cable is unplugged). **2.** If charging cable unplugged on the Charging Station side: send StatusNotificationRequest to the CSMS to inform it about a Connector that became *Available*. **3.** The Charging Station sends a TransactionEventRequest (eventType = Ended) notifying the CSMS about a transaction that has ended. **4.** The CSMS responds with a TransactionEventResponse, confirming that the TransactionEventRequest was received.
	Prerequisite(s)	A transaction is ongoing. Configuration Variable: `TxStopPoint` contains: EVConnected

S2	*Scenario objective*	**Stop a transaction when communication between the Charging Station and the EV is lost. (for example: cable unplugged)**
	Postcondition(s)	**Successful postcondition:** The transaction is ended and the CSMS is *Successfully* informed. **Failure postcondition:** The transaction is still ongoing. *or* The CSMS is *not* informed.

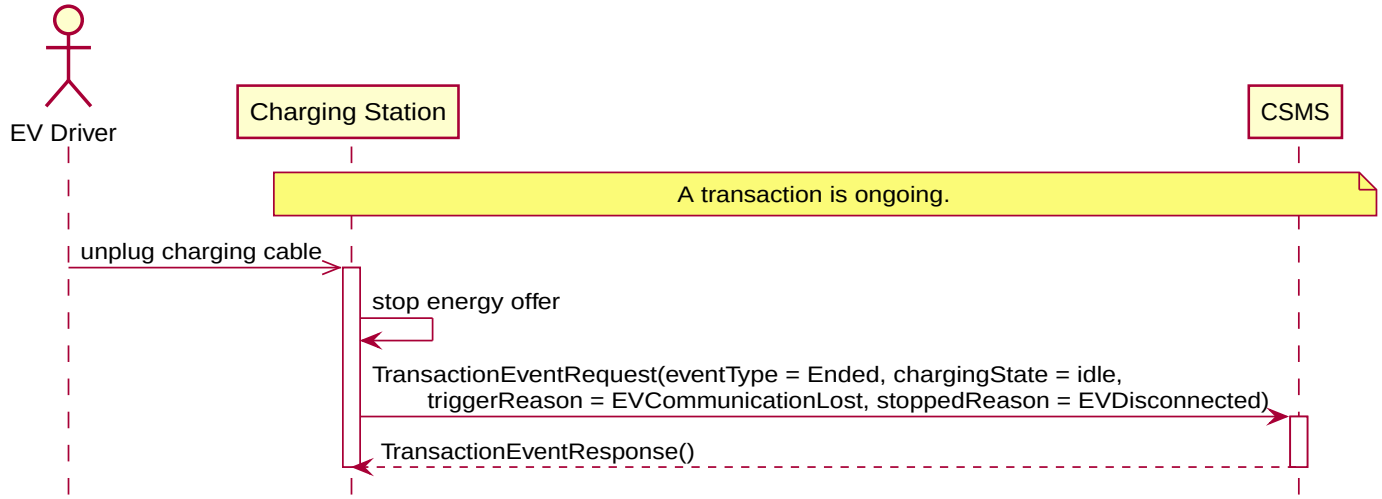

Figure 51. Sequence Diagram: Stop Transaction options - EVConnected

S3	*Scenario objective*	**Stop a transaction when the driver is no longer authorized.**
	Scenario description	**1.** The Charging Station sends a TransactionEventRequest to the CSMS. **2.** An invalid IdToken is received in a TransactionEventResponse. **3.** The Charging Station sends a TransactionEventRequest (eventType = Ended) notifying the CSMS about a transaction that has ended. **4.** The CSMS responds with a TransactionEventResponse, confirming that the TransactionEventRequest was received.
	Prerequisite(s)	A transaction is ongoing. Configuration Variable: `TxStopPoint` contains: Authorized
	Postcondition(s)	**Successful postcondition:** The transaction is ended and the CSMS is *Successfully* informed. **Failure postcondition:** The transaction is still ongoing. *or* The CSMS is *not* informed.

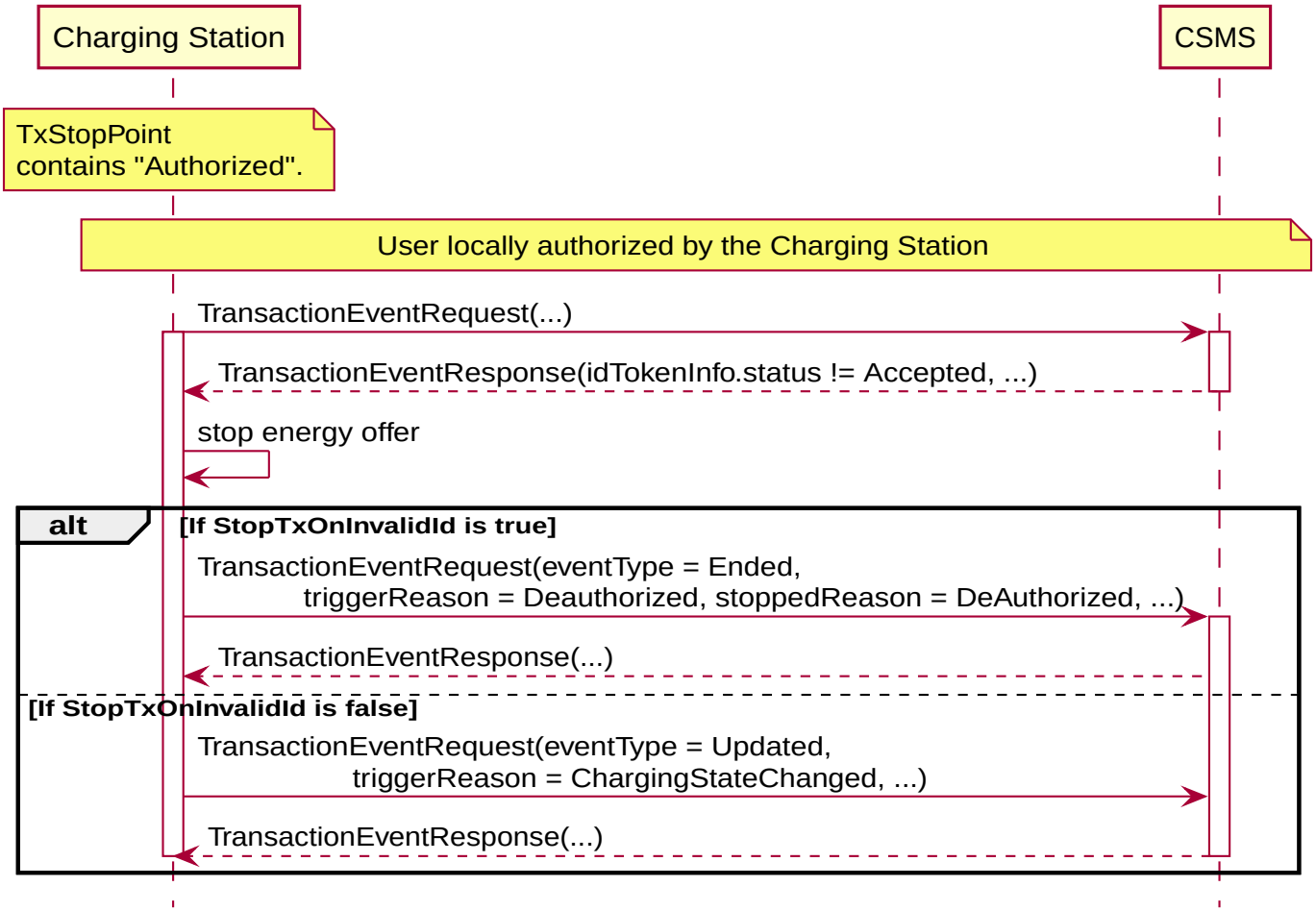

Figure 52. Sequence Diagram: Stop Transaction options - Deauthorized

S4	*Scenario objective*	**Stop a transaction when the meter stops providing signed meter values.**
	Scenario description	**1.** The Charging Station can no longer get signed meter values. **2.** The Charging Station sends a TransactionEventRequest (eventType = Ended) notifying the CSMS about a transaction that has ended. **3.** The CSMS responds with a TransactionEventResponse, confirming that the TransactionEventRequest was received.
	Prerequisite(s)	A transaction is ongoing. Configuration Variable: `TxStopPoint` contains: DataSigned
	Postcondition(s)	**Successful postcondition:** The transaction is ended and the CSMS is *Successfully* informed. **Failure postcondition:** The transaction is still ongoing. *or* The CSMS is *not* informed.

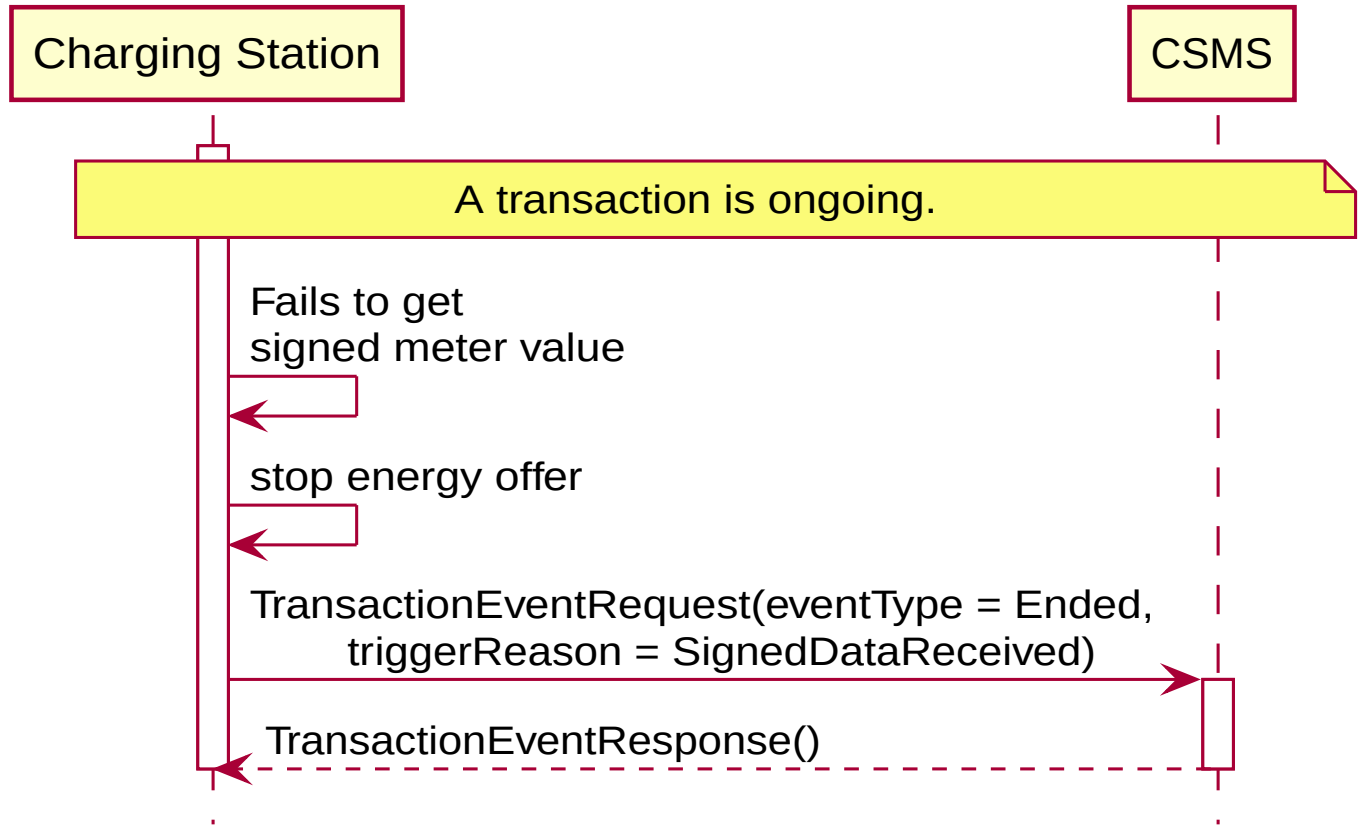

Figure 53. Sequence Diagram: Stop Transaction options - DataSigned

S5	*Scenario objective*	**Stop a transaction when the power path is no longer closed. (For example: power relay opened.)**
	Scenario description	**1.** The Charging Station opens the power relay (for any reason). **2.** The Charging Station sends a TransactionEventRequest (eventType = Ended) notifying the CSMS about a transaction that has ended. **3.** The CSMS responds with a TransactionEventResponse, confirming that the TransactionEventRequest was received.
	Prerequisite(s)	A transaction is ongoing. Configuration Variable: `TxStopPoint` contains: PowerPathClosed
	Postcondition(s)	**Successful postcondition:** The transaction is ended and the CSMS is *Successfully* informed. **Failure postcondition:** The transaction is still ongoing. *or* The CSMS is *not* informed.

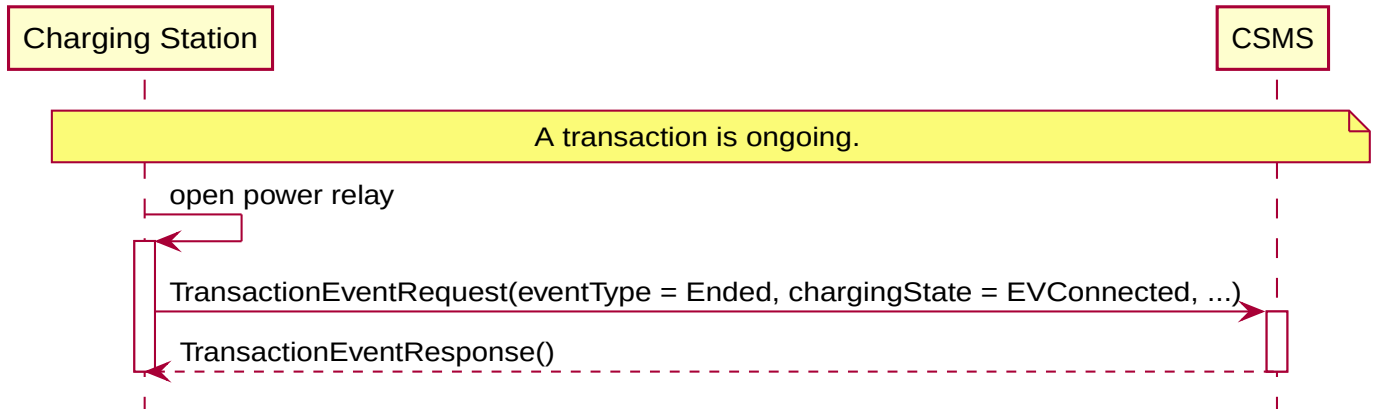

Figure 54. Sequence Diagram: Stop Transaction options - PowerPathClosed

S6	*Scenario objective*	**Stop a transaction when energy transfer stops. This will also mean the transaction stops when the EV stops taking energy, for example when the battery is to hot.**
	Scenario description	**1.** The energy transfer between EV and the Charging Station stops (for example: EV stops charging). **2.** The Charging Station sends a TransactionEventRequest (eventType = Ended) notifying the CSMS about a transaction that has ended. **3.** The CSMS responds with a TransactionEventResponse, confirming that the TransactionEventRequest was received.
	Prerequisite(s)	A transaction is ongoing. Configuration Variable: `TxStopPoint` contains: EnergyTransfer
	Postcondition(s)	**Successful postcondition:** The transaction is ended and the CSMS is *Successfully* informed. **Failure postcondition:** The transaction is still ongoing. *or* The CSMS is *not* informed.

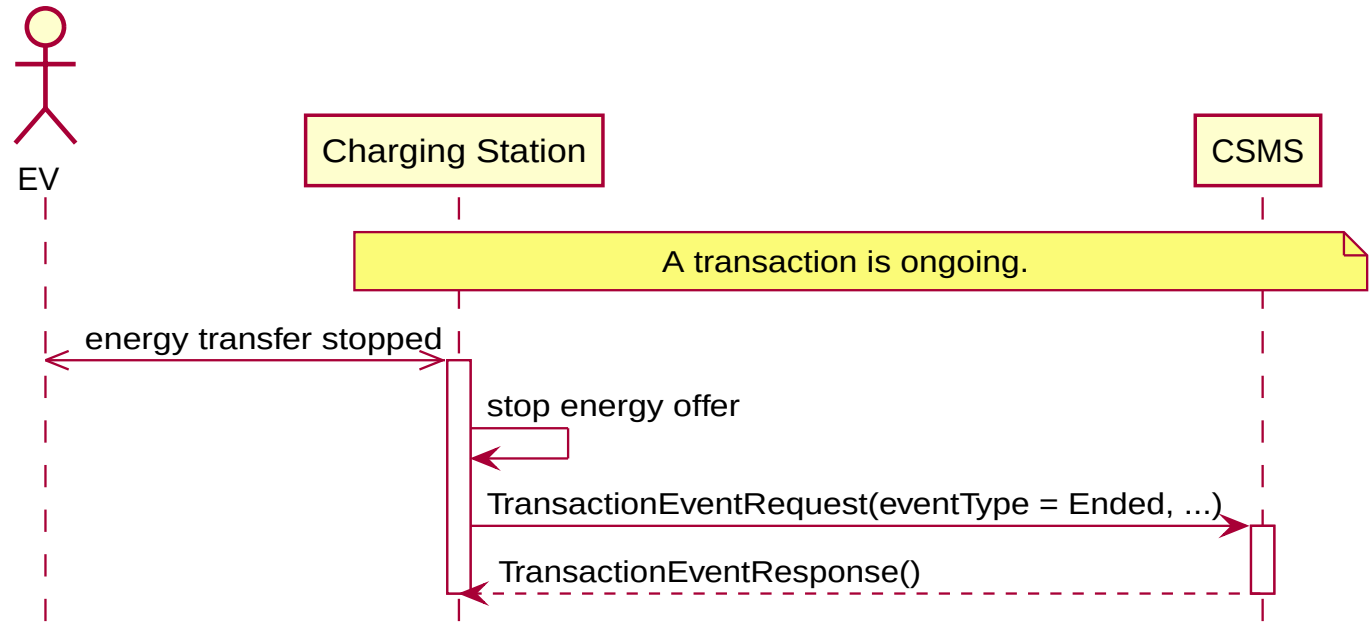

Figure 55. Sequence Diagram: Stop Transaction options - EnergyTransfer

7	**Error handling**	n/a
8	**Remark(s)**	n/a

E06 - Stop Transaction options - Requirements

Table 107. E06 - Requirements

ID	Precondition	Requirement definition
E06.FR.01	`TxStopPoint` contains: ParkingBayOccupancy AND Parking Bay Detector no longer detects the "EV"	The Charging Station SHALL stop the transaction and send a TransactionEventRequest (eventType = Ended) to the CSMS.
E06.FR.02	`TxStopPoint` contains: EVConnected AND Connection between Charging Station and EV is lost.	The Charging Station SHALL stop the transaction and send a TransactionEventRequest (eventType = Ended) to the CSMS.
E06.FR.03	`TxStopPoint` contains: Authorized AND EV Driver is authorized to stop a transaction.	The Charging Station SHALL stop the transaction and send a TransactionEventRequest (eventType = Ended) to the CSMS.

ID	Precondition	Requirement definition
E06.FR.04	`TxStopPoint` contains: Authorized AND CSMS returns a non-valid idTokenInfo in a TransactionEventResponse	The Charging Station SHALL stop the transaction and send a TransactionEventRequest (eventType = Ended) to the CSMS.
E06.FR.05	`TxStopPoint` contains: DataSigned AND Charging Station can no longer retrieve signed meter values.	The Charging Station SHALL stop the transaction and send a TransactionEventRequest (eventType = Ended) to the CSMS.
E06.FR.06	`TxStopPoint` contains: PowerPathClosed AND Power relay is opened	The Charging Station SHALL stop the transaction and send a TransactionEventRequest (eventType = Ended) to the CSMS.
E06.FR.07	`TxStopPoint` contains: EnergyTransfer AND Energy transfer stops	The Charging Station SHALL stop the transaction and send a TransactionEventRequest (eventType = Ended) to the CSMS.
E06.FR.08	If a transaction is not ended by the EV Driver at the Charging Station	The Charging Station SHALL include the stoppedReason element in the TransactionEventRequest(eventType = Ended). What reason to use is described in the description of reasonEnumType.
E06.FR.09	If a transaction is ended by the EV Driver at the Charging Station (e.g. EV Driver presented IdToken to stop the transaction)	The Charging Station MAY omit the stoppedReason element in the TransactionEventRequest (eventType = Ended) (hence the CSMS can interpret the reason as local when omitted).
E06.FR.10	As part of the normal transaction termination.	The Charging Station SHALL unlock the cable (if not permanently attached).
E06.FR.11	When configured to send meter data in the TransactionEventRequest (eventType = Started), See: Meter Values - Configuration AND EVSE is known at start of transaction	The Charging Station SHALL add the configured measurands to the optional meterValue field with *context* = `Transaction.Begin` in the TransactionEventRequest(eventType = Started) sent to the CSMS to provide more details during the transaction.
E06.FR.12	E06.FR.11 AND The Charging Station is running low on memory	The Charging Station MAY drop meter data in the TransactionEventRequest(eventType = Ended) message.
E06.FR.13	E06.FR.12	When dropping meter data, the Charging Station SHALL drop intermediate values first (1st value, 3th value, 5th etc), not start dropping values from the start of the list or stop adding values to the list.
E06.FR.14	When a TransactionEventRequest has to be created	The Charging Station SHALL set the message's **seqNo** field as specified in Sequence Number Generation.
E06.FR.15	When sending a TransactionEventRequest	The Charging Station SHALL set the triggerReason to inform the CSMS about what triggered the event. What reason to use is described in the description of TriggerReasonEnumType.
E06.FR.16	A transaction was stopped by an Abnormal Error or Fault Condition.	The Charging Station SHALL send TransactionEventRequest(eventType = Ended, triggerReason=AbnormalCondition)_ to the CSMS.
E06.FR.17	When configured to send meter data in the TransactionEventRequest (eventType = Started), See: Meter Values - Configuration AND EVSE is not known at start of transaction	The Charging Station SHALL add the measurands for *eventType* = `Started` to the optional meterValue field with *context* = `Transaction.Begin` in the TransactionEventRequest(eventType = Updated) that occurs when charging starts.

E07 - Transaction locally stopped by IdToken

Table 108. E07 - Transaction locally stopped by IdToken

No.	Type	Description
1	**Name**	Transaction locally stopped by IdToken
2	**ID**	E07
	Functional block	E. Transactions
3	**Objective(s)**	The EV Driver wants to stop an ongoing transaction, by locally presenting his IdToken.

No.	Type	Description
4	**Description**	This use case covers how the EV Driver can stop a transaction when he wants to leave the charging station.
	Actors	Charging Station, CSMS, EV Driver
	Scenario description	**1.** The EV Driver is authorized by the Charging Station and/or CSMS. **2.** If the cable is not permanently attached, the Charging Station unlocks the cable. **3.** The Charging Station sends a TransactionEventRequest (eventType = Updated) with trigger *StopAuthorized* **4.** The CSMS responds with a TransactionEventResponse. **5.** The EV Driver unplugs the cable (and drives away the EV). **6.** The Charging Station sends a StatusNotificationRequest with status *Available*, notify the CSMS that the Connector is available again. **7.** The CSMS responds with a StatusNotificationResponse. **8.** The Charging Station sends a TransactionEventRequest (eventType = Ended) **9.** The CSMS responds with a TransactionEventResponse.
	Alternative scenario(s)	**1.** The Charging Station MAY unlock the cable (if not permanently attached) when the cable is disconnected at the EV. If supported, this functionality is reported and controlled by the Configuration Variable `UnlockOnEvSideDisconnect`. **2.** The Charging Station MAY stop an ongoing transaction when the cable is disconnected at the EV. If supported, this functionality is reported and controlled by the Configuration Variable `StopTxOnEVSideDisconnect`. E07 - Offline Stop Transaction E08 - When cable disconnected on EV-side: Stop Transaction E09 - When cable disconnected on EV-side: Suspend Transaction
5	**Prerequisite(s)**	A transaction is ongoing.
6	**Postcondition(s)**	**Successful postcondition:** The CSMS has received all relevant information about the transaction and the Charging Station is in *Idle* status. **Failure postcondition:** The transaction is still ongoing or the Charging Station is in Idle status and still holds information about the transaction that it has to deliver to the CSMS.

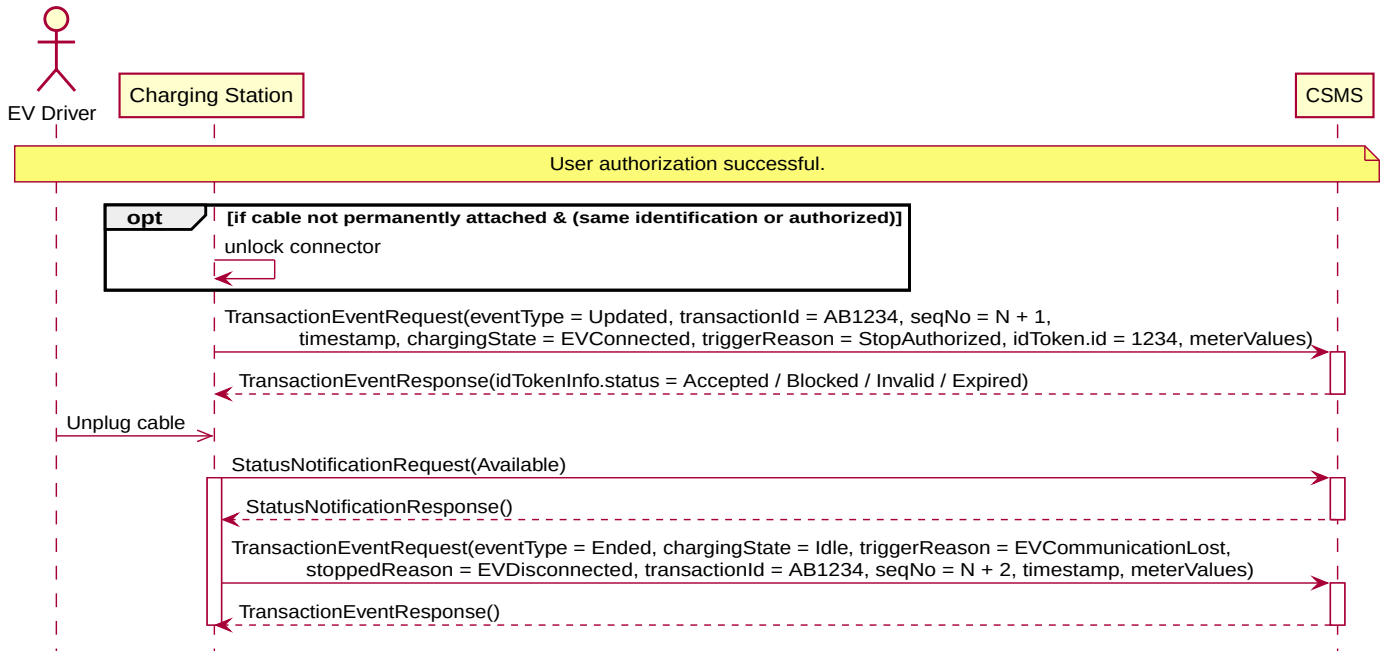

Figure 56. Sequence Diagram: Transaction locally stopped by IdToken

7	**Error handling**	n/a

8	**Remark(s)**	It is likely that the CSMS applies sanity checks to the data contained in TransactionEventRequest it received. The outcome of such sanity checks SHOULD NOT ever cause the CSMS to not respond with a TransactionEventResponse. The scenario description and sequence diagram above are based on the Configuration Variable for stop transaction being configured as follows. `TxStopPoint`: EVConnected This use-case is also valid for other configurations, but then the transaction might stop at another moment, which might change the sequence in which message are sent. For more details see the use case: E06 - Stop Transaction options The CSMS cannot prevent a transaction from stopping.

E07 - Transaction locally stopped by IdToken - Requirements

Table 109. E07 - Requirements

ID	Precondition	Requirement definition	Note
E07.FR.01		The CSMS SHALL only inform the Charging Station it has received TransactionEventRequest.	
E07.FR.02	E07.FR.01 and when stopping a transaction.	The CSMS MAY send information about the IdTokenType used to stop the transaction.	
E07.FR.03		The IdTokenType in the request message MAY be omitted when the Charging Station itself needs to stop the transaction.	e.g. when the Charging Station is requested to reset.
E07.FR.04	If a transaction is ended in a normal way.	The stoppedReason element MAY be omitted.	e.g. EV-driver presented IdToken to stop the transaction.
E07.FR.05	If a transaction is ended in a normal way	The stoppedReason SHOULD be assumed 'Local'.	e.g. EV-driver presented IdToken to stop the transaction.
E07.FR.06	If the transaction is *not* ended normally.	stoppedReason SHOULD be set to a correct value.	
E07.FR.07	As part of the normal transaction termination.	The Charging Station SHALL unlock the cable (if not permanently attached).	
E07.FR.08	When configured to send meter data in the TransactionEventRequest (eventType = Started), See: Meter Values - Configuration AND EVSE is known at start of transaction	The Charging Station SHALL add the configured measurands to the optional meterValue field with *context* = `Transaction.Begin` in the TransactionEventRequest(eventType = Started) sent to the CSMS to provide more details during the transaction.	
E07.FR.09	E07.FR.08 AND The Charging Station is running low on memory	The Charging Station MAY drop meter data in the TransactionEventRequest(eventType = Ended) message.	
E07.FR.10	E07.FR.09	When dropping meter data, the Charging Station SHALL drop intermediate values first (1st value, 3th value, 5th etc), not start dropping values from the start of the list or stop adding values to the list.	
E07.FR.11	When a TransactionEventRequest has to be created	The Charging Station SHALL set the message's **seqNo** field as specified in Sequence Number Generation.	This enables the CSMS to track the completeness of transaction information
E07.FR.12	`AlignedDataSignReadings` is *true*	The Charging Station SHALL retrieve signed meter values and put them in the *signedMeterValue* field of sampledValues.	

ID	Precondition	Requirement definition	Note
E07.FR.13	When configured to send meter data in the TransactionEventRequest (eventType = Started), See: Meter Values - Configuration AND EVSE is not known at start of transaction	The Charging Station SHALL add the measurands for *eventType* = `Started` to the optional meterValue field with *context* = `Transaction.Begin` in the TransactionEventRequest(eventType = Updated) that occurs when charging starts.	

E08 - Transaction stopped while Charging Station is offline

Table 110. E08 - Transaction stopped while Charging Station is offline

No.	Type	Description
1	**Name**	Transaction stopped while Charging Station is offline
2	**ID**	E08
	Functional block	E. Transactions
	Parent use case	E07 - Local Stop Transaction
3	**Objective(s)**	To enable the EV Driver to stop a transaction while the Charging Station is *Offline*.
4	**Description**	This use case describes how an EV Driver can stop a transaction while the Charging Station is *Offline*. While a transaction is ongoing and the Charging Station is *Offline*, the EV Driver presents his IdToken, if the Charging Stations knows locally (without asking the CSMS) that this IdToken is allowed to stop the transaction, it will stop the ongoing transaction. When the Charging Station restores the connection with the CSMS, it needs to send the information about this *Offline* stop transaction to the CSMS.
	Actors	Charging Station, CSMS, EV Driver
	Scenario description	**1.** The EV Driver presents IdToken to stop the transaction. **2.** When this is the same IdToken as was used to start the transaction, or via the Local Authorization List and / or Authorization Cache the GroupId can be validated: the transaction is stopped. **3.** The Charging Station stops the energy offer. **4.** The TransactionEventRequest (eventType = Ended) is stored/queued by the Charging Station. **5.** The connection between Charging Station and CSMS is restored. **6.** The Charging Station starts to send queued messages **7.** The stored TransactionEventRequest is sent, notifying the CSMS about the transaction that was stopped.
5	**Prerequisite(s)**	Transaction ongoing and connection lost.
6	**Postcondition(s)**	Charging Station is in *Idle* status.

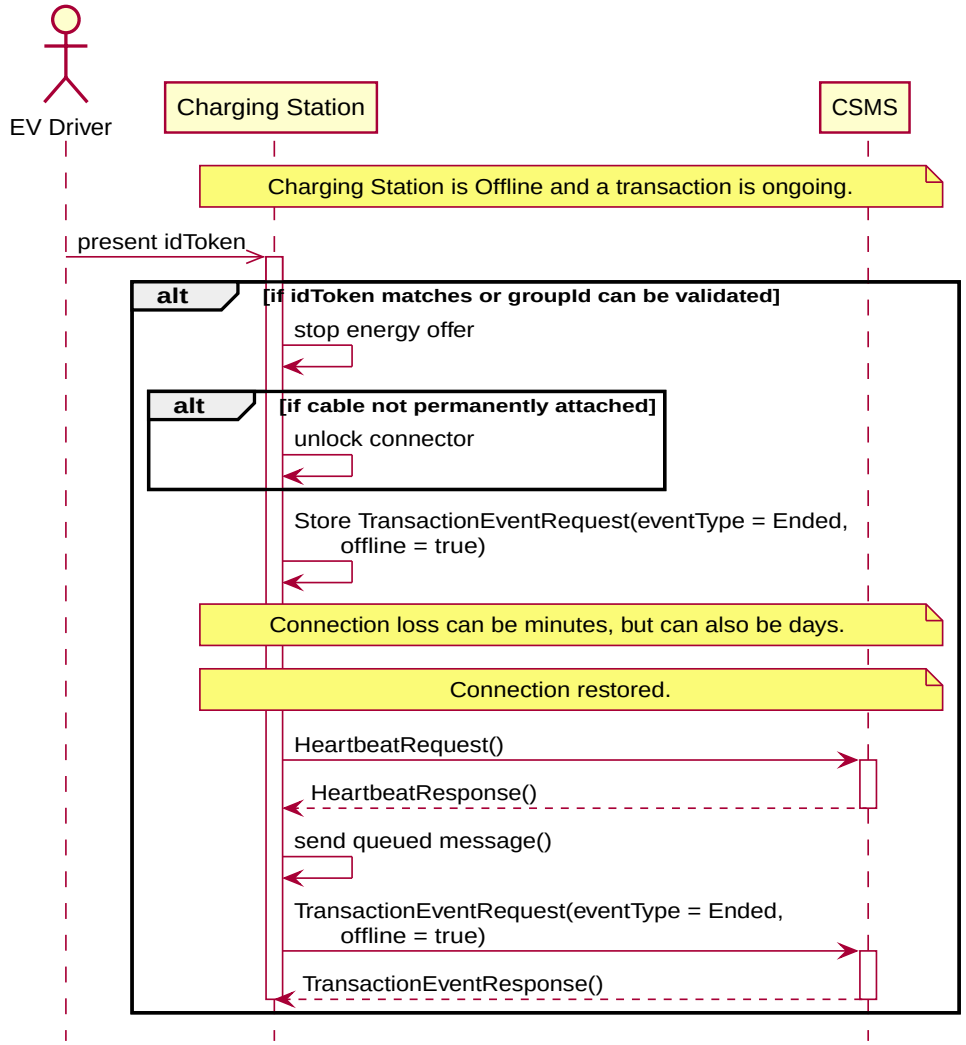

Figure 57. Sequence Diagram: Transaction stopped while Charging Station is offline

7	**Error handling**	n/a
8	**Remark(s)**	groupId check must be done on Local Authorization List and / or Authorization Cache if available. The scenario description and sequence diagram above are based on the Configuration Variable for stop transaction being configured as follows. `TxStopPoint`: ParkingBayOccupancy, EVConnected, Authorized This use-case is also valid for other configurations, but then the transaction might stop at another moment, which might change the sequence in which message are sent. For more details see the use case: E06 - Stop Transaction options

E08 - Transaction stopped while Charging Station is offline - Requirements

Table 111. E08 - Requirements

ID	Precondition	Requirement definition	Note
E08.FR.01	If the IdToken presented is the same as the IdToken used to start the transaction.	The Charging Station SHALL stop the energy offering.	
E08.FR.02	If the IdToken presented has the same GroupId as the IdToken used to start the transaction.	The Charging Station SHALL stop the energy offering.	
E08.FR.03	(E08.FR.01 OR E08.FR.02) AND Cable not permanently attached	The Charging Station SHALL unlock the connector.	
E08.FR.04	(E08.FR.01 OR E08.FR.02)	The Charging Station SHALL "generate" a TransactionEventRequest (eventType = Ended).	

ID	Precondition	Requirement definition	Note
E08.FR.05	When *Offline*.	The Charging Station MUST queue any TransactionEventRequest messages.	
E08.FR.06	After the connection is restored.	The Charging Station MUST send queued TransactionEventRequest messages.	
E08.FR.07		The flag: *offline* SHALL be set to TRUE for any TransactionEventRequest that occurred while the Charging Station was offline.	
E08.FR.08	When a TransactionEventRequest has to be created	The Charging Station SHALL set the message's **seqNo** field as specified in Sequence Number Generation.	This enables the CSMS to track the completeness of transaction information.
E08.FR.09	When configured to send meter data in the TransactionEventRequest (eventType = Ended), See: Meter Values - Configuration	The Charging Station SHALL add the configured measurands to the optional meterValue field in the TransactionEventRequest(eventType = Ended) sent to the CSMS to provide more details about transaction usage.	
E08.FR.10	E08.FR.09 AND The Charging Station is running low on memory	The Charging Station MAY drop meter data in the TransactionEventRequest(eventType = Ended) message.	
E08.FR.11	E08.FR.10	When dropping meter data, the Charging Station SHALL drop intermediate values first (1st value, 3th value, 5th etc), not start dropping values from the start of the list or stop adding values to the list.	
E08.FR.12	`AlignedDataSignReadings` is *true*	The Charging Station SHALL retrieve signed meter values and put them in the *signedMeterValue* field of sampledValues.	

E09 - When cable disconnected on EV-side: Stop Transaction

Table 112. E09 - When cable disconnected on EV-side: Stop Transaction

No.	Type	Description
1	**Name**	When cable disconnected on EV-side: Stop Transaction
2	**ID**	E09
	Functional block	E. Transactions
	Parent use case	E07 - Local Stop Transaction
3	**Objective(s)**	To stop an ongoing transaction when the Charging Cable is unplugged on the EV side.
4	**Description**	This use case covers how a transaction is stopped when the EV Driver unplugs the cable at the EV side. In this use case the Configuration Variable: `StopTxOnEVSideDisconnect` = true. The Charging Cable is unplugged at the EV side. This is detected by the Charging Station. The Charging Station stops the transaction and sends a TransactionEventRequest to the CSMS. The Charging Cable, if locked and `UnlockOnEvSideDisconnect` = false, will remain locked at the Charging Station until the EV Driver returns and presents his/hers IdToken. Otherwise it will unlock the cable.
	Actors	Charging Station, CSMS, EV Driver
	Scenario description	**1.** The cable is unplugged at the EV. **2.** The energy offer is suspended. **3.** The Charging Station sends TransactionEventRequest (eventType = Ended, stoppedReason = EVDisconnected) to the CSMS. **4.** The CSMS responds with TransactionEventResponse. **5.** The EV Driver is authorized and unplugs the cable. **6.** The Charging Station sends StatusNotificationRequest to the CSMS with the status *Available*. **7.** The CSMS responds with StatusNotificationResponse.
	Alternative scenario(s)	E09 - When cable disconnected on EV-side: Suspend Transaction
5	**Prerequisite(s)**	Configuration Variable: `StopTxOnEVSideDisconnect` = true A transaction is ongoing
6	**Postcondition(s)**	**Successful postcondition:** The Charging Station is in *Idle* status. **Failure postcondition:** n/a

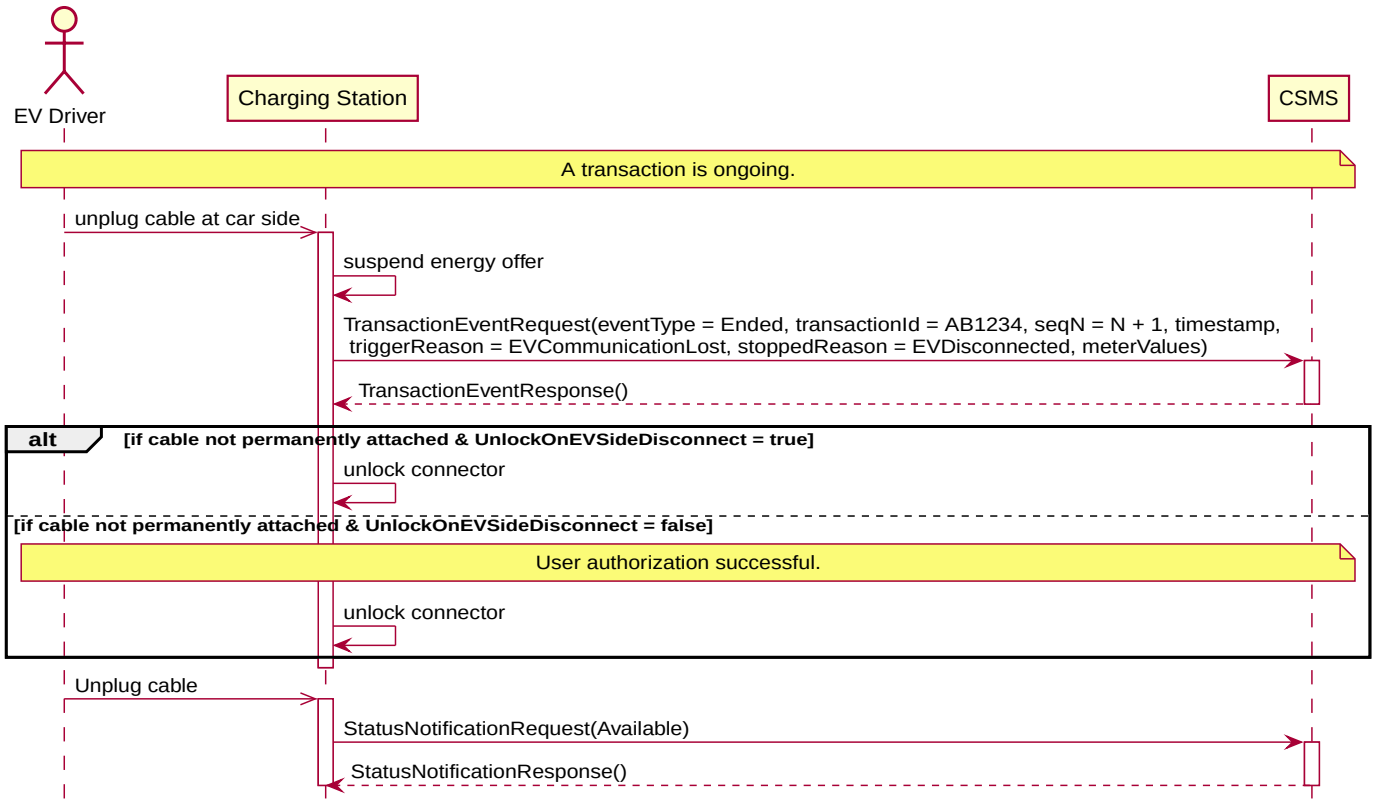

Figure 58. Sequence Diagram: When cable disconnected on EV-side: Stop Transaction

7	**Error handling**	n/a
8	**Remark(s)**	When the Charging Cable is plugged back in, the charging will not resume/continue. The scenario description and sequence diagram above are based on the Configuration Variable for stop transaction being configured as follows. `TxStopPoint`: Authorized This use-case is also valid for other configurations, but then the transaction might stop at another moment, which might change the sequence in which message are sent. For more details see the use case: E06 - Stop Transaction options

E09 - When cable disconnected on EV-side: Stop Transaction - Requirements

Table 113. E09 - Requirements

ID	Precondition	Requirement definition	Note
E09.FR.01	If `StopTxOnEVSideDisconnect` = *true* .	The transaction SHALL be deauthorized when the cable is disconnected from the EV. If the EV is reconnected, energy transfer is not allowed until the transaction is authorized once again.	Setting `StopTxOnEVSideDisconnect` to *true* will prevent sabotage acts when unplugging not locked cables on EV side.
E09.FR.02	E09.FR.01 AND the cable is not permanently attached AND `UnlockOnEvSideDisconnect` = true.	The Charging Station SHALL unlock the Charging Cable.	

ID	Precondition	Requirement definition	Note
E09.FR.03	E09.FR.01 AND the cable is not permanently attached AND `UnlockOnEvSideDisconnect` = false.	The Charging Station SHALL unlock the Charging Cable only after authorization by the EV Driver.	
E09.FR.04	When a TransactionEventRequest has to be created	The Charging Station SHALL set the message's **seqNo** field as specified in Sequence Number Generation.	This enables the CSMS to track the completeness of transaction information
E09.FR.05	When configured to send meter data in the TransactionEventRequest (eventType = Ended), See: Meter Values - Configuration	The Charging Station SHALL add the configured measurands to the optional meterValue field in the TransactionEventRequest(eventType = Ended) sent to the CSMS to provide more details about transaction usage.	
E09.FR.06	E09.FR.05 AND The Charging Station is running low on memory	The Charging Station MAY drop meter data in the TransactionEventRequest(eventType = Ended) message.	
E09.FR.07	E09.FR.06	When dropping meter data, the Charging Station SHALL drop intermediate values first (1st value, 3th value, 5th etc), not start dropping values from the start of the list or stop adding values to the list.	
E09.FR.08	`AlignedDataSignReadings` is *true*	The Charging Station SHALL retrieve signed meter values and put them in the *signedMeterValue* field of sampledValues.	

E10 - When cable disconnected on EV-side: Suspend Transaction

Table 114. E10 - When cable disconnected on EV-side: Suspend Transaction

No.	Type	Description
1	**Name**	When cable disconnected on EV-side: Suspend Transaction
2	**ID**	E10
	Functional block	E. Transactions
	Parent use case	E07 - Local Stop Transaction
3	**Objective(s)**	To suspend an ongoing transaction when the Charging Cable is unplugged on the EV side.
4	**Description**	This use case covers how a transaction is suspended when the EV Driver unplugs the cable at the EV side. In this use case the Configuration Variable: `StopTxOnEVSideDisconnect` = false. The Charging Cable is unplugged at the EV side. This is detected by the Charging Station. The Charging Station stops the energy offering (safety), but does not stop the transaction. The Charging Cable, if locked, will remain locked at the Charging Station until the EV Driver returns and presents his/hers IdToken.
	Actors	Charging Station, CSMS, EV Driver
	Scenario description	**1.** EV Driver unplugs the cable at the EV while a transaction is ongoing. **2.** The energy offer is suspended. *If the EV Driver plugs the cable back in, the transaction is resumed.* **A1.** The Charging Station sends a TransactionEventRequest (eventType = Updated, trigger = CablePluggedIn) **A2.** The CSMS responds with a TransactionEventResponse. *If cable not permanently attached* **B1.** The EV Driver is authorized by the Charging Station and/or CSMS to unlock the charging cable. **B2.** The Cable is unlocked. **B3.** The Charging Station sends a TransactionEventRequest (eventType = Ended, trigger = StopAuthorized). **B4.** The EV Driver removes the charging cable. **B5.** The Charging Station sends a StatusNotificationRequest to the CSMS with the status *Available*. **B6.** The CSMS responds with a StatusNotificationResponse. *If cable permanently attached* **C1.** The Cable is not plugged in within timeout. **C2.** The Charging Station sends a TransactionEventRequest (eventType = Ended, trigger = EVCommunicationLost, stoppedReason = EVDisconnected). **C3.** The Charging Station sends a StatusNotificationRequest to the CSMS with the status *Available*. **C4.** The CSMS responds with a StatusNotificationResponse.
	Alternative scenario(s)	E09 - When cable disconnected on EV-side: Stop Transaction
5	**Prerequisite(s)**	Configuration Variable: `StopTxOnEVSideDisconnect` = false A transaction is ongoing
6	**Postcondition(s)**	**Successful postcondition:** The Charging Station is in *Idle* status. The regular transaction is resumed. **Failure postcondition:** n/a

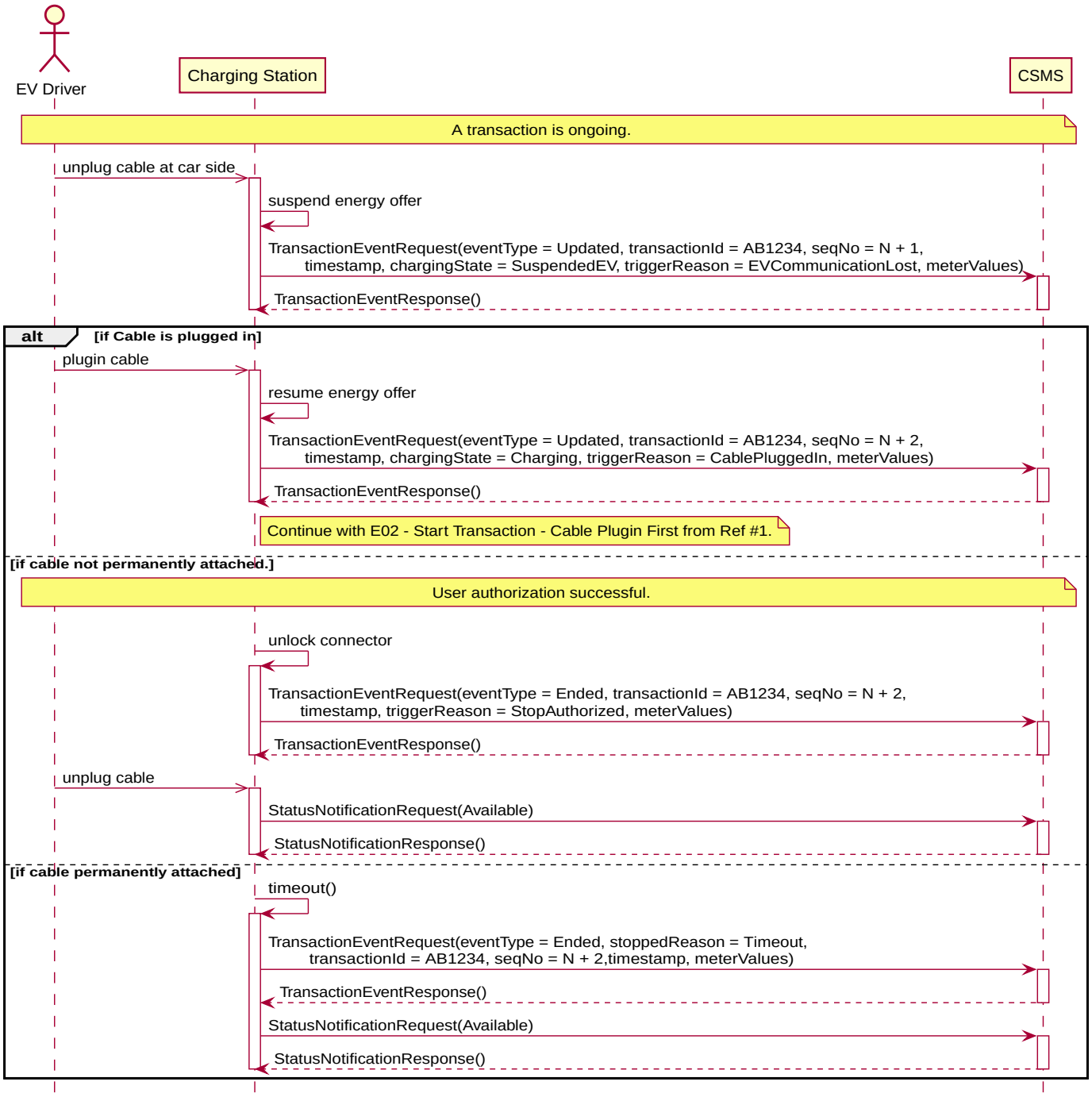

Figure 59. Sequence Diagram: When cable disconnected on EV-side: Suspend Transaction

7	**Error handling**	n/a
8	**Remark(s)**	When the Charging Cable is plugged back in, the charging is resumed. When the cable is permanently attached and the cable is not plugged in within a certain timeout, the Charging Station stops the transaction. This timeout is not defined by OCPP, it is left to the implementor of the Charging Station. The scenario description and sequence diagram above are based on the Configuration Variable for stop transaction being configured as follows. `TxStopPoint`: ParkingBayOccupancy, Authorized This use-case is also valid for other configurations, but then the transaction might stop at another moment, which might change the sequence in which message are sent. For more details see the use case: E06 - Stop Transaction options

E10 - When cable disconnected on EV-side: Suspend Transaction - Requirements

Table 115. E10 - Requirements

ID	Precondition	Requirement definition	Note
E10.FR.01	Cable not permanently attached	The Connector SHALL remain locked at the Charging Station until the EV Driver presents the IdToken.	
E10.FR.02	Cable permanently attached AND Cable not plugged in within timeout	The Charging Station SHALL deauthorize the transaction.	
E10.FR.03	When a TransactionEventRequest has to be created	The Charging Station SHALL set the message's **seqNo** field as specified in Sequence Number Generation.	This enables the CSMS to track the completeness of transaction information
E10.FR.04	When configured to send meter data in the TransactionEventRequest (eventType = Ended), See: Meter Values - Configuration	The Charging Station SHALL add the configured measurands to the optional meterValue field in the TransactionEventRequest(eventType = Ended) sent to the CSMS to provide more details about transaction usage.	
E10.FR.05	E10.FR.04 AND The Charging Station is running low on memory	The Charging Station MAY drop meter data in the TransactionEventRequest(eventType = Ended) message.	
E10.FR.06	E10.FR.05	When dropping meter data, the Charging Station SHALL drop intermediate values first (1st value, 3th value, 5th etc), not start dropping values from the start of the list or stop adding values to the list.	
E10.FR.07	`AlignedDataSignReadings` is *true*	The Charging Station SHALL retrieve signed meter values and put them in the *signedMeterValue* field of sampledValues.	

E11 - Connection Loss During Transaction

Table 116. E11 - Connection Loss During Transaction

No.	Type	Description
1	**Name**	Connection Loss During Transaction
2	**ID**	E11
	Functional block	E. Transactions
3	**Objective(s)**	To enable a Charging Station to continue a transaction while the Charging Station loses its connection
4	**Description**	This use cases describes how a Charging Station can continue an ongoing transaction while losing and regaining the connection with the CSMS.
	Actors	Charging Station, CSMS
	Scenario description	**1.** The connection of the Charging Station is lost, while a transaction is ongoing. **2.** The transaction events of the Charging Station are stored. **3.** The connection with the CSMS is restored. **4.** The Charging Station sends the stored transaction events to the CSMS using TransactionEventRequest (offline = TRUE). **5.** The Charging Station resumes regular communication.
	Alternative scenario(s)	E04 - Offline Start Transaction
5	**Prerequisite(s)**	Transaction ongoing and connection lost.
6	**Postcondition(s)**	**Successful postcondition:** The Charging Station resumes regular communication. **Failure postcondition:** n/a

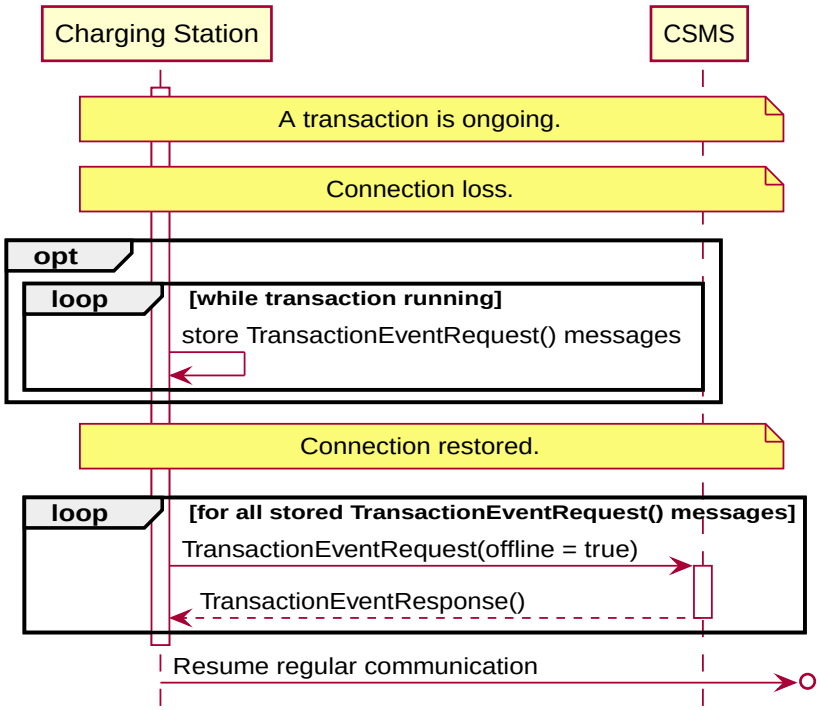

Figure 60. Sequence Diagram: Connection Loss During Transaction

7	**Error handling**	n/a
8	**Remark(s)**	n/a

E11 - Connection Loss During Transaction - Requirements

Table 117. E11 - Requirements

ID	Precondition	Requirement definition
E11.FR.01	When *Offline*	The Charging Station MUST queue all TransactionEventRequest messages, that it would have sent to the CSMS if the Charging Station had been online.
E11.FR.02	After the connection is restored.	The Charging Station MUST send queued TransactionEventRequest messages with the flag *offline* set to TRUE.
E11.FR.03	When configured to send meter data in the TransactionEventRequest(eventType = Updated), See: Meter Values - Configuration	The Charging Station SHALL add the configured measurands to the optional meterValue field in the TransactionEventRequest (eventType = Updated) sent to the CSMS to provide more details during the transaction.
E11.FR.04	E11.FR.03 AND *Offline* AND The Charging Station is running low on memory	The Charging Station MAY drop TransactionEventRequest (eventType = Updated) messages.
E11.FR.05	E11.FR.04	When dropping TransactionEventRequest(eventType = Updated) messages, the Charging Station SHALL drop intermediate messages first (1st message, 3th message, 5th message etc.), not start dropping messages from the start or stop adding messages to the queue.
E11.FR.06	E11.FR.03 AND Amount of meter data is too much for 1 TransactionEventRequest(eventType = Updated)	The Charging Station MAY split the meter data over multiple TransactionEventRequest(eventType = Updated) messages with the same *timestamp*.
E11.FR.07		If the Charging Station goes offline, every message that is still in the queue SHALL be set *Offline*.
E11.FR.08	`AlignedDataSignReadings` is *true*	The Charging Station SHALL retrieve signed meter values and put them in the *signedMeterValue* field of sampledValues.

E12 - Inform CSMS of an Offline Occurred Transaction

Table 118. E12 - Inform CSMS of an Offline Occurred Transaction

No.	Type	Description
1	**Name**	Inform CSMS of an Offline Occurred Transaction
2	**ID**	E12
	Functional block	E. Transactions
3	**Objective(s)**	To enable the Charging Station to inform the CSMS that a transaction occurred while the Charging Station was *Offline*.
4	**Description**	This use case covers how the Charging Station starts and stops a transaction since connection loss.
	Actors	Charging Station, CSMS
	Scenario description	**1.** The connection with the CSMS is restored. **2.** The Charging Station sends a Heartbeat message to the CSMS. **3.** The Charging Station sends TransactionEventRequest (eventType = Started, offline = TRUE) to the CSMS. **4.** The CSMS responds with TransactionEventResponse, accepting the transaction. **5.** The Charging Station sends TransactionEventRequest (eventType = Updated, offline = TRUE) **6.** The CSMS responds with TransactionEventResponse. **7.** The Charging Station sends TransactionEventRequest (eventType = Ended, offline = TRUE) **8.** The CSMS responds with TransactionEventResponse.
5	**Prerequisite(s)**	At least one *Offline* transaction has taken place.
6	**Postcondition(s)**	**Successful postcondition:** The CSMS has processed all transactions that occurred *Offline*. **Failure postcondition:** n/a

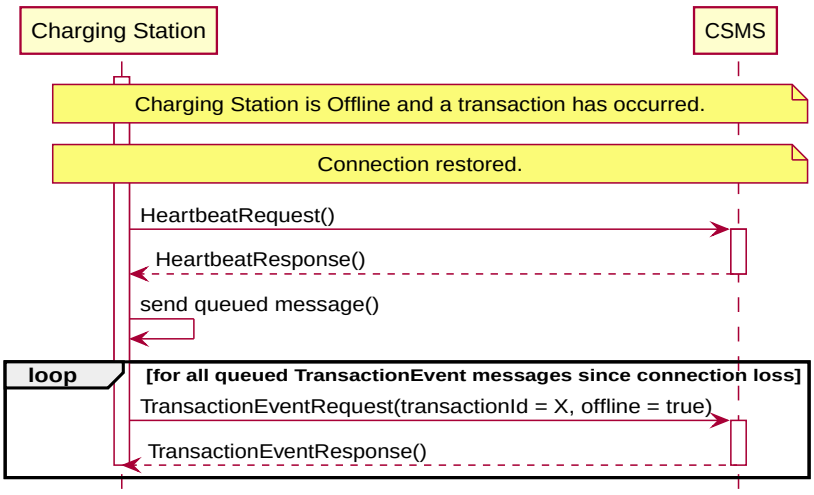

Figure 61. Sequence Diagram: Inform CSMS of an Offline Occurred Transaction

7	**Error handling**	n/a
8	**Remark(s)**	n/a

E12 - Inform CSMS of an Offline Occurred Transaction - Requirements

Table 119. E12 - Requirements

ID	Precondition	Requirement definition
E12.FR.01	When *Offline*	The Charging Station MUST queue all TransactionEventRequest messages, that it would have sent to the CSMS if the Charging Station had been online.

ID	Precondition	Requirement definition
E12.FR.02	After the connection is restored.	The Charging Station MUST send queued TransactionEventRequest messages with the flag *offline* set to TRUE.
E12.FR.03	When configured to send meter data in the TransactionEventRequest(eventType = Updated), See: Meter Values - Configuration	The Charging Station SHALL add the configured measurands to the optional meterValue field in the TransactionEventRequest (eventType = Updated) sent to the CSMS to provide more details during the transaction.
E12.FR.04	E12.FR.03 AND *Offline* AND The Charging Station is running low on memory	The Charging Station MAY drop TransactionEventRequest (eventType = Updated) messages.
E12.FR.05	E12.FR.04	When dropping TransactionEventRequest(eventType = Updated) messages, the Charging Station SHALL drop intermediate messages first (1st message, 3th message, 5th message etc.), not start dropping messages from the start or stop adding messages to the queue.
E12.FR.06	E12.FR.03 AND Amount of meter data is too much for 1 TransactionEventRequest(eventType = Updated)	The Charging Station MAY split the meter data over multiple TransactionEventRequest(eventType = Updated) messages with the same *timestamp*.
E12.FR.07	When configured to send meter data in the TransactionEventRequest(eventType = Ended), See: Meter Values - Configuration	The Charging Station SHALL add the configured measurands to the optional meterValue field in the TransactionEventRequest (eventType = Ended) sent to the CSMS to provide more details about transaction usage.
E12.FR.08	E12.FR.07 AND The Charging Station is running low on memory	The Charging Station MAY drop meter data in the TransactionEventRequest(eventType = Ended) message.
E12.FR.09	E12.FR.08	When dropping meter data, the Charging Station SHALL drop intermediate values first (1st value, 3th value, 5th etc), not start dropping values from the start of the list or stop adding values to the list.
E12.FR.10	`AlignedDataSignReadings` is *true*	The Charging Station SHALL retrieve signed meter values and put them in the *signedMeterValue* field of sampledValues.

E13 - Transaction-related message not accepted by CSMS

Table 120. E13 - Transaction-related message not accepted by CSMS

No.	Type	Description
1	**Name**	Transaction-related message not accepted by CSMS
2	**ID**	E13
	Functional block	E. Transactions
3	**Objective(s)**	To define how a Charging Station shall handle not accepted messages.
4	**Description**	There are a situation/issues why a CSMS might not accept a transaction related message, or does not reply within the MessageTimeout. Most are error scenarios. When something like this happens, the Charging Station SHALL retry the messages a couple of times.
	Actors	Charging Station, CSMS
	Scenario description	**1.** The Charging Station sends a transaction-related message to the CSMS. **2.** The message is not accepted and `MessageAttemptsTransactionEvent` not reached. **3.** The Charging Station waits the number of preceding transmissions of this same message times `MessageAttemptIntervalTransactionEvent` seconds. **4.** The Charging Station resends the transaction-related message to the CSMS.
5	**Prerequisite(s)**	n/a
6	**Postcondition(s)**	**Successful postcondition:** MessageAttemptsTransactionEvent is *not* reached AND the transaction-related message is accepted. MessageAttemptsTransactionEvent is reached AND the transaction-related message is disposed. **Failure postcondition:** MessageAttemptsTransactionEvent is *not* reached AND the transaction-related message is disposed. MessageAttemptsTransactionEvent is reached AND the transaction-related message is accepted.

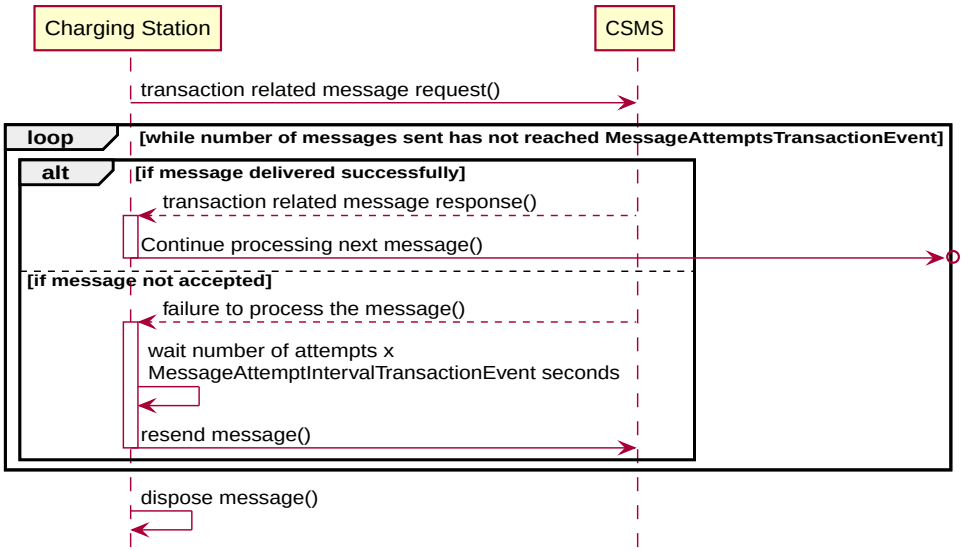

Figure 62. Sequence Diagram: Transaction-related message not accepted by CSMS

7	**Error handling**	n/a
8	**Remark(s)**	This use case describes the expect behaviour when the CSMS does not accept a message, or does not reply within the message timeout, this is different from a situation where the communication between Charging Station and CSMS is *Offline*.

E13 - Transaction-related message not accepted by CSMS - Requirements

Table 121. E13 - Requirements

ID	Precondition	Requirement definition
E13.FR.01		The number of times and the interval with which the Charging Station should retry such failed transaction-related messages MAY be configured using the `MessageAttemptsTransactionEvent` and `MessageAttemptIntervalTransactionEvent` Configuration Variables.
E13.FR.02	When the Charging Station encounters a first failure to deliver a certain transaction-related message.	The Charging Station SHALL send this message again as long as it keeps resulting in a failure to process this message and it has not yet encountered as many failures to process this message for this message as specified in its `MessageAttemptsTransactionEvent` Configuration Variable.
E13.FR.03	The CSMS does not accept a transaction-related message.	The Charging Station SHALL wait as many seconds as specified in its `MessageAttemptIntervalTransactionEvent` key, multiplied by the number of preceding transmissions of this same message.
E13.FR.04	If the final attempt fails.	The Charging Station SHALL discard the message and continue with the next transaction-related message, if there is any.

E13 - Transaction-related message not accepted by CSMS - Example

As an example, consider a Charging Station that has the value "3" for the `MessageAttemptsTransactionEvent` Configuration Variable and the value "60" for the `MessageAttemptIntervalTransactionEvent` Configuration Variable. It sends a TransactionEventRequest message and detects a failure to process the message in the CSMS. The Charging Station SHALL wait for 60 seconds, and resend the message. In the case when there is a second failure, the Charging Station SHALL wait for 120 seconds, before resending the message. If this final attempt fails, the Charging Station SHALL discard the message and continue with the next transaction-related message, if there is any.

E14 - Check transaction status

No.	Type	Description
1	**Name**	Check transaction status
2	**ID**	E14
	Functional block	E. Transactions
3	**Objectives**	To enable the CSMS to request the status of a transaction and to find out whether there are queued transaction-related messages.
4	**Description**	There are scenarios where a CSMS needs to know whether there are still messages for a transaction that need to be delivered. For example: A CSMS receives a TransactionEventRequest (eventType = Ended), it wants to start the billing process for this transaction but detects it is still missing some intermediate messages (it can check this via the sequence number in the messages). It can ask if the Charging Station has still messages in the queue for this transaction with the GetTransactionStatusRequest specifying the transactionId. Depending on the result the CSMS might for example: wait for the messages to be delivered, or start the billing process without the information. It may also need to know whether a transaction is still ongoing. If the CSMS wants to know if there are transaction-related messages in the queue at all (not just for a specific transaction), it can send a GetTransactionStatusRequest without a transactionId.
	Actors	CSMS, Charging Station
	Scenario description	**1.** The CSMS sends a GetTransactionStatusRequest with or without a transactionId to the Charging Station. **2.** The Charging Station responds with a GetTransactionStatusResponse.
5	**Prerequisites**	The CSMS knows the transactionId of a transaction it wants to know the status of.
6	**Postcondition(s)**	**Successful postcondition:** The CSMS knows the status of the requested transaction. **Failure postcondition:** The CSMS does not know the status of the requested transaction.

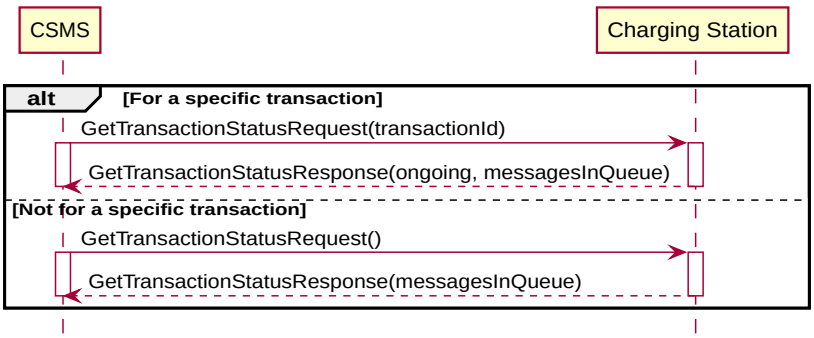

Figure 63. Sequence Diagram: Check transaction status

7	**Error Handling**	n/a
8	**Remarks**	When the CSMS receives a GetTransactionStatusResponse with both fields (*ongoing* and *messagesInQueue*) set to false, this might mean that the transaction is finished and there are no more messages in the queue for this transaction, or the Charging Station doesn't know anything about this transaction (anymore).

E14 - Check transaction status - Requirements

ID	Precondition	Requirements
E14.FR.01	The Charging Station receives a GetTransactionStatusRequest with a **transactionId** AND It did not do a transaction with that **transactionId**	The Charging Station SHALL respond with *ongoing* = false AND *messagesInQueue* = false.
E14.FR.02	The Charging Station receives a GetTransactionStatusRequest with a **transactionId** AND The transaction with that **transactionId** has not stopped yet	The Charging Station's response SHALL have *ongoing* = true.

ID	Precondition	Requirements
E14.FR.03	The Charging Station receives a GetTransactionStatusRequest with a **transactionId** AND The transaction with that **transactionId** has stopped	The Charging Station's response SHALL have *ongoing* = false.
E14.FR.04	The Charging Station receives a GetTransactionStatusRequest with a **transactionId** AND It has transaction-related messages to be delivered about the transaction with that **transactionId**	The Charging Station's response SHALL have *messagesInQueue* = true.
E14.FR.05	The Charging Station receives a GetTransactionStatusRequest with a **transactionId** AND It has no transaction-related messages to be delivered about the transaction with that **transactionId**	The Charging Station's response SHALL have *messagesInQueue* = false.
E14.FR.06	The Charging Station receives a GetTransactionStatusRequest without a **transactionId**	The Charging Station's response SHALL NOT have *ongoing* set.
E14.FR.07	The Charging Station receives a GetTransactionStatusRequest without a **transactionId** AND It has transaction-related messages to be delivered	The Charging Station's response SHALL have *messagesInQueue* = true.
E14.FR.08	The Charging Station receives a GetTransactionStatusRequest without a **transactionId** AND It has no transaction-related messages to be delivered	The Charging Station's response SHALL have *messagesInQueue* = false.

2.2. Interrupting and Stopping ISO 15118 Charging

E15 - End of charging process

Table 122. E15 - End of charging process

No.	Type	Description
1	**Name**	End of charging process.
2	**ID**	E15
	Functional block	E. Transactions
	Reference	ISO15118-1 H1 - End of charging process
3	**Objectives**	See ISO15118-1, use case Objective H1, page 44.
4	**Description**	See ISO15118-1, use case Description H1, page 44.
5	**Actors**	EV, EVSE, EV Driver
6	**Scenario Description**	See ISO15118-1, use case Description H1, Basic elementary use case description, first 5 bullets and last 2 remarks, page 44. **6.** The EV driver unplugs the cable from the EV **7.** The Charging Station sends a TransactionEventRequest with eventType eventType = Ended to the CSMS.
7	**Prerequisites**	See ISO15118-1, use case Prerequisites H1, page 44.
8	**Postcondition(s)**	The CSMS has received all relevant information about the transaction. See ISO15118-1, use case End Conditions H1, page 44.

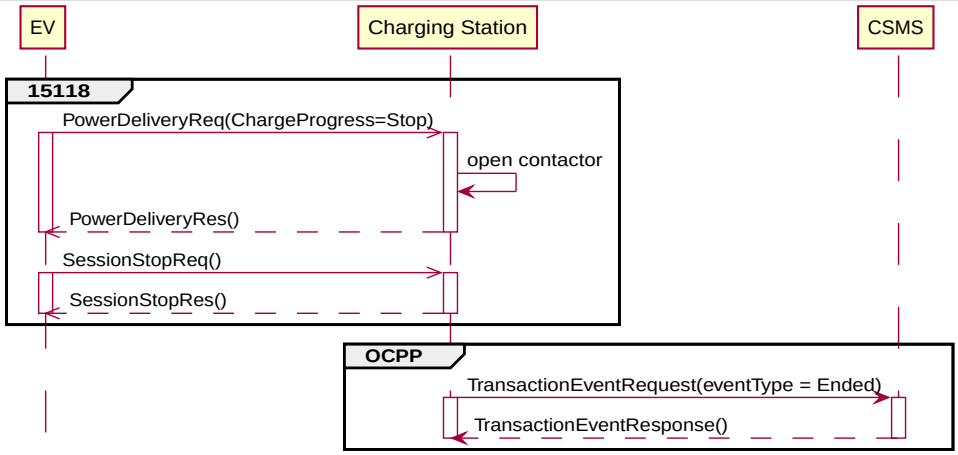

Figure 64. End of charging process

9	**Error handling**	n/a
10	**Remark(s)**	See ISO15118-1, use case Requirements H1, page 44 for the trigger. The scenario description and sequence diagram above are based on the Configuration Variable for stop transaction being configured as follows. `TxStopPoint`: ParkingBayOccupancy, EVConnected, Authorized, DataSigned, PowerPathClosed This use-case is also valid for other configurations, but then the transaction might stop at another moment, which might change the sequence in which message are sent. For more details see the use case: E06 - Stop Transaction options

Source: ISO15118-1

E15 - End of charging process - Requirements

Table 123. E15 - Requirements

ID	Precondition	Requirement definition
E15.FR.01	When configured to send meter data in the TransactionEventRequest (eventType = Ended), See: Meter Values - Configuration	The Charging Station SHALL add the configured measurands to the optional meterValue field in the TransactionEventRequest (eventType = Ended) sent to the CSMS to provide more details about transaction usage.
E15.FR.02	E15.FR.01 AND The Charging Station is running low on memory	The Charging Station MAY drop meter data in the TransactionEventRequest(eventType = Ended) message.
E15.FR.03	E15.FR.02	When dropping meter data, the Charging Station SHALL drop intermediate values first (1st value, 3th value, 5th etc), not start dropping values from the start of the list or stop adding values to the list.
E15.FR.04		After receiving a SessionStopReq message from the EV, the CS SHALL send a TransactionEventRequest message with eventType = Ended to inform the CSMS that the charging transaction has been stopped (by the EV).