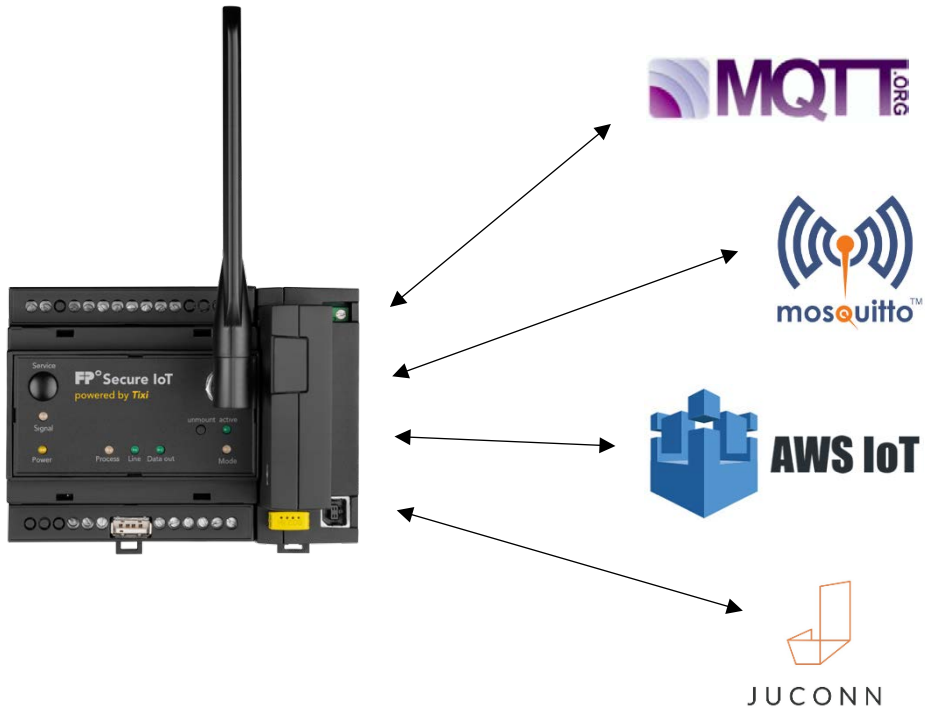
Universeller MQTT Client für FP Gateways Handbuch



Version: 1.2.6

www.inovolabs.com

Redaktionsschluss: 23.02.2021

Inhaltsverzeichnis

1 Einführung

Ab Firmware-Version 5.2.6.14 kann in FP Gateways ein universeller MQTT Client konfiguriert werden. Der MQTT Client kann mit jedem MQTT-Broker verbunden werden, der den MQTT Standard „MQTT 3.1.1“ unterstützt.

1.1 Verbindungskontrolle

Im Client können bis zu 3 MQTT Broker konfiguriert werden. Der Client versucht nach dem Start zunächst eine Verbindung zum ersten konfigurierten Broker aufzubauen. Sobald dieser Broker ausfällt, versucht der Client, eine Verbindung mit dem zweiten Broker aufzubauen usw. Wenn der dritte Broker ebenfalls ausfällt, wird wieder mit dem ersten Broker begonnen.

1.2 Verschlüsselt vs. unverschlüsselt

Die Verbindung zum Broker kann unverschlüsselt (Port 1883) oder verschlüsselt (Port 8883) aufgebaut werden. Eine verschlüsselte Verbindung kann sowohl zu einem öffentlichen Broker mit öffentlichen Zertifikaten aufgebaut werden als auch zu einem Broker, der mit privaten Zertifikaten arbeitet. Im letzteren Fall können die benötigten Zertifikate im Gateway abgelegt werden (siehe Kapitel 2).

1.3 Sendeintervall

Sobald eine MQTT-Verbindung aufgebaut wurde, sendet der Client regelmäßig Prozessdaten zum Broker. Es können bis zu 5 unabhängige Timer definiert werden, die zyklisch oder über eine Hysteresefunktion unterschiedliche Daten zum Broker senden.
Das Sendeintervall kann in einem weiten Bereich definiert werden (Default: 3 Sekunden).
Zusätzlich gibt es einen Rückkanal, über den der Broker dem Client TiXML-Befehle senden kann.
Die Ergebnisse der Befehle werden dann über einen weiteren Topic an den Broker zurückgesendet.

1.4 MQTT-Topics

Der Topic-Pfad zum Senden von Daten an den Broker (pubish topic) setzt sich aus einem festen Teil (**`/asset/telemetry/`**) und einem variablen Teil (**`asset_id`**) zusammen:
`/asset/telemetry/asset_id`

Beispiel: In der Konfiguration ist die asset-id **`Liegenschaft1`** konfiguriert.
Daraus ergibt sich der pubish topic: **`/asset/telemetry/Liegenschaft1`**
an die der Client die Realtime-Daten sendet.

Das Datenformat wird in den folgenden Kapiteln erläutert.

Übersicht der verwendeten Topics

Topic	Richtung	Bedeutung
/asset/telemetry/asset_id	publish	Senden von Daten an den Broker asset_id kann konfiguriert werden
/asset/commands/asset_id	subscribe	Broker kann über diesen Topic Kommandos an den Client senden
/asset/commandresp/asset_id	Publish	Client sendet das Ergebnis einer Kommandoausführung an den Broker zurück

Tabelle 1-1: Übersicht der verwendeten Topics

2 TiXML-Konfiguration des MQTT Clients

Der MQTT-Client wird über eine TiXML-Datenbank konfiguriert.

```
[<SetConfig _="ISP" ver="y">

   <CloudConn>

     <!-- connection without tls/ssl -->
     <CloudBaseUrl _="tcp://URL_or_IP_of_MQTT_broker:1883"/>

     <!-- optional: up to two additional URLs definable -->
     <CloudBaseUrl2 _="tcp://2nd_URL_or_IP_of_MQTT_broker:1883"/>
     <CloudBaseUrl3 _="tcp://3rd_URL_or_IP_of_MQTT_broker:1883"/>

     <username _="user" />
     <password _="password" />

     <!-- asset ID, e.g. ID of the device; used in subscribe topics,
          e.g. "asset/telemetry/asset_id" -->
     <asset_id _="Liegenschaft1" />

     <!-- optional: QoS (quality of service setting)
         valid values: 0 = send once (not guaranted); Default
                       1 = send at least once (guaranted)
                       2 = send only once (guaranted)
         Please note that not all brokers supports QoS 1 and 2
         (e.g. AWS:  supports QoS=1 only)    -->
     <QoS _="1" />

     <!-- optional: client authentication setting.
```

TLS_Client_authmode	Description
file	keys and certificates will be stored in internal flash memory. Path: /flash_user/app/VPN files to be used are defined with TLS_Client_key and TLS_Client_cert
Hsm	(Default) keys and certificates will be stored in hardware security module

```
     -->
     <TLS_Client_authmode _="file" />

     <!-- optional: key and cert files -->
     <!-- The following parameters will only be used if
          TLS_Client_authmode was set to "file" -->
     <TLS_Client_key _="mykey.key" />
     <TLS_Client_cert _="myclientcert.crt" />
     <!-- optional: server certificate.
```

TLS_Server_authmode	Description
cabundle none	no server authentication (Default)
cabundle	(Default) internal Mozilla root certs bundle file
file	server certificate will be stored in internal flash memory. Path: /flash_user/app/VPN TLS_Server_cert (see below)
hsm	server certificate will be stored in hardware security module

```
     -->
```

```
<TLS_Server_authmode _="file" />

<!-- optional: key and cert files -->
<!-- The following parameters will only be used if
     TLS_Client_authmode was set to "file" -->
<TLS_Server_cert _="servercert.crt" />


<!-- MQTT Keep Alive interval in seconds (default = 300) -->
<MQTTKeepAlive _="200"/>

<!-- if set to 1 the MQTT connection will be started at device
     startup; otherwise MQTT connection will not be established -->
<CloudConnStart _="1"/>

<!-- fixed setting (mandatory) -->
<CloudConnType _="TixiSimpleMQTT_1"/>

<!-- RTSendTimeX and UpdateRate1 controls the send interval
     X = 1 .. 5
     RTSendTimeX ranges:
      RTSendTimeX = -1: do not send data using a time
                        or activate hysteresis (RTSendTime1 only)
      RTSendTimeX = 1 .. 2^30: send interval in seconds
      RTSendTimeX = empty: send interval = 3 seconds (default)

     UpdateRate1 ranges (only used when RTSendTime1 = -1)
      UpdateRate1 = 1 .. 3600 = update interval of data
                                points in seconds

     Hysteresis is defined within the RealTime1 section (below).
     In hysteresis mode the variables will only be sent to the
     broker if a pre-defined treshhold is reached
     After a connection to the broker has been established all
     variables of a hysteris RealTime1-section will be sent once.

     hysteresis ranges: floating point number, max. two digits
     after the decimal point. Examples: 0.1 / 2 / 2.68
-->

<!-- specifies the first send timer, operating mode = hysteresis -->
<RTSendTime1 _="-1" />

<!-- update rate for variables in hysteresis mode in seconds -->
<UpdateRate1 _="20" />

<!-- list of variables for first send timer to be sent to broker
     The gateway reads out the current values from the Process
     tree using the UpdateRateX. If a data point has changed and
     the changes are bigger than the specified hysteresis the data
     point will be sent to the broker.
-->
<Realtime1>
  <Datapoint1 _="/Process/Bus1/Device1/Datapoint1" hysteresis="2"/>
  <Datapoint2 _="/Process/Bus1/Device1/Datapoint2" hysteresis="1"/>
  <Datapoint3 _="/Process/Bus1/Device2/Datapoint3" hysteresis="5"/>
  <Datapoint4 _="/Process/PV/ProcVar1" hysteresis="0.4"/>
  <Datapoint5 _="/Process/PV/ProcVar2" hysteresis="1.7"/>
</Realtime1>
```

```
    <!-- specifies the second send timer, send interval = 10 seconds -->
    <RTSendTime2 _="10" />
    <Realtime2>
       <Obis_180 _="/Process/Meter/MT174/Obis_180"/>
    </Realtime2>

    <!-- specifies the third send timer, no send interval -->
    <!-- data can be sent event driven using CloudSendRealtimeData -->
    <RTSendTime3 _="-1" />
    <Realtime3>
       <Obis_181 _="/Process/Meter/MT174/Obis_181"/>
    </Realtime3>

    <!-- specifies the forth send timer, send interval = 60 seconds -->
    <RTSendTime4 _="60" />
    <Realtime4>
       <Obis_182 _="/Process/Meter/MT174/Obis_182"/>
    </Realtime4>

    <!-- specifies the fifth send timer, send interval = 100 seconds -->
    <RTSendTime5 _="100" />
    <Realtime5>
       <Temp_Board _="/Process/OneWire/Mainboard/TEMP_Board"/>
    </Realtime5>
    .
    .
    <!-- specifies the fifth send timer, send interval = 100 seconds -->
    <RTSendTime5 _="100" />
    <Realtime5>
       <Temp_Board _="/Process/OneWire/Mainboard/TEMP_Board"/>
    </Realtime5>

   </CloudConn>

</SetConfig>]
```

Bitte beachten Sie folgende Hinweise:

- Die fett gedruckten Konfigurationsoptionen müssen konfiguriert werden (mandatory).
- Es muss mindestens eine CloudBaseUrl konfiguriert werden.
- Der Pfad für unverschlüsselte Urls lautet: **`tcp`**`://URL-or_IP_of_MQTT_broker:`**`1883`**
- Der Pfad für verschlüsselte Urls lautet: **`ssl`**`://URL-or_IP_of_MQTT_broker:`**`8883`**
- Es können bis zu 3 CloudBaseUrls konfiguriert werden.
- Die asset_id muss konfiguriert werden. Maximale Länge des Textes: 50 Zeichen.
  Es werden auch Referenzen unterstützt.
  Beispiel: Geräteseriennummer als asset_id:
  `<asset_id _="&#xae;/SerialNo" />`
- CloudConnStart und CloudConnType müssen wie oben beschrieben konfiguriert werden.
- Die Daten im Zweig Realtime1 .. RealTime5 können vom Anwender frei konfiguriert werden.

Nach dem Einspielen der MQTT-Client-Konfiguration sollte das FP Gateway neu gestartet werden.
Den Verbindungstatus des MQTT-Client kann man im Process-Zweig überwachen:
`[<Get _="/Process/CloudConn/" ver="y" />]`

Ergebnis (Beispiel):

```
<Get>
  <CloudConn>
    <ConnectionType _="mqtt+ssl:172.18.121.247:8883"/>
    <ConnectionState _="1"/>
    <ConnectionStateMsg _="connected"/>
    <LastTimeStamp _="Zeitstempel"/>
    <ChangeToggle _="Togglebit"/>
  </CloudConn>
</Get>
```

`ConnectionType` zeigt die Art der Verbindung (unverschlüsselt = mqtt/verschlüsselt = mqtt+ssl), die URL / IP-Adresse des gerade aktiven Brokers und den MQTT-Port an (z.B. 1883 oder 8883).
`ConnectionState` = 0 bedeutet: keine Verbindung
`ConnectionState` = 1 Verbindung ist aufgebaut
`ConnectionStateMsg` = (registering, not connected, connecting, connected)
`LastTimeStamp` = Der Zeitstempel zeigt an, wann die letzte MQTT Message gesendet wurde.
Format: DD.MM.YYYY HH:MM:SS Timezone
`ChangeToggle` = (wechselt jedesmal zwischen 0 und 1, sobald sich der Zeitstempel geändert hat)

2.1 Realtime-Sektionen

In bis zu 5 Realtime Sektionen kann festgelegt werden, welche Datenpunkte zu übertragen sind.

Jeder Realtime Sektion muss ihr Zyklus vorangestellt werden. Über den Zykluswert wird auch festgelegt, ob die Daten zyklisch oder über eine Hysteresefunktion übertragen werden.

Z.B. `RTSendTime1, …, RTSendTime5.`
Dazu gehören dann die Daten aus den entsprechenden Sektionen <RealTime1>..</RealTime1> usw.

Beispiel:

```
<CloudConn>
    <RTSendTime1 _="-1" />
    <UpdateRate1 _=20 />
    <Realtime1>
       <Volt_L1_N _="/Process/Modbus/D1/Volt_L1_N" hysteresis="2" />
       .
       .
    </Realtime1>

    <RTSendTime2 _="10" />
    <Realtime2>
       <Obis_180 _="/Process/Meter/MT174/Obis_180"/>
    </Realtime2>
    .
    .
    <RTSendTime5 _="100" />
    <Realtime5>
       <Temp_Board _="/Process/OneWire/Mainboard/TEMP_Board"/>
    </Realtime5>
</CloudConn>
```

2.2 Senden von Daten über EventHandler oder Kommando

Datenpunkte können per EventHandler oder auch per Kommando verschickt werden:
Kommando (TICO, …):

```
<CloudSendRealtimeData [_="CloudConn[,Realtime_Specifier"]] />
```

bzw.

```
[<CloudSendRealtimeData />]
```

CloudConn (optional) = Name des CloudConnectors. Werte: CloudConn, CloudConn2, CloudConn3. Default: CloudConn

Realtime_Specifier (optional) = Name der Realtime-Sektion, z.B. Realtime1. Default=Realtime1
Wird der optionale Parameter weggelassen, werden alle definierten Sektionen versendet.

Beispiel 1: Realtime1-Sektionen von CloudConn versenden
```
<CloudSendRealtimeData />
```

Beispiel 2: Realtime-Sektion "Realtime2" von CloudConn2 versenden
```
<CloudSendRealtimeData _="CloudConn2,Realtime2" />
```

Beispiel 3: Realtime-Sektion "Realtime3" von CloudConn versenden
```
<CloudSendRealtimeData _="CloudConn,Realtime3" />
```

Beispiel 4: Realtime-Sektion "Realtime2" von CloudConn2 über einen EventHandler versenden
```
<CloudSendRealTimeDate_2_2>
      <CloudSendRealtimeData _="CloudConn2,Realtime2" />
    </CloudSendRealTimeDate_2_2>
```

Minütliche Ausführung des EventHandlers per Scheduler:
```
<CloudSendRealTimeDate_2_2_SCH _="CloudSendRealTimeDate_2_2">
      <Minute _="0-59" />
</CloudSendRealTimeDate_2_2_SCH>
```

Test per TiXML-Kommando mit TICO
```
[<DoOn _="CloudSendRealTimeDate_2_2" ver="v" />]
```

2.3 Hysterese

Für den Timer 1 (`RTsendTime1`) ist optional für jeden Datenpunkt ist einstellbar, wann dieser abhängig von einer Hystereseeinstellung übertragen werden soll.

Beispiel: Datenpunkt „Temperatur“
Startwert = 20,5K
Hysterese absolut = 0,5

Ergebnis: Wenn der Wert des Datenpunktes auf 20 oder 21 wechselt, wird dieser übertragen.

Festlegungen:
- Nach dem Neustart des Gerätes werden alle Werte einmalig übertragen.
- Wenn sich der zuletzt übertragene Wert um den Hysterese-Wert ändert, wird der Wert wieder einmalig übertragen.

Die benötigten Parameter:

`<RTsendTime1 _="-1" />`
Der Wert `-1` bedeutet, dass Daten nicht mehr zyklisch gesendet werden, sondern anhand der Hysterese-Parametrierung.

`<UpdateRate1 _="Pollzeit" />`
Pollzeit spezifiziert, wie häufig die Werte aus dem Process-Zweig abgefragt werden sollen. Die Pollzeit wird in Sekunden angegeben. Wertebereich: 1 .. 3600

Achtung:
Die Hysteresefunktion ist derzeit nur für den ersten Timer verfügbar (`RTsendTime1` und `UpdateRate1`). Die Hysterese ist nur für numerische Werte nutzbar.

Für jeden Realtime-Wert wird in der Konfiguration die Hysterese als Fliesskommazahl festgelegt.
Beispiel:

```
hysteresis="1.5"
```

Definition:

- Komma / Dezimalpunkt wird mit "." dargestellt
- Max. 2 Nachkommastellen
- keine negativen Zahlen

Beispiel:

```
<RTsendTime1 _="-1" />
<UpdateRate1 _="20" />

<Realtime1>
   <Voltage_L1_N _="/Process/Modbus/D1/Voltage_L1_N" hysteresis="2" />
   <CPULoad _="/Process/MB/CPULoad" hysteresis="8.5" />
   <Vorlauftemp _="/Process/PV/Vorlauftemp" hysteresis="0.5" />
   <Ruecklauftemp _="/Process/PV/Ruecklauftemp" hysteresis="0.8" />
</Realtime1>
```

Im oben gezeigten Beispiel wird die Hysterese der einzelnen Datenpunkte festgelegt.
Wenn sich der Datenpunkt `Voltage_L1_N` um +/- 2.0 zum zuvor übertragenen Wert ändert, wird dieser Wert übertragen. Wenn sich der Datenpunkt `CPULoad` um +/- 8.5 zum zuvor übertragenen Wert ändert, wird dieser Wert übertragen usw.

2.4 Optische Signalisierung der MQTT-Verbindung

Die „Signal“-LED signalisiert den Zustand der MQTT-Verbindung.
Aus = keine MQTT-Verbindung aktiv
Rot blinkend = MQTT-Verbindung wird aufgebaut
Grün leuchtend = MQTT-Verbindung ist aufgebaut

2.5 Service-Routing

Normalerweise wird die Verbindung zum Broker über das aktive LAN-Interface aufgebaut.
Über die TiXML-Datenbank `ISP/ISP/OUT` kann die MQTT-Verbindung auch über eine Mobilfunkverbindung oder einen VPN-Tunnel aufgebaut werden:

```
[<SetConfig _="ISP/ISP" ver="y">
    <!-- Kommunikationsinterface fuer Dienste definieren -->
    <OUT>
        <SMTP _="MODEM"/>
        <CBIS _="MODEM"/>
        <POP3 _="MODEM"/>
        <URLSend _="MODEM"/>
        <INetTime _="MODEM"/>
        <HTTPConn _="MODEM"/>
        <CloudConn _="Ethernet"/>
        <IBMConn _="MODEM"/>
        <FTPPut _="MODEM"/>
        <SFTPPut _="MODEM"/>
        <VPN _="MODEM"/>
    </OUT>
</SetConfig>]
```

Die folgenden Optionen sind verwendbar für den MQTT-Client:

```
Ethernet
MODEM
VPN
```

3 Datenformat der Prozessdaten

Der MQTT-Client sendet die Prozessdaten als MQTT-Payload in folgendem Format:

```
<Data>
    <TimeStamp _="03.04.2018 11:48:00 +0100" />
    <Datapoint1 _="1" flag="1"/>
    <Datapoint2 _="20" flag="1"/>
    <Datapoint3 _="75.7" flag="1"/>
    <Datapoint4 _="13" flag="2"/>
    <Datapoint5 _="189" flag="2"/>
</Data>
```

Der Zeitstempel wird aus der aktuellen Systemzeit und der in der USER-Datenbank konfigurierten Zeitzone gebildet (`/USER/USER/TimeZone`).

Das `flag` gibt Auskunft über den Variablentyp.
`flag` = 0 oder 1: zeigt den DeviceState einer **Busvariablen** an. (0 = Daten ungültig, 1 = Daten gültig)
`flag` = 2: es handelt sich nicht um einer **Busvariable**, sondern z.B. um eine Process-Variable oder eine Variable aus einem Erweiterungsmodul (z.B. „`/Process/C621/Q0`“).

Je nach Konfiguration der Timer `RTSendTime1` .. `RTSendTime5` werden die Prozessdaten an den Broker gesendet.

4 Rückkanal zum FP-Gateway

Über einen Rückkanal kann der MQTT-Broker TiXML-Befehle an das FP-Gateway senden, um so zum Beispiel eine neue Konfiguration in das Gerät zu speichern, das Gerät neu zu starten oder einen Ausgang zu schalten. Der MQTT-Broker kann sämtliche TiXML-Befehle an das Gerät schicken, wie es auch mit der Konfigurationssoftware TICO möglich ist. Der Client beantwortet die Befehle über einen speziellen publish-Topic .

4.1 Senden eines Kommandos

Der MQTT-Client abonniert (subscribe) den Topic `/asset/commands/asset_id`

Der Broker kann nun beliebige TiXML-Befehle an den MQTT-Client schicken:

```
<Command>
    <!-- The RequestID must be generated by the broker
         It can be any string containing letters and numbers.
         Max. Length: 30 characters
         The RequestID will be sent back by the FP gateway
         together with the response to the TiXML command. -->

    <RequestID _="ID" />
    <!-- Put your command within the TiXML tag.
         It can be any TiXML command (without the command frame [])
         including Set, SetConfig, GetConfig, etc.
         The FP gateway will execute the command and sends
         back the response using the topic asset/commandresp/asset_id -->

     <TiXML>
      <!-- Here comes the TiXML command...
           Example: SetConfig of AccRights Database
      -->
        <SetConfig _="USER" ver="y">
            <AccRights>
               <Groups>
                  <Admin>
                    <LocalLogin AccLevel="1"/>
                    <CardLogin AccLevel="1"/>
                    <RemoteLogin AccLevel="1"/>
                    <EthernetLogin AccLevel="1"/>
                  </Admin>

                  <Webserver>
                    <WebServer AccLevel="20"/>
                  </Webserver>
               </Groups>

               <User _="Plain">
                    <Def_LocalLogin Plain="" Group="Admin"/>
                    <Def_CardLogin Plain="" Group="Admin"/>
                    <Def_RemoteLogin Plain="" Group="Admin"/>
                    <Def_EthernetLogin Plain="" Group="Admin"/>
                    <ADMIN Plain="" Group="Admin"/>
               </User>
            </AccRights>
         </SetConfig>

      </TiXML>
</Command>
```

4.2 Ergebnis eines Kommandos empfangen

Der Client beantwortet die Befehle über den publish-Topic `/asset/commandresp/asset_id`
Dazu sendet der Client die Antwort in folgendem Format an den Broker:

```
<CommandResp>
    <!-- The FP gateway sends back the command response together with
         the RequestID from the command sent by the broker.
         It can be any string containing letters and numbers.
         Max. Length: 30 characters
         The RequestID will be sent back by the FP gateway
         together with the response to the TiXML command. -->

    <RequestID _="ID" />

    <!-- Here comes the command response from the FP gateway.
         The response is exactly what you see in TICO, except the
         command frame ([ ] will be omitted).
    -->

     <TiXML>
      <!-- Here comes the TiXML command response ...
           Example: SetConfig of AccRights Database was sent
           Response is an empty SetConfig command or an error description
           in case the command execution failed.
      -->
         <SetConfig/>

      </TiXML>
</CommandResp>
```

4.3 Fehlermeldungen des Gateways

Wenn das Gateway Kommandos vom Broker empfängt und diese Kommandos ausführt, werden unter Umständen Fehlermeldungen generiert, wenn das Kommando z.B. falsche Parameter enthält oder aus anderen Gründen nicht ausführbar ist.

Über einen Parameter beim Senden des Kommandos kann dabei spezifiziert werden, wie ausführlich die Fehlermeldungen vom Gateway beantwortet werden.

Es gibt drei mögliche Formate, mit denen diese Fehler mitgeteilt werden. Abhängig von der Vorgabe durch den Parameter “ver“, erfolgt eine kurze oder lange Fehlermeldung. Siehe Tabelle 4-1.

Beispielkommando:

```
[<Get _="/Processs/" ver="vmode"/>]
```

vmode	Beschreibung	Beispiel
n	Fehlermeldung als numerischer Wert	`<Error _="-2196" />`

y	Fehlermeldung, kurze Beschreibung	`<Error>`
		`  <ErrNo _="-2196" />`
		`  <ErrText _="path to key not found" />`
		`  <ErrorCause>`
		`  <ErrNo _="-2196" />`
		`  <ErrText _="path to key not found" />`
		`  <Class _="TXSTCPGetSetValueCmd" />`
		`  </ErrorCause>`
		`</Error>`

v	Fehlermeldung, lange Beschreibung	<Error> <ErrNo _="-2196" /> <ErrText _="path to key not found" /> <ErrorCause> <ErrNo _="-2196" /> <ErrText _="path to key not found" /> <Line _="127" /> <Module _="SSet.cpp" /> <Class _="TXSTCPGetSetValueCmd" /> </ErrorCause> </Error>

Tabelle 4-1: Fehlerrückgabewerte in Abhängigkeit des Parameters "ver"

Wichtig:
Vor jedem Konfigurationsbefehl (also alle `SetConfig`-Befehle) sollte immer die Prozeßbearbeitung mit folgendem Befehl gestoppt werden:

```
<Command>
    <RequestID _="ID" />
    <TiXML>
       <Set _="/Process/Program/Mode" value="Stop" ver="v"/>
    </TiXML>
</Command>
```

Das Ergebnis des oben genannten Befehls sollte abgewartet werden. Nur wenn der o.g. Stop-Befehl erfolgreich ausgeführt wurde, sollte anschließend ein `SetConfig`-Befehl geschickt werden.
Wenn innerhalb von max. 15 Sekunden keine Antwort auf den Stop-Befehl empfangen wird, muss der Stop-Befehl erneut gesendet werden.
Nachdem die Konfigurationsdatenbank(en) an das Gerät geschickt wurden, kann dann die Prozeßbearbeitung mit folgendem Befehl wieder gestartet werden:

```
<Command>
    <RequestID _="ID" />
    <TiXML>
       <Set _="/Process/Program/Mode" value="Run" ver="v"/>
    </TiXML>
</Command>
```