HomeMatic XML-RPC-Schnittstelle

Spezifikation

V2.9

Inhaltsverzeichnis

1 Allgemeines

Dieses Dokument beschreibt eine universelle Programmierschnittstelle zu einem Schnittstellenprozess, über den sich Aktoren und Sensoren in der Hausautomation ansprechen lassen. Es wird davon ausgegangen, dass es einen logikverarbeitenden Prozeß („Logikschicht“) gibt, der sich beliebig vieler Schnittstellenprozesse bedient, um die über unterschiedliche Übertragungssysteme angebundenen Geräte anzusprechen.

Es soll möglich sein, dass der Logikschicht beliebig viele Schnittstellenprozesse bekannt sind. Jeder Schnittstellenprozess wird dazu über eine eindeutige String-Id identifiziert.

Für die Kommunikation zwischen Logikschicht und den Schnittstellenprozessen wird XmlRpc (http://www.xmlrpc.org) verwendet.

Der Schnittstellenprozess kennt nur logische Geräte. Jeder Kanal eines Mehrfachgerätes sowie das Mehrfachgerät selbst werden als logische Geräte mit jeweils eigener Adresse betrachtet. Jedem logischen Gerät sind mehrere von folgenden möglichen Parameter-Sets zugeordnet:

- MASTER
  Dieses Parameter-Set enthält die für das logische Gerät möglichen Konfigurationseinstellungen
- VALUES
  Dieses Parameter-Set enthält die für das logische Gerät gültigen dynamischen Werte (Schaltzustand, Dimmerhelligkeit, Tastendruck-Ereignisse)
- LINK
  Von diesen Parameter-Sets sind meist mehrere vorhanden. Jedes beschreibt die Kommunikationsbeziehung zu einem anderen Gerät am selben Bus.

Beispiel:

Für ein Multifunktionsgerät (Adresse ABC1234567) mit zwei Schaltausgängen und zwei Tastereingängen werden folgende logische Geräte erzeugt:

- „ABC1234567“ für das Gerät. Parameter-Set „MASTER“ ist vorhanden.
- „ABC1234567:0“ für den ersten Schaltausgang. Parameter-Sets „MASTER“ und „VALUES“ sind vorhanden.
- „ABC1234567:1“ für den zweiten Schaltausgang. Parameter-Sets „MASTER“ und „VALUES“ sind vorhanden.
- „ABC1234567:2“ für den ersten Tastereingang. Parameter-Sets „MASTER“, „VALUES“ und „LINK“ sind vorhanden. „LINK“-Parametersets beschreiben jeweils einen durch diesen Tastereingang geschalteten Aktor.
- „ABC1234567:3“ für den zweiten Tastereingang. Parameter-Sets „MASTER“, „VALUES“ und „LINK“ sind vorhanden. „LINK“-Parametersets beschreiben jeweils einen durch diesen Tastereingang geschalteten Aktor.

2 Zugriff auf die Schnittstelle

2.1 HomeMatic Zentrale

Die HomeMatic Zentrale verfügt derzeit über drei Schnittstellen, über die Geräte angeschlossen sein können:

- BidCos-RF für Funk-Komponenten
  Port-Nummer: 2001
- BidCos-Wired für drahtgebundene Geräte (ist nur in Verbindung mit einem HomeMatic Wired LAN Gateway verfügbar)
  Port-Nummer: 2000

Für jede dieser Schnittstellen stellt die HomeMatic Zentrale einen XML-RPC-Server zur Verfügung. Die entsprechende URL lautet jeweils:

http://<IP_DER_ZENTRALE>:<PORT_NUMMER>/

Um Ereignisbenachrichtigungen zu erhalten, kann sich eine Logikschicht bei den Schnittstellenprozessen anmelden. Dies geschieht über die Methode init. In diesem Fall muss die Logikschicht selbst einen XML-RPC-Server und diverse Methoden bereitstellen (siehe Abschitt 4: Methoden der Logikschicht). Des Weiteren muss der von der Logikschicht bereitgestellte XML-RPC-Server die Standardmethode *system.multicall* unterstützen.

3 Datentypen

3.1 ParamsetDescription

ParamsetDescription ist ein Struct, das für jeden Parameter einen Member enthält. Dieser Eintrag ist wiederum ein Struct vom Type ParameterDescription.

3.2 ParameterDescription

ParameterDescription ist ein Struct, der einen einzelnen Parameter eines Parameter-Sets beschreibt. Folgende Member sind in jedem Fall vorhanden:

- TYPE
  Ist ein String, der den Datentypen des Paramters angibt.
- OPERATIONS
  Ist ein Integer, der eine Oder-Verknüpfung der mit diesem Parameter möglichen Operationen ist. Dabei sind folgende Werte möglich: 1=Read, 2=Write, 4=Event.
- FLAGS
  Ist ein Integer, der eine Oder-Verknüpfung von Flags für die UI-Darstellung ist. Folgende Werte haben eine Bedeutung:
  0x01 : Visible-Flag. Dieser Parameter sollte für den Endanwender sichtbar sein.
  0x02 : Internal-Flag. Dieser Parameter wird nur intern verwendet.
  0x04 : Transform-Flag. Änderungen dieses Parameters ändern das Verhalten des entsprechenden Kanals völlig. Darf nur geändert werden, wenn keine Verknüpfungen am entsprechenden Kanal vorhanden sind.
  0x08 : Service-Flag. Dieser Parameter soll als Servicemeldung behandelt werden. Als Datentyp für Servicemeldungen sind nur Boolean und Integer zulässig. Bei `0` bzw. `false` liegt dabei keine Meldung vor, ansonsten liegt eine Meldung vor.
  0x10 : Sticky-Flag. Nur bei Servicemeldungen. Servicemeldung setzt sich nicht selbst zurück, sondern muss von der Oberfläche zurückgesetzt werden.
- DEFAULT
  Gibt den default-Wert des Parameters an. Der Datentyp entspricht dem des entsprechenden Parameters.
- MAX
  Gibt den maximalen Wert des Parameters an. Der Datentyp entspricht dem des entsprechenden Parameters.
- MIN
  Gibt den minimalen Wert des Parameters an. Der Datentyp entspricht dem des entsprechenden Parameters.
- UNIT
  Ist ein String, der die Maßeinheit des Parameters angibt.
- TAB_ORDER
  Ist ein Integer, der die Reihenfolge der Parameter innerhalb eines Parametersets angibt.
- CONTROL
  Optional. Ist ein String, der für das Userinterface angibt, welches Control für die Darstellung dieses Wertes verwendet werden sollte. Der Aufbau ist „<Control-Name>.<Variablen-Name>:<Control-Index>“. Der Variablenname ist optional. Er wird

nur benötigt für Controls, die auf mehreren Werten operieren. Der Control-Index ist ebenfalls optional. Er wird nur benötigt, um mehrere gleichartige Controls im gleichen Gültigkeitsbereich (z.B. innerhalb eines Kanals) unterscheiden zu können.
Beispiel:
Ein Dimmer hat zwei Werte. Einer ist die Helligkeit vom Typ Float und ein weiterer ist vom Typ „ACTION“ und erlaubt es, den Dimmer auf alte Helligkeit einzuschalten. Die Oberfläche definiert ein Control namens „Dimmer“ um diese beiden Werte dem Benutzer darzustellen.
CONTROL ist dann belegt mit „Dimmer.Level“ für die Helligkeit sowie „Dimmer.OldValue“ für das Einschalten auf alten Wert.

Es können beliebig viele weitere Member vorhanden sein. Diese werden in der entsprechenden Gerätebeschreibung oder unten beim Datentyp definiert und liefern der Benutzeroberfläche weitere Informationen, um das Gerät sinnvoll abbilden zu können.

Folgende Werte von „TYPE“ werden unterstützt:

3.2.1 TYPE=FLOAT

Datentyp des entsprechenden Parameters ist Float.

3.2.1.1 Weitere Member in ParameterDescription

- SPECIAL [optional]
  Array diskreter Spezialwerte mit einer speziellen Bedeutung außerhalb des Wertebereichs. Die Elemente des Arrays sind vom Typ Struct. Jedes Element hat zwei Felder:
  `String ID` gibt den Bezeichner des Spezialwertes an
  `Float VALUE` gibt den Wert des Spezialwertes an

3.2.2 TYPE=INTEGER

Datentyp des entsprechenden Parameters ist Integer.

- SPECIAL [optional]
  Array diskreter Spezialwerte mit einer speziellen Bedeutung außerhalb des Wertebereichs. Die Elemente des Arrays sind vom Typ Struct. Jedes Element hat zwei Felder:
  `String ID` gibt den Bezeichner des Spezialwertes an
  `Integer VALUE` gibt den Wert des Spezialwertes an

3.2.3 TYPE=BOOL

Datentyp des entsprechenden Parameters ist Boolean. Es gibt keine weiteren datentypspezifischen Member in der ParameterDescription.

3.2.4 TYPE=ENUM

Datentyp des entsprechenden Parameters ist Integer. Der Wert des Integers gibt der Index einer der möglichen Optionen an.

3.2.4.1 Weitere Member in ParameterDescription

- VALUE_LIST
  ist ein Array<String>. Für jeden möglichen Wert des ENUM ist ein Element vorhanden, das den Namen des Wertes angibt. Jedem möglichen (Integer-)Wert des Parameters entspricht dabei eine Position innerhalb des Arrays. Der String an der entsprechenden

Position gibt den Symbolischen Namen des Wertes an. Steht an einen Position im Array der leere String, so ist der entsprechende Wert des Parameters nicht definiert und kann nicht angenommen werden.

3.2.5 TYPE=STRING

Datentyp des entsprechenden Parameters ist String. Es gibt keine weiteren datentypspezifischen Member in der ParameterDescription.

3.2.6 TYPE=ACTION

Datentyp des entsprechenden Parameters ist Boolean. Es wird beim Lesen immer FALSE zurückgegeben. Bei einem Event ist der Parameter jedoch immer TRUE. Beim Schreiben auf den Parameter spielt der geschriebene Wert keine Rolle.

Der Typ ACTION wird verwendet, um Vorgänge wie das Drücken einer Fernbedienungstaste abzubilden. In diesem Fall wird beim Drücken der Taste ein Event generiert. Umgekehrt wird beim Schreiben auf diesen Parameter ein Tastendruck simuliert.

3.3 Paramset

Paramset ist ein Struct, das für jeden Parameter einen Member vom entsprechenden Datentyp (siehe ParamsetDescription) enthält.

3.4 DeviceDescription

DeviceDescription ist ein Struct, das die folgenden Member enthält:

- TYPE
  Datentyp String. Typ des Gerätes.
- ADDRESS
  Datentyp String. Adresse des Kanals/Gerätes.
- RF_ADDRESS (Nur be Greäten)
  Datentyp Integer. Funk-Adresse des Gerätes.
- CHILDREN
  Datentyp Array<String>. Adressen der untergeordneten Kanäle.
- PARENT
  Datentyp String. Adresse des übergeordneten Gerätes. Ist bei Geräten vorhanden aber leer.
- PARENT_TYPE (Nur bei Kanälen)
  Datentyp String. Typ (Kurzbezeichnung) des übergeordneten Gerätes.
- INDEX (Nur bei Kanälen)
  Datentyp Integer. Gibt die Kanalnummer an.
- AES_ACTIVE
  Datentyp Boolean. Gibt an, ob die gesicherte Übertragung für den Kanal aktiviert ist.
- PARAMSETS
  Datentyp Array<String>. Liste der Namen der vorhandenen Parameter-Sets.
- FIRMWARE (Nur bei Geräten, Optional)
  Datentyp String. Firmwareversion des Gerätes.
- AVAILABLE_FIRMWARE (Nur bei Geräten, Optional)

Datentyp String. Für ein Firmwareupdate verfügbare Firmwareversion.

- VERSION
  Datentyp Integer. Version der Geräte- oder Kanalbeschreibung.
- FLAGS
  Datentyp Integer. Oder-Verknüpfung von Flags für die UI-Darstellung. Folgende Werte haben eine Bedeutung:
  0x01 : Visible-Flag. Dieses Objekt sollte für den Endanwender sichtbar sein.
  0x02 : Internal-Flag. Dieses Objekt wird nur intern verwendet und ist für den Endanwender nicht sichtbar.
  0x08 : Dontdelete-Flag. Dieser Objekt kann nicht gelöscht werden.
- LINK_SOURCE_ROLES (nur bei Kanälen)
  Datentyp String. Durch Leerzeichen getrennte Liste von Rollen, die der Kanal in einer Verknüpfung als Sender annehmen kann. Eine Rolle ist z.B. „SWITCH“ für einen Kanal, der Schaltbefehle senden kann.
- LINK_TARGET_ROLES (nur bei Kanälen)
  Datentyp String. Durch Leerzeichen getrennte Liste von Rollen, die der Kanal in einer Verknüpfung als Empfänger annehmen kann. Eine Rolle ist z.B. „SWITCH“ für einen Kanal, der auf empfangene Schaltbefehle reagieren kann.
- DIRECTION (nur bei Kanälen)
  Datentyp Integer. Gibt die Richtung (Senden oder Empfangen) dieses Kanals in einer direkten Verknüpfung an.
  0 = DIRECTION_NONE (Kanal unterstützt keine direkte Verknüpfung)
  1 = DIRECTION_SENDER
  2 = DIRECTION_RECEIVER
- GROUP (optional, nur bei Kanälen)
  Datentyp String. Nur bei gruppierten Kanälen (Tastenpaaren) vorhanden. Hier wird die Adresse des anderen zur Gruppe gehörenden Kanals angegeben.
- TEAM (optional, nur bei Kanälen mit Team)
  Datentyp String. Nur bei Kanälen mit Team (z.B. Rauchmelder) vorhanden. Gibt die Adresse des virtuellen Teamkanals an.
- TEAM_TAG (optional, nur bei Kanälen und Teams)
  Datentyp String. Nur bei Kanälen mit Team (z.B. Rauchmelder) und bei virtuellen Teamkanälen vorhanden. Für die Auswahl eines Teams an der Oberfläche. Ein Kanal, der einem Team zugeordnet werden soll und der virtuelle Teamkanal (das Team) müssen hier denselben Wert haben.
- TEAM_CHANNELS (optional, nur bei Kanälen, die ein Team darstellen)
  Datentyp Array<String>. Adressen der dem Team zugeordneten Kanäle.
- INTERFACE (optional, nur bei BidCos-RF)
  Datentyp String. Seriennummer des dem Gerät zugeordneten Interfaces.
- ROAMING (optional, nur bei BidCos-RF)
  Datentyp Boolean. Ist true, wenn die Interfacezuordnung des Geräts automatisch den Empfangsverhältnissen angepasst wird.
- RX_MODE (nur bei Geräten, nur bei BidCos-RF)
  Datentyp Integer. Oder-Verknüpfung von Flags die den Empfangsmodes des Gerätes

repräsentieren. Folgende Werte haben eine Bedeutung:
0x01 = RX_ALWAYS; Das Gerät ist dauerhaft auf Empfang
0x02 = RX_BURST; Das Gerät arbeitet im wake on radio Modus
0x04 = RX_CONFIG; Das Gerät kann nach drücken der Konfigurationstaste erreicht werden
0x08 = RX_WAKEUP; Das Gerät kann nach einer direkter Kommunikation mit der Zentrale geweckt werden
0x10 = RX_LAZY_CONFIG; Das Gerät unterstützt LazyConfig. Das Gerät kann nach einer normalen Bedienung (z.B. Tastendruck einer Fernbedienung) konfiguriert werden.

4 Methoden der Schnittstellenprozesse

In der folgenden Liste mit [optional] gekennzeichneten Methoden müssen von einem konkreten Schnittstellenprozess nicht unbedingt exportiert werden. Bei Verwendung dieser Methoden ist mit einem Fehler zu rechnen, wenn nich vor dem Aufruf mit `system.methodHelp` oder `system.listMethods` die Existenz geprüft wird.

4.1 Methodenübersicht

Methode	***BidCos-RF (Port: 2001)***	***BidCos-Wired (Port: 2000)***
activateLinkParamset	x	-
addDevice	x	-
addLink	x	x
changekey	x	-
clearConfigCache	x	x
deleteDevice	x	x
determineParameter	x	-
getDeviceDescription	x	x
getInstallMode	x	-
getKeyMismatchDevice	x	-
getLinkInfo	x	x
getLinkPeers	x	x
getLinks	x	x
getParamset	x	x
getParamsetDescription	x	x
getParamsetId	x	x
getValue	x	x
init	x	x
listDevices	x	x
listTeams	x	-
logLevel	x	x
putParamset	x	x
removeLink	x	x
reportValueUsage	x	x
restoreConfigToDevice	x	-
rssiInfo	x	-
searchDevices	-	x
setInstallMode	x	-
setLinkInfo	x	x
setTeam	x	-
setTempKey	x	-
setValue	x	x
system.listMethods	x	x
system.methodHelp	x	x
system.multicall	x	x
updateFirmware	x	x
listBidcosInterfaces	x	-

setBidcosInterface	x	-
getServiceMessages	x	-
getMetadata	x	-
setMetadata	x	-
getAllMetadata	x	-
abortDeleteDevice	x	-
hasVolatileMetadata	x	-
setVolatileMetadata	x	-
getVolatileMetadata	x	-
deleteVolatileMetadata	x	-
getVersion	x	-
setInterfaceClock	x	-
replaceDevice	x	x
listReplaceableDevices	x	x
ping	x	x
refreshDeployedDeviceFirmwareList	x	-

4.2 Allgemeine Methoden

4.2.1 init

```
void init(String url, String interface_id)
```

Mit dieser Methode teilt die Logikschicht dem Schnittstellenprozess mit, dass sie gerade gestartet wurde. Der Schnittstellenprozess wird sich daraufhin selbst initialisieren und z.B. mit listDevices() die der Logikschicht bekannten Geräte abfragen.

Der Parameter `url` gibt die Adresse des XmlRpc-Servers an, unter der die Logikschicht zu erreichen ist.

Der Parameter `interface_id` teilt dem Schnittstellenprozess die Id, mit unter der er sich gegenüber der Logikschicht identifiziert.

Zum Abmelden von der Ereignisbehandlung wird `interface_id` leer gelassen.

4.2.2 listDevices

```
Array<DeviceDescription> listDevices()
```

Diese Methode gibt alle dem Schnittstellenprozess bekannten Geräte in Form von Gerätebeschreibungen zurück.

4.2.3 getDeviceDescription

```
DeviceDescription getDeviceDescription(Sting address)
```

Diese Methode gibt die Gerätebeschreibung des als `address` übergebenen Gerätes zurück.

4.2.4 getParamsetDescription

```
ParamsetDescription getParamsetDescription(String address,
String paramset_type)
```

Mit dieser Methode wird die Beschreibung eines Parameter-Sets ermittelt. Der Parameter

address ist die Adresse eines logischen Gerätes (z.B. von listDevices zurückgegeben). Der Parameter paramset_type ist „MASTER“, „VALUES“ oder „LINK“.

4.2.5 getParamsetId

```
String getParamsetId(String address, String type)
```

Diese Methode gibt die Id eines Parametersets zurück. Diese wird verwendet, um spezialisierte Konfigurationsdialoge (Easymode) den Parametersets zuzuordnen.

4.2.6 getParamset

```
Paramset getParamset(String address, String paramset_key) /*(1)*/

Paramset getParamset(String address, String paramset_key, Integer mode)   /*(2) nur BidcosRF*/
```

Mit dieser Methode wird ein komplettes Parameter-Set für ein logisches Gerät gelesen. Der Parameter address ist die Addresses eines logischen Gerätes. Der Parameter paramset_key ist „MASTER“, „VALUES“ oder die Adresse eines Kommunikationspartners für das entsprechende Link-Parameter-Set (siehe getLinkPeers).

Dem Parameter mode können folgende Werte übergeben werden:

- 0 default: Keien Auswirkung, die Funktion verhält sicht wie der Aufruf ohne mode
- 1 UndefinedValues: Jeder Eintrag inerhalb des zurückgelieferten Paramset ins eine Struktur mit folgendem Aufbau:
  „UNDEFINED“(Boolean) Flag ob der angeforderte Wert initial gesetzt wurde und somit wahrscheinlich nicht der Realität entspricht oder ob der Wert von einem Gerät empfangen wurde, true = Wert wurde initial gesetzt und noch nicht verändert, false = der Wert wurde neu gesetzt
  „VALUE“(ValueType) Wert des angeforderten Parameter.
  UndefindeValues kann nur für Parameter aus dem Parameterset „VALUES“ abgefragt werden.

4.2.7 putParamset

```
void putParamset(String address, String paramset_key, Paramset set)

void putParamset(String address, String paramset_key, Paramset set, String rx_mode) /*(2) nur BidcosRF*/
```

Mit dieser Methode wird ein komplettes Parameter-Set für ein logisches Gerät geschrieben. Der Parameter address ist die Addresses eines logischen Gerätes. Der Parameter paramset_key ist „MASTER“, „VALUES“ oder die Adresse eines Kommunikationspartners für das entsprechende Link-Parameter-Set (siehe getLinkPeers).

Der Parameter set ist das zu schreibende Parameter-Set. In set nicht vorhandene Member werden einfach nicht geschrieben und behalten ihren alten Wert.

(2)

Es handelt sich um eine Erweiterung für Geräte, die sowohl den rx_mode BURST als auch

WAKEUP unterstützen. Mit dem Parameter rx_mode kann in diesem Fall angegeben werden, ob das übergebene Paramset über BURST oder WAKEUP übertragen werden soll. Gültige Werte sind:

- BURST – Übertragung mit Burst
- WAKEUP – Übertragung unter Verwendung des WakeUp Verfahrens.

4.2.8 getValue

```
ValueType getValue(String address, String value_key)   /*(1)*/
ValueType getValue(String address, String value_key, Integer mode)   /*(2) nur BidcosRF*/
```

Mit dieser Methode wird ein einzelner Wert aus dem Parameter-Set „VALUES“ gelesen. Der Parameter `address` ist die Addresse eines logischen Gerätes. Der Parameter `value_key` ist der Name des zu lesenden Wertes. Die möglichen Werte für value_key ergeben sich aus der ParamsetDescription des entsprechenden Parameter-Sets „VALUES“.

Dem Parameter `mode` können folgende Werte übergeben werden:

- 0 default: Keien Auswirkung, die Funktion verhält sicht wie der Aufruf ohne `mode`
- 1 UndefinedValues: Es wird ein Struktur zurückgeliefert die folgenden Aufbau hat:
  „UNDEFINED“(Boolean) Flag ob der angeforderte Wert initial gesetzt wurde und somit wahrscheinlich nicht der Realität entspricht oder ob der Wert von eimen Gerät empfangen wurde, true = Wert wurde initial gesetzt und noch nicht verändert, false = der Wert wurde neu gesetzt
  „VALUE“(ValueType) Wert des angeforderten Parameter.
  UndefindeValues kann nur für Parameter aus dem Parameterset „VALUES“ abgefragt werden die mit OPERATIONS = Read gekenzeichnet sind.

4.2.9 setValue

```
void setValue(String address, String value_key, ValueType value)
void setValue(String address, String value_key, ValueType value, String rx_mode) /*(2) nur Bidcos-RF)*/
```

Mit dieser Methode wird ein einzelner Wert aus dem Parameter-Set „VALUES“ geschrieben. Der Parameter `address` ist die Addresse eines logischen Gerätes. Der Parameter `value_key` ist der Name des zu schreibenden Wertes. Die möglichen Werte für `value_key` ergeben sich aus der ParamsetDescription des entsprechenden Parameter-Sets „VALUES“. Der Parameter `value` ist der zu schreibende Wert.

(2)

Es handelt sich um eine Erweiterung für Geräte, die sowohl den rx_mode BURST als auch WAKEUP unterstützen. Mit dem Parameter rx_mode kann in diesem Fall angegeben werden, ob der übergebene Wert über BURST oder WAKEUP übertragen werden soll. Gültige Werte

sind:

- BURST – Übertragung mit Burst
- WAKEUP – Übertragung unter Verwendung des WakeUp Verfahrens

4.2.10 getLinks

```
Array<Struct>getLinks(String address, Integer flags)
```

Diese Methode gibt alle einem logischen Kanal oder Gerät zugeordneten Kommunikationsbeziehungen zurück.

Der Parameter `address` ist die Kanal- oder Geräteadresse des logischen Objektes, auf das sich die Abfrage bezieht. Bei `address==""` werden alle Kommunikationsbeziehungen des gesamten Schnittstellenprozesses zurückgegeben.

Der Parameter `flags` ist ein bitweises oder folgender Werte:

- 1 = `GL_FLAG_GROUP`
  Wenn `address` einen Kanal bezeichnet, der sich in einer Gruppe befindet, werden die Kommunikationsbeziehungen für alle Kanäle der Gruppe zurückgegeben.
- 2 = `GL_FLAG_SENDER_PARAMSET`
  Das Feld `SENDER_PARAMSET` des Rückgabewertes wird gefüllt.
- 4 = `GL_FLAG_RECEIVER_PARAMSET`
  Das Feld `RECEIVER_PARAMSET` des Rückgabewertes wird gefüllt.

`flags` ist optional. Defaultwert ist 0x00.

Der Rückgabewert ist ein Array von Strukturen. Jede dieser Strukturen enthält die folgenden Felder:

- `SENDER`
  Datentyp String. Adresse des Senders der Kommunikationsbeziehung
- `RECEIVER`
  Datentyp String. Adresse des Empfängers der Kommunikationsbeziehung
- `FLAGS`
  Datentyp Integer. `FLAGS` ist ein bitweises oder folgender Werte:
  - `1=LINK_FLAG_SENDER_BROKEN`
    Diese Verknüpfung ist auf der Senderseite nicht intakt
  - `2=LINK_FLAG_RECEIVER_BROKEN`
    Diese Verknüpfung ist auf der Empfängerseite nicht intakt
- `NAME`
  Datentyp String. Name der Kommunikationsbeziehung
- `DESCRIPTION`
  Datentyp String. Textuelle Beschreibung der Kommunikationsbeziehung
- `SENDER_PARAMSET`
  Datentyp Paramset. Parametersatz dieser Kommunikationsbeziehung für die Senderseite

- RECEIVER_PARAMSET
  Datentyp Paramset. Parametersatz dieser Kommunikationsbeziehung für die Empfängerseite

4.2.11 addLink

```
void addLink(String sender, String receiver, String name, String description)
```

Diese Methode erstellt eine Kommunikationsbeziehung zwischen zwei logischen Geräten. Die Parameter sender und receiver bezeichnen die beiden zu verknüpfenden Partner. Die Parameter name und description sind optional und beschreiben die Verknüpfung näher.

4.2.12 removeLink

```
void removeLink(String sender, String receiver)
```

Diese Methode löscht eine Kommunikationsbeziehung zwischen zwei Geräten. Die Parameter sender und receiver bezeichnen die beiden Kommunikationspartner deren Kommunikationszuordnung gelöscht werden soll.

4.3 Optionale Methoden

4.3.1 setLinkInfo

BidCos-RF, BidCos-Wired

```
void setLinkInfo(String sender, String receiver, String name, String description)
```

Diese Methode ändert die beschreibenden Texte einer Kommunikationsbeziehung. Die Parameter sender und receiver bezeichnen die beiden zu verknüpfenden Partner. Die Parameter name und description beschreiben die Verknüpfung textuell.

4.3.2 getLinkInfo

BidCos-RF, BidCos-Wired

```
Array<String> getLinkInfo(String sender_address, String receiver_address)
```

Diese Methode gibt den Namen und die Beschreibung für eine bestehende Kommunikations-beziehung zurück. Die Parameter sender_address und receiver_address bezeichnen die beiden verknüpften Partner.

4.3.3 activateLinkParamset

BidCos-RF

```
void activateLinkParamset(String address, String peer_address, Boolean long_press)
```

Mit dieser Methode wird ein Link-Parameterset aktiviert. Das logische Gerät verhält sich dann so als ob es direkt von dem entsprechenden zugeordneten Gerät angesteuert worden

wäre. Hiermit kann z.B. ein Link-Parameter-Set getestet werden. Der Parameter `address` ist die Addresses des anzusprechenden logischen Gerätes. Der Parameter `peer_address` ist die Addresse des Kommunikationspartners, dessen Link-Parameter-Set aktiviert werden soll.

Der Parameter `long_press` gibt an, ob das Parameterset für den langen Tastendruck aktiviert werden soll.

4.3.4 determineParameter

BidCos-RF

```
void determineParameter(String address, String paramset_key, String parameter_id)
```

Mit dieser Methode wird ein Parameter eines Parameter-Sets automatisch bestimmt. Der Parameter kann bei erfolgreicher Ausführung anschließend sofort über getParamset gelesen werden.

Der Parameter `address` ist die Addresses eines logischen Gerätes.

Der Parameter `paramset_key` ist „MASTER“, „VALUES“ oder die Adresse eines Kommunikationspartners für das entsprechende Link-Parameter-Set (siehe getLinkPeers).

Der Parameter `parameter_id` bestimmt den automatisch zu bestimmenden Parameter.

4.3.5 deleteDevice

BidCos-RF, BidCos-Wired

```
void deleteDevice(String address, Integer flags)
```

Diese Methode löscht ein Gerät aus dem Schnittstellenprozess.

Der Parameter `address` ist die Addresse des zu löschenden Gerätes.

`flags` ist ein bitweises oder folgender Werte:

- `0x01=DELETE_FLAG_RESET`
  Das Gerät wird vor dem Löschen in den Werkszustand zurückgesetzt
- `0x02=DELETE_FLAG_FORCE`
  Das Gerät wird auch gelöscht, wenn es nicht erreichbar ist
- `0x04=DELETE_FLAG_DEFER`
  Wenn das Gerät nicht erreichbar ist, wird es bei nächster Gelegenheit gelöscht

4.3.6 abortDeleteDevice

BidCos-RF

```
void abortDeleteDevice(String address)
```

Diese Methode bricht einen anhängigen Löschvorgang für ein Gerät ab.

Es können nur Löschvorgänge abgebrochen werden, die mit dem Flag `DELETE_FLAG_DEFER` ausgeführt wurden.

4.3.7 setInstallMode

BidCos-RF

```
void setInstallMode(Boolean on)                                    /*(1)*/
void setInstallMode(Boolean on, Integer time, Integer mode)
/*(2)*/
void setInstallMode(Boolean on, Integer time, String address)
/*(3)*/
```

Diese Methode aktiviert und deaktiviert den Installations-Modus, in dem neue Geräte an der HomeMatic-CCU angemeldet werden können.

Der Parameter `on` bestimmt, ob der Installations-Modus aktiviert oder deaktiviert werden soll.

(2)

Der Parameter `time` bestimmt die Zeit in Sekunden die der Installations-Modus aktiviert ist

Der Parameter `mode` bestimmt die Art des Installations-Modus:

1 = Normaler Installations-Modus

2 = Während des Anlernens werden in den Parametersets „MASTER“ default Parameter gesetzt und alle besehenden Verknüpfungen werden gelöscht.

(3)

Installationsmodus mit Gerätefilter. Es kann nur das Gerät mit der angegebenen Seriennummer angelernt werden. Alle Anlernversuche von Geräten mit anderer Seriennummer werden ignoriert.

Der Parameter `time` bestimmt die Zeit in Sekunden die der Installations-Modus aktiviert ist

Der Parameter `address` ist die Seriennummer des Gerätes, dass angelernt werden soll.

4.3.8 getInstallMode

BidCos-RF

```
Integer getInstallMode(void)
```

Diese Methode gibt die verbliebene Restzeit in Sekunden im Anlernmodus zurück. Der Wert 0 bedeutet, der Anlernmodus ist nicht aktiv.

4.3.9 getKeyMissmatchDevice

BidCos-RF

```
String getKeyMissmatchDevice(bool reset)
```

Diese Methode gibt die Seriennummer des letzten Gerätes zurück, das aufgrund eines falschen AES-Schlüssels nicht angelernt werden konnte. Mit `reset=true` wird diese Information im Schnittstellenprozess zurückgesetzt.

4.3.10 setTempKey

BidCos-RF

```
void setTempKey(String passphrase)
```

Diese Methode ändert den von der CCU verwendeten temporären AES-Schlüssel. Der temporäre AES-Schlüssel wird verwendet, um ein Gerät anzulernen, in dem ein anderer Schlüssel gespeichert ist als der Schlüssel der CCU.

4.3.11 addDevice

BidCos-RF

```
DeviceDescription addDevice(String serial_number)                /*(1)*/
DeviceDescription addDevice(String serial_number,Integer mode) /*(2)*/
```

Diese Methode lernt ein Gerät anhand seiner Seriennummer an die CCU an. Diese Funktion wird nicht von jedem Gerät unterstützt. Rückgabewert ist die DeviceDescription des neu angelernten Geräts.

(2)

Der Parameter `mode` bestimmt die Art des Installations-Modus:

1 = Normaler Installations-Modus

2 = Während des Anlernens werden in den Parametersets „MASTER“ default Parameter gesetzt und alle besehenden Verknüpfungen werden gelöscht.

4.3.12 searchDevices

BidCos-Wired

```
int searchDevices(void)
```

Diese Methode durchsucht den Bus nach neuen Geräten und gibt die Anzahl gefundener Geräte zurück. Die neu gefundenen Geräte werden mit `newDevices` der Logikschicht gemeldet.

4.3.13 changeKey

BidCos-RF

```
void changeKey(String passphrase)
```

Diese Methode ändert den vom Schnittstellenprozess verwendeten AES-Schlüssel. Der Schlüssel wird ebenfalls in allen angelernten Geräten getauscht.

4.3.14 listTeams

BidCos-RF

```
Array<DeviceDescription> listTeams(void)
```

Diese Methode gibt alle dem Schnittstellenprozess bekannten Teams in Form von Geräte-beschreibungen zurück.

4.3.15 setTeam

BidCos-RF

```
void setTeam(String channel_address, String team_address)
```

Diese Methode fügt den Kanal `channel_address` zum Team `team_address` hinzu. Bei `team_address==""` wird der Kanal `channel_address` seinem eigenen Team zugeordnet.

Dabei muss `team_address` entweder leer sein ("") oder eine Seriennummer eines existierenden Teams enthalten.

Teams werden dabei je nach Bedarf erzeugt und gelöscht.

4.3.16 restoreConfigToDevice

BidCos-RF

```
void restoreConfigToDevice(String address)
```

Diese Methode überträgt alle zu einem Gerät in der CCU gespeicherten Konfigurationsdaten erneut an das Gerät.

4.3.17 clearConfigCache

BidCos-RF, BidCos-Wired

```
void clearConfigCache(String address)
```

Diese Methode löscht alle zu einem Gerät in der CCU gespeicherten Konfigurationsdaten. Diese werden nicht sofort wieder vom Gerät abgefragt, sondern wenn sie das nächste mal benötigt werden.

4.3.18 rssiInfo

BidCos-RF

```
Struct rssiInfo(void)
```

Gibt ein zweidimensionales assoziatives Array zurück, dessen Schlüssel die Geräteadressen sind. Die Felder des assoziativen Arrays sind Tupel, die die Empfangsfeldstärken zwischen beiden Schlüsselgeräten für beide Richtungen in dbm angeben. ein Wert von 65536 bedeutet, dass keine Informationen vorliegen.

- `Rückgabewert[<Gerät 1>][<Gerät 2>][0]`
  Empfangsfeldstärke an Gerät 1 für Sendungen von Gerät 2
- `Rückgabewert[<Gerät 1>][<Gerät 2>][1]`
  Empfangsfeldstärke an Gerät 2 für Sendungen von Gerät 1

4.3.19 updateFirmware

BidCos-Wired

```
Array<Boolean> updateFirmware(Array<String> device)
```

Diese Methode führt ein Firmware-Update für das in device angegebene Gerät durch. Das Gerät wird durch seine Seriennummer spezifiziert.

Der Rückgabewert gibt an, ob das Firmware-Update erfolgreich war.

BidCos-RF

```
Boolean updateFirmware(String device)
```

Diese Methode führt ein Firmware-Update für das in device angegebene Gerät durch. Das Gerät wird durch seine Seriennummer spezifiziert.

Der Rückgabewert gibt an, ob das Firmware-Update erfolgreich war.

4.3.20 getLinkPeers

BidCos-RF, BidCos-Wired

```
Array<String> getLinkPeer(String address)
```

Diese Methode gibt alle einem logischen Gerät zugeordneten Kommunikationspartner zurück. Die zurückgegebenen Werte können als Parameter `paramset_key` für getParamset() und putParamset() verwendet werden. Der Parameter `address` ist die Adresse eines logischen Gerätes.

4.3.21 logLevel

BidCos-RF, BidCos-Wired

```
int logLevel(void);       /* (1) */
int logLevel(int level);  /* (2) */
```

Diese Methode gibt den aktuellen Log-Level zurück (1) bzw. setzt diesen (2). Folgende Werte sind für `level` möglich:

- `6=LOG_FATAL_ERROR:` Nur schwere Fehler loggen.
- `5=LOG_ERROR:` Zusätzlich normale Fehler loggen.
- `4=LOG_WARNING:` Zusätzlich Warnungen loggen.
- `3=LOG_NOTICE:` Zusätzlich Notizmeldungen loggen.
- `2=LOG_INFO:` Zusätzlich Infomeldungen loggen.
- `1=LOG_DEBUG:` Zusätzlich Debugmeldungen loggen.
- `0=LOG_ALL:` Alles wird geloggt.

4.3.22 reportValueUsage

BidCos-RF, BidCos-Wired

```
Boolean reportValueUsage(String address, String value_id,
    Integer ref_counter)
```

Diese Methode teilt dem Interfaceprozess in `ref_counter` mit, wie oft der Wert `value_id` des Kanals `address` innerhalb der Logikschicht (z.B. in Programmen) verwendet wird. Dadurch kann der Interfaceprozess die Verbindung mit der entsprechenden Komponente herstellen bzw. löschen. Diese Funktion sollte bei jeder Änderung aufgerufen werden.

Der Rückgabewert ist `true`, wenn die Aktion sofort durchgeführt wurde. Er ist `false`, wenn die entsprechende Komponente nicht erreicht werden konnte und vom Benutzer zunächst in den Config-Mode gebracht werden muss. Der Interfaceprozess hat dann aber die

neue Einstellung übernommen und wird sie bei nächster Gelegenheit automatisch an die Komponente übertragen.

In diesem Fall ist dann auch der Wert CONFIG_PENDING im Kanal MAINTENANCE der Komponente gesetzt.

4.3.23 setBidcosInterface

BidCos-RF

```
Void setBidcosInterface(String device_address, String interface_address,
    Boolean roaming)
```

Diese Methode setzt das für die Kommunikation mit dem durch `device_address` spezifizierten Gerät verwendete Bidcos-Interface. Die Seriennummer des in Zukunft für die Kommunikation mit diesem Gerät zu verwendenden Interfaces wird in `interface_address` übergeben. Ist der Parameter `roaming` gesetzt, so wird die Interfacezuordnung für das Gerät automatisch in Abhängigkeit von der Empfangsfeldstärke angepasst. Das ist für nicht ortsfeste Geräte wie Fernbedienungen sinnvoll.

4.3.24 listBidcosInterfaces

BidCos-RF

```
Array<Struct>listBidcosInterfaces()
```

Diese Methode gibt eine Liste aller vorhandenen BidCoS-Interfaces in Form eines Arrays von Structs zurück.

Der Rückgabewert ist ein Array von Strukturen. Jede dieser Strukturen enthält die folgenden Felder:

- `ADDRESS`
  Datentyp String. Seriennummer des BidCoS-Interfaces.
- `DESCRIPTION`
  Datentyp String. Textuelle Beschreibung des Interfaces wie in der Konfigurationsdatei für den Schnittstellenprozess angegeben.
- `CONNECTED`
  Datentyp Boolean. Gibt an, ob zum Zeitpunkt der Abfrage eine Kommunikationsverbindung zum Interface besteht.
- `DEFAULT`
  Datentyp Boolean. Gibt an, ob es sich um das Standardinterface handelt. Das Standardinterface wird verwendet, wenn das einem Gerät zugeordnete Interface nicht mehr existiert.
- `TYPE`
  Datentyp String. Enthält den Typ des Interfaces, wie in der Konfigurationsdatei angegeben.
- `FIRMWARE_VERSION`
  Enthält die Firmware-Version des Bidcos-Interfaces. Abhängig vom jeweiligen Interface kann es eine Weile dauern, bis dieser Wert verfügbar ist. Solange die Firmware-Version nicht initialisiert ist, enthält das Feld die leere Zeichenkette.

- DUTY_CYCLE
  Datentyp Integer. Enthält die aktuelle Duty-Cycle-Auslastung, Auflösung 1%.

4.3.25 getServiceMessages

BidCos-RF

```
Array<Array> getServiceMessages()
```

Diese Methode gibt eine Liste aller vorhandenen Servicemeldungen in Form eines Arrays zurück.

Der Rückgabewert ist ein Array mit einem Element pro Servicemeldung. Jedes Element ist wiederum ein Array mit

drei Feldern:

- Rückgabewert[index][0]
  Datentyp String. Adresse (Seriennummer) des Kanals, der die Servicemeldung generiert hat
- Rückgabewert[index][1]
  Datentyp String. ID der Servicemeldung (CONFIG_PENDING, UNREACH, etc.)
- .Rückgabewert[index][2]
  Datentyp variabel. Wert der Servicemeldung

4.3.26 setMetadata

BidCos-RF

```
void setMetadata(String object_id, String data_id, Variant value)
```

Diese Methode setzt ein Metadatum zu einem Objekt.

object_id ist die Id des Metadaten-Objekts. Üblicherweise ist dies die Seriennummer eines Gerätes oder Kanals.

Durch Übergabe einer beliebigen Id können aber auch eigene Metadaten-Objekte angelegt werden.

data_id ist die Id des zu setzenden Metadatums. Diese Id kann frei gewählt werden.

value ist eine beliebige Variable. Diese wird gespeichert und kann später mittels getMetadata() und getAllMetadata() wieder abgefragt werden.

4.3.27 getMetadata

BidCos-RF

```
Variant getMetadata(String object_id, String data_id)
```

Diese Methode gibt ein Metadatum zu einem Objekt zurück.

object_id ist die Id des Metadaten-Objekts. Üblicherweise ist dies die Seriennummer eines Gerätes oder Kanals.

Durch Übergabe einer beliebigen Id können aber auch eigene Metadaten-Objekte angelegt werden.

`data_id` ist die Id des abzufragenden Metadatums

. Diese Id kann frei gewählt werden.

Der Rückgabewert entspricht in Datentyp und Wert der zuvor an `setMetadata()` als Parameter `value` übergebenen Variablen.

4.3.28 getAllMetadata

BidCos-RF

```
Struct getAllMetadata(String object_id)
```

Diese Methode gibt alle zuvor gesetzten Metadaten zu einem Objekt zurück.

`object_id` ist die Id des Metadaten-Objekts. Üblicherweise ist dies die Seriennummer eines Gerätes oder Kanals.

Durch Übergabe einer beliebigen Id können aber auch eigene Metadaten-Objekte angelegt werden.

Der Rückgabewert ist ein Struct, der zu jedem zuvor gesetzten Metadatum ein Feld enthält. Der Feldname ist der zuvor an `setMetadata()` als Parameter `data_id` übergebene Wert. Der Wert des Feldes entspricht in Datentyp und Wert der zuvor an `setMetadata()` als Parameter `value` übergebenen Variablen.

4.3.29 HasVolatileMetadata (nur zur internen Verwendung von eQ-3)

BidCos-RF

```
bool hasVolatileMetadata(String key)
```

Prüft, ob für einen Schlüssel flüchtige Metadaten vorhanden sind. Flüchtige Metadaten werden durch den BidCoS-Service nicht persistiert.

4.3.30 setVolatileMetadata (nur zur internen Verwendung von eQ-3)

BidCos-RF

```
void setVolatileMetadata(String key, Variant value)
```

Speichert flüchtige Metadaten zu einem Schlüssel. Flüchtige Metadaten werden durch den BidCoS-Service nicht persistiert.

4.3.31 getVolatileMetadata (nur zur internen Verwendung von eQ-3)

BidCos-RF

```
Variant getVolatileMetadata(String key)
```

Liefert flüchtige Metadaten zu einem Schlüssel. Flüchtige Metadaten werden durch den BidCoS-Service nicht persistiert.

4.3.32 DeleteVolatileMetadata (nur zur internen Verwendung von eQ-3)

BidCos-RF

```
void deleteVolatileMetadata(String key)
```

Löscht flüchtige Metadaten zu einem Schlüssel.

4.3.33 getVersion

BidCos-RF

```
string getVersion ()
```

Liefert die Version des BidCoS-Service'.

4.3.34 setInterfaceClock

BidCoS-RF

```
bool setInterfaceClock(int utcSeconds, int offsetMinutes)
```

Setzt die UTC Zeit für alle Interfaces, die dies benötigen.

Mit dem Parameter `utcSeconds` werden die Sekunden seti 01.01.1970 00:00 Uhr (UTC) gesetzt.

Über den Parameter offsetMinutes wird der Offset in Minuten entsprechend der jeweiligen Zeitzone übergeben.

Der Rückgabewert der Methode ist `false` im Fehlerfall, ansonsten `true`.

4.3.35 replaceDevice

BidCoS-RF, BidCoS-Wired

```
bool replaceDevice(String oldDeviceAddress, String newDeviceAddress)
```

Mit dieser Funktion kann ein altes gegen ein neues Gerät ausgetauscht werden.

Alle direkten Geräteverknüpfungen und Konfigurationen werden auf das neue Gerät kopiert und das alte Gerät gelöscht.

Die Beiden Geräte müssen hinsichtlich ihrer Funktionalität kompatibel sein. Mit der Mathode `listReplaceableDevice()` kann eine List kompatibeler Geräte abgefragt werden.

Das neue Gerät muss an dem Schnittstellenprozess angemeldet sein und darf noch nicht in Verknüpfungen verwendet werden.

Über die Parameter `oldDeviceAddress` und `newDeviceAddress` wird der Methode die Adresse des alten Gerätes und des neuen Gerätes übergeben.

Der Rückgabewert ist `true` wenn der Tausch erfolgreich war, ansonsten `false`

4.3.36 listReplaceableDevice

BidCoS-RF, BidCoS-Wired

```
Array<DeviceDescription>  listReplaceableDevices(String newDeviceAddress)
```

Mit dieser Funktionen kann eine Liste der Geräte angefordert werden die durch das übergebene Gerät ersetzt werden können.

Über den Parameter `newDeviceAddress` wird die Adresse des neuen Geräts übergeben für die die möglichen Tauschpartner ermittelt werden sollen.

Der Rückgabewert ist ein Array der DeviceDescriptions. Diese Array enthält die Geräte und die Gerätekanäle.

4.3.37 Ping

BidCoS-RF, BidCoS-Wired

```
bool ping(String callerId)
```

Beim Aufruf dieser Funktion wird ein Event (im Folgenden PONG genannt) erzeugt und an alle registrierten Logikschichten versandt. Da das PONG Event an alle registrierten Logikschichten (wie bei allen anderen Events auch) verschickt wird, muss in einer Logikschicht damit gerechnet werden, ein PONG Event zu empfangen ohne zuvor ping aufgerufen zu haben.

Der Parameter `callerId` ist vom Aufrufer zu übergeben und wird als Wert des PONG Events verwendet. Der Inhalt des String ist unerheblich.

Tritt während der Verarbeitung keine Exception auf, so wird von der Methode `true` zurückgegeben.

Das PONG Event wird über die event Methode der Logikschicht ausgeliefert. Die Adresse ist dabei immer „CENTRAL“, der key lautet „PONG“ und der Wert ist die im ping Aufruf übergebene `callerId`.

4.3.38 refreshDeployedDeviceFirmwareList

BidCoS-RF

```
void refreshDeployedDeviceFirmwareList ()
```

Beim Aufruf dieser Funktion liest der Schnittstellenprozess die installierten Geräte Firmware Update Files neu ein und aktualisiert seine internen Strukturen. Diese Methode sollte immer dann aufgerufen werden, wenn eine Update File neu installiert oder gelöscht wurde.

Methoden der Logikschicht

Eine Logikschicht kann ihrerseits über einen XML-RPC-Server verfügen und sich per *init* bei den Schnittstellenprozessen anmelden. In diesem Fall erhält die Logikschicht diverse Informationen direkt, ohne explizit Nachfragen zu müssen, z.B. ob sich die Konfiguration eines Gerätes geändert hat.

Damit der Ablauf reibungslos funktioniert, muss der eingesetzte XML-RPC-Server zwingend die Standardmethode *system.multicall* unterstützen.

4.4 event

```
void event(String interface_id, String address, String value_key, ValueType value)
```

Mit dieser Methode teilt der Schnittstellenprozess der Logikschicht mit, dass sich ein Wert geändert hat oder ein Event (z.B. Tastendruck) empfangen wurde.

Der Parameter `interface_id` gibt die id des Schnittstellenprozesses an, der das Event sendet.

Der Parameter `address` ist die Addresse des logischen Gerätes, zu dem der geänderte Wert / das Event gehört.

Der Parameter `value_key` ist der Name des entsprechenden Wertes. Die möglichen Werte für value_key ergeben sich aus der ParamsetDescription des entsprechenden Parameter-Sets „VALUES“.

Der Parameter `value` gibt den neuen Wert bzw. den dem Event zugeordneten Wert an. Der Datentyp von `value` ergibt sich aus der ParamsetDescription des Values-Parameter-Sets des entsprechenden logischen Gerätes.

4.4.1 Besondere Events

Es gibt einige besondere Events, die über die Methode „event“ publiziert werden. Diese werden im Folgenden aufgezählt und kurz erläutert:

4.4.1.1 UNREACH

Das Event UNREACH wird immer dann erzeugt, wenn ein Kommunikationsfehler mit einem Gerät auftritt. Der Wert des Events ist vom Typ Boolean. Tritt eine Kommunikationsstörung auf, so wird ein UNREACH Event mit dem Wert `true` publiziert. Konnte daraufhin erfolgreich mit dem entsprechenden Gerät kommuniziert werden, wird das UNREACH Event mit dem Wert `false` publiziert. UNREACH gibt also Auskunft über den aktuellen Zustand.

4.4.1.2 STICKY_UNREACH

Das Event STICKY_UNREACH tritt zusammen mit dem UNREACH Event auf. Der Wert des Events ist vom Typ Boolean und immer `true`. Im Gegensatz zum UNREACH Event, wird das STICKY_UNREACH Event nicht erzeugt, wenn nach einer Kommunikationsstörung wieder erfolgreich mit einem Gerät kommuniziert werden kann. Das STICKY_UNREACH Event gibt also nicht Auskunft über den aktuellen Zustand, sondern gibt an, dass eine Kommunikationsstörung vorlag.

4.4.1.3 CONFIG_PENDING

Das Event CONFIG_PENDING gibt an, dass noch Konfigurationsdaten an ein Gerät übertragen werden müssen. Der Wert dieses Events ist vom Typ Boolean. Müssen noch Konfigurationsdaten an ein Gerät übertragen werden, weil dieses nicht direkt erreicht werden kann oder nicht alle Daten übertragen werden konnten, so wird das CONFIG_PENDING Event mit dem Wert `true` publiziert. Sind alle Konfigurationsdaten an ein Gerät übertragen worden, so wird das Event mit dem Wert `false` versendet (sofern zuvor CONFIG_PENDING mit `true` versandt wurde).

4.4.1.4 ERROR

(Nur BidCoS-RF)

Das ERROR Event wird versendet, wenn ein Gerät bei der Übertragung von Konfigurationsdaten mit einem Fehlercode antwortet. Der Wert des Events besteht aus einem Integer (Fehlercode) und einem String (kurze Beschreibung).

Fehler Codes:

```
{1, "Unknown Error"}
{2, "Busy"}
{3, "MemFull"}
{4, "Target Invalid"}
{5, "Channel Invalid"}
```

4.5 listDevices

```
Array<DeviceDescription> listDevices(String interface_id)
```

Diese Methode gibt alle der Logikschicht bekannten Geräte für den Schnittstellenprozess mit der Id `interface_id` in Form von Gerätebeschreibungen zurück. Damit kann der Schnittstellenprozess durch Aufruf von newDevices() und deleteDevices() einen Abgleich vornehmen.

Damit das funktioniert, muss sich die Logikschicht diese Informationen zumindest teilweise merken. Es ist dabei ausreichend, wenn jeweils die Member ADDRESS und VERSION einer DeviceDescription gesetzt sind.

4.6 newDevices

```
void newDevices(String interface_id , Array<DeviceDescription>
dev_descriptions)
```

Mit dieser Methode wird der Logikschicht mitgeteilt, dass neue Geräte gefunden wurden.

Der Parameter `interface_id` gibt die id des Schnittstellenprozesses an, zu dem das Gerät gehört.

Der Parameter `dev_descriptions` ist ein Array, das die Beschreibungen der neuen Geräte enthält.

Wenn `dev_descriptions` Geräte enthält, die der Logikschicht bereits bekannt sind, dann ist davon auszugehen, dass sich z.B. durch ein Firmwareupdate das Verhalten des Gerätes geändert hat. Die Basisplatform muß dann einen Abgleich mit der neuen Beschreibung durchführen. Dabei sollte die Konfiguration des Gerätes innerhalb der Logikschicht so weit wie möglich erhalten bleiben.

4.7 deleteDevices

```
void deleteDevices(String interface_id, Array<String>
addresses)
```

Mit dieser Methode wird der Logikschicht mitgeteilt, dass Geräte im Schnittstellenprozess gelöscht wurden.

Der Parameter `interface_id` gibt die id des Schnittstellenprozesses an, zu dem das Gerät gehört.

Der Parameter `addresses` ist ein Array, das die Adressen der gelöschten Geräte enthält.

4.8 updateDevice

```
void updateDevice(String interface_id, String address, int hint)
```

Mit dieser Methode wird der Logikschicht mitgeteilt, dass sich an einem Gerät etwas geändert hat.

Der Parameter `interface_id` gibt die id des Schnittstellenprozesses an, zu dem das Gerät gehört.

Der Parameter `address` ist die Adresse des Gerätes oder des Kanals, auf das sich die Meldung bezieht.

Der Parameter `hint` spezifiziert die Änderung genauer:

- `UPDATE_HINT_ALL=0`
  Es hat eine nicht weiter spezifizierte Änderung stattgefunden und es sollen daher alle möglichen Änderungen berücksichtigt werden.
- `UPDATE_HINT_LINKS=1`
  Es hat sich die Anzahl der Verknüpfungspartner geändert.

Derzeit werden nur Änderungen an den Verknüpfungspartnern auf diesem Weg mitgeteilt.

4.9 replaceDevice

```
void replaceDevice(String interface_id, String oldDeviceAddress, String newDeviceAddress)
```

Mit dieser Methode wird der Logikschicht mitgeteilt, dass ein Gerät getuscht wurde.

Der Parameter `interface_id` gibt die id des Schnittstellenprozesses an, zu dem das Gerät gehört.

Der Parameter `oldDeviceAddress` ist die Adresse des ersetzten Gerätes.

Der Parameter `newDeviceAddress` ist die Adresse des Gerätes welches an Stelle des alten Gerätes im System eingefügt wurde.

4.10 readdedDevice

```
void readdedDevice(String interfaceId, Array<String> addresses)
```

Diese Methode wird aufgerufen, wenn ein bereits angelerntes Gerät in den Anlernmodus versetzt wird, während der Installations-Modus aktiviert ist (vgl. setInstallMode).

Der Parameter `interface_id` gibt die id des Schnittstellenprozesses an, zu dem das Gerät gehört.

Der Parameter `addresses` ist ein Array, das die Adressen der gelöschten logischen Geräte enthält.

5 Fehlercodes

Fehlercode	Bedeutung
-1	Allgemeiner Fehler
-2	Unbekanntes Gerät / unbekannter Kanal
-3	Unbekannter Paramset
-4	Es wurde eine Geräteadresse erwartet
-5	Unbekannter Parameter oder Wert
-6	Operation wird vom Parameter nicht unterstützt
-7	Das Interface ist nicht in der Lage ein Update durchzuführen
-8	Es seht nicht genügend DutyCycle zur Verfügung
-9	Das Gerät ist nicht in Reichweite

eQ-3 AG
Maiburger Straße 29
D-26789 Leer
www.eQ-3.com