# History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Author, Editor | Reason |
| 1.0 | 19.03.2017 | Ivo Kunadt | Initial Version |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# RemoteSoftware

## RemoteApp

## RemoteHiddenHelper

Windows Services sind in bestimmten Dingen, gegenüber normalen Userapplikationen limitiert. So können sie z.B. keine Oberfläche besitzen oder Programme starten die über eine verfügen. Um diese Limitierungen zu umgehen wird eine Applikation in der Usersession gestartet.

Die Kommunikation findet überlicherweise über eine IPC Schnittstelle statt. Die RemoteCommunicationLibrary verfügt über ein IPC Interface. Dies wird automatisch für durch die Verwendung des CommunicationsServers verwendet.

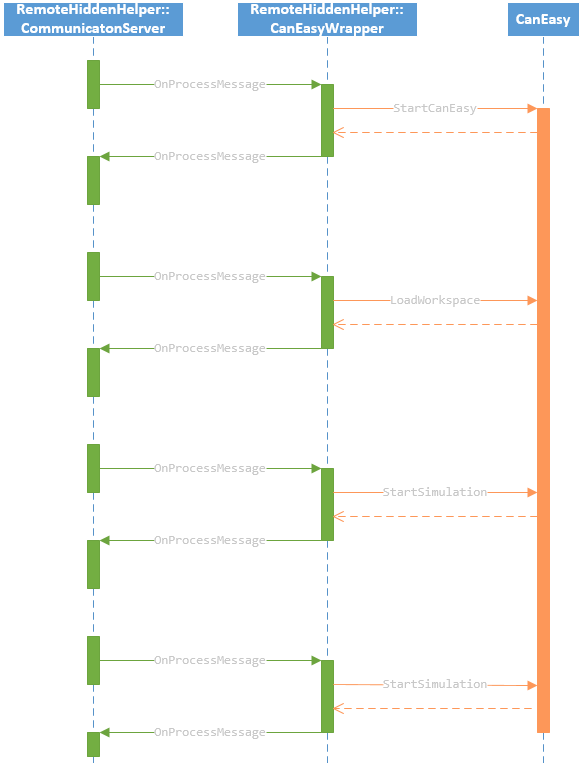
### CanEasyWrapper

Für den Flashvorgang ist die Übertragung von Daten bzw. Signalen notwendig. Für die Übertragung wird dabei CanEasy verwendet, welches auch im Hause Schleißheimer entwickelt wurde.

Das Interface ist dabei die COM API die von C++ aus genutzt werden kann.

Kommandoübersicht:

|  |  |
| --- | --- |
| **Funktion** | **Beschreibung der Funktion** |
| *CanEasyStartApplication* | *Startet CanEasy* |
| *CanEasyLoadWorkspace* | *Ladet den übergebenen Arbeitsbereich* |
| *CanEasyStartSimulation* | *Startet die CanEasySimulation* |
| *CanEasyStopSimulation* | *Stoppt die CanEasySimulation* |
| *CanEasyCloseApplication* | *Beendet CanEasy* |



### FhostWrapper

### MKSWrapper

### FileUtils

### InactivityWatcher

### PortalInfo

Das Integrationsportal von Continental bietet die Möglichkeit

### USBHidLoader

## RemoteService

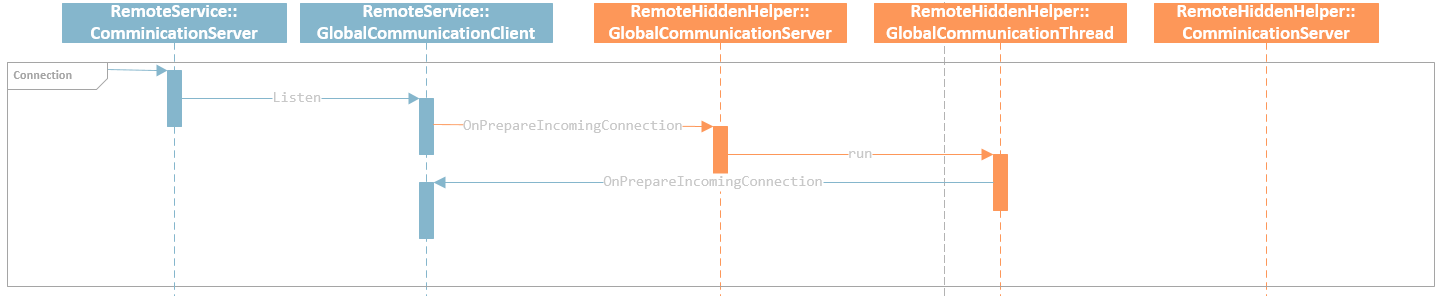
## RemoteUtil

Third Party Bibliotheken oder eigene Bibliotheken werden im Repository RemoteUtil abgelegt. Jede eigene Bibliothek muss über einen Order Bin, Lib und Include verfügen. Unter Include befindet sich der API Header für die Bereitstellung der Funktionen für andere Applikationen.

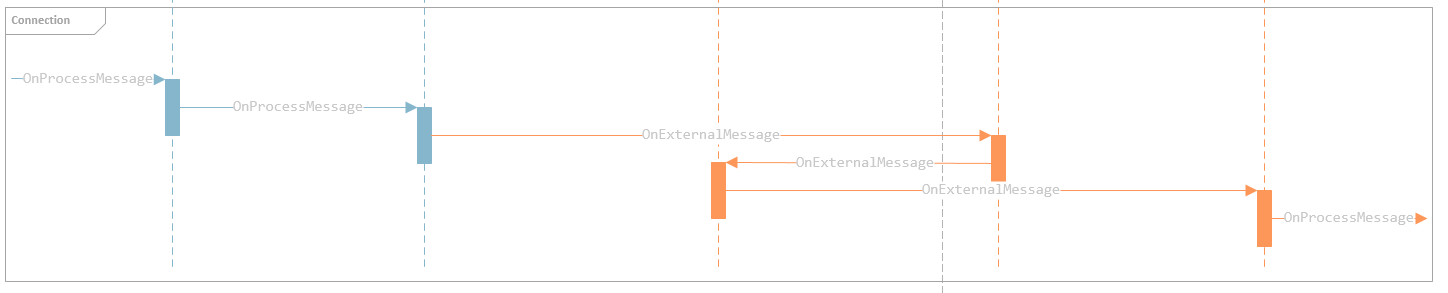
Für das Logging wird die Opensource Header Only Variante des SPDLOG verwendet. Sie ist unter folgendem GitHub Link abrufbar <https://github.com/gabime/spdlog>.

### RemoteCommunicationLibrary

|  |  |
| --- | --- |
| Header | #include "RemoteCommunicationLibrary.h" |
|  |  |



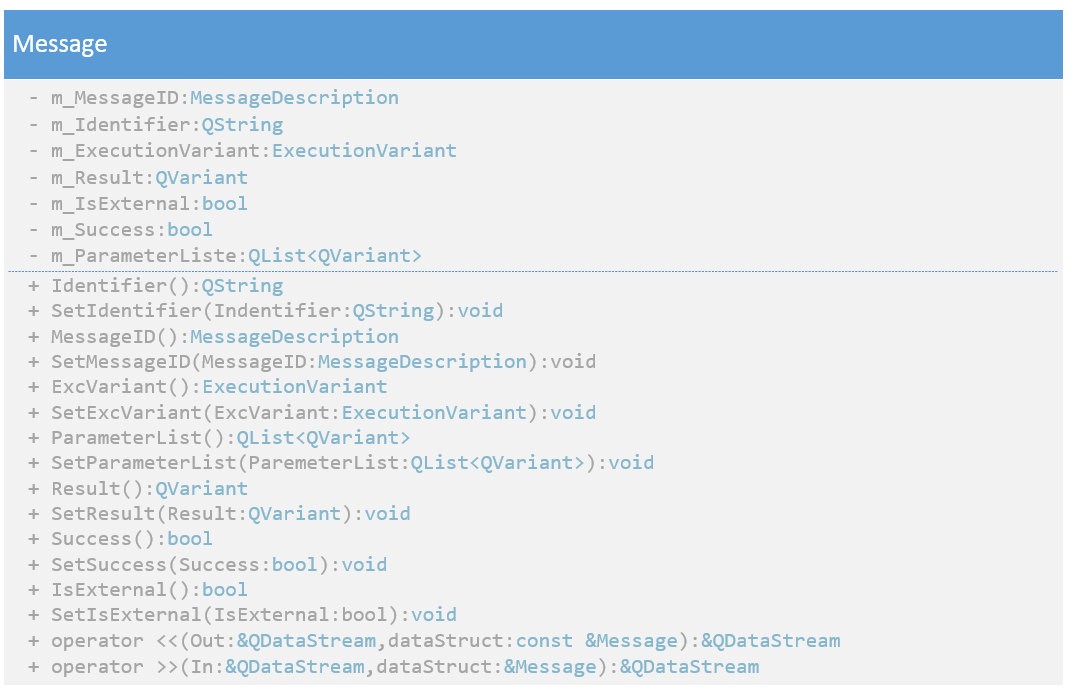
Nachdem die Verbindung mit dem Server hergestellt ist, können sowohl Client als auch Server, die Kommunikation initierien. Dabei wird dem CommincationsServer zunächst in der OnProcessMessage einen neue Message übermittelt. Diese wird je nach Empfänger



#### Message

Das Message Objekt kapselt alle wichtigen Eigenschaften, die für die Übertragung von Daten oder Ereignissen benötigt wird. Jedes Modul, die über den [CommunicationsServer](#_RemoteCommunicationLibrary) Informationen oder Ereignisse an der andere mitteln möchte, muss diese Klasse benutzen.

Sobald alle benötigten Informationen bereit stehen, kann die Message an die nächst weiterführende Instanz weitergeleited werden. Dies passiert durch das emitieren des Signals NewMessage(Message Msg).



Die **MessageID** ist der Adressat der Nachricht. Eine vollständige Liste aller möglichen Adressaten befindet sich im Header MessageDescription.h. Dabei wird zum einen zwischen internen und exteren Nachrichten unterschieden. Externe Nachrichten, sind Nachrichten, die der RemoteService empfängt, sie werden mit einem EX\_ Prefix kenntlich gemacht. Interne Nachrichten werden vom RemoteService für interne Abläufe verwendet und werden mit IN\_ Prefix gekennzeichnet. Benötigen auch andere Applikationen interne Nachrichten so werden diese mit einem weiteren Prefix gekennzeichnet

|  |  |
| --- | --- |
| **Applikation** | **Prefix** |
| *RemoteApp* | *IN\_RA\_XXX/EX\_RA\_XXX* |
| *RemoteHiddenHelper* | *IN\_RHH\_XXX/EX\_RHH\_XXX* |
| *RemoteView* | *IN\_RV\_XXX/EX\_RV\_XXX* |

Der **Identifier** kennzeichnet eindeutig den Absender der Botschaft und ist für externe Botschaften gedacht.

Die **Executionvariant** gibt an, ob es sich um eine Abfrage von Information handelt oder Daten mit dieser Botschaft gesetzt werden soll.

**ParameterListe** TODO Länge klären

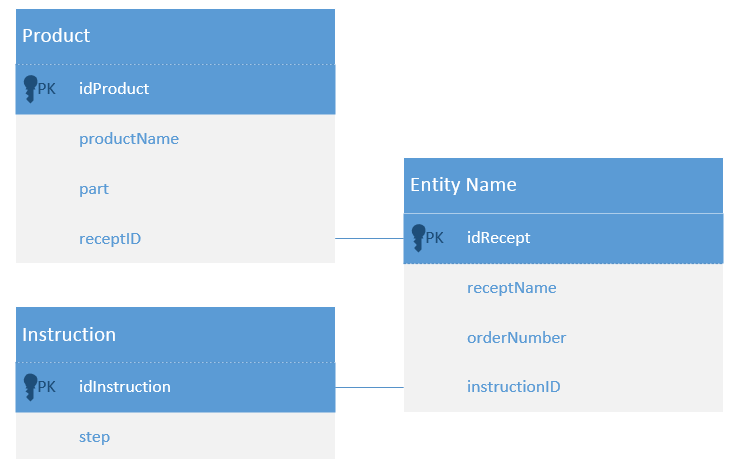
Werden Befehle zu einem Modul oder Applikationen gesendet, ist es eventuell wichtig zu erfahren, ob das Kommando erfolgreich prozessiert wurde. Dafür gibt es innerhalb der Message ein Flag **Success** und eine Möglichkeit Rückgabe Werte zu übermitteln **Result**

### RemoteDataConnectLibrary

Die RemotedataConnectLibrary dient als Schnittschnelle zwischen den Daten in der Datenbank und der Datenverwaltung in der Applikation, weiterhin ist es aber möglich jede andere Form von Datenquelle z.B. XML zu verwenden. Sie ist als ORM ausgelegt, bietet jedoch aktuell nicht die Möglichkeit über Transaktions.

Daten werden in sogenannten Entities aus der Datenbank abgelegt. Um die Zuordnung zwischen Entity und Datenbankinhalt kümmert sich ein DataMapper.

#### Enitities



#### DataMapper

#### Repository

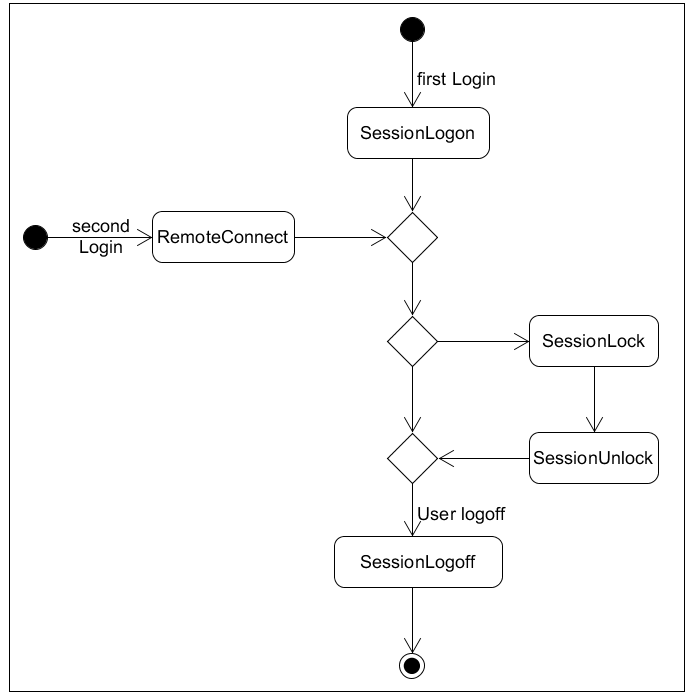
### RemoteBoxWrapper

## RemoteView

# Verbindungsaufbau

## Login

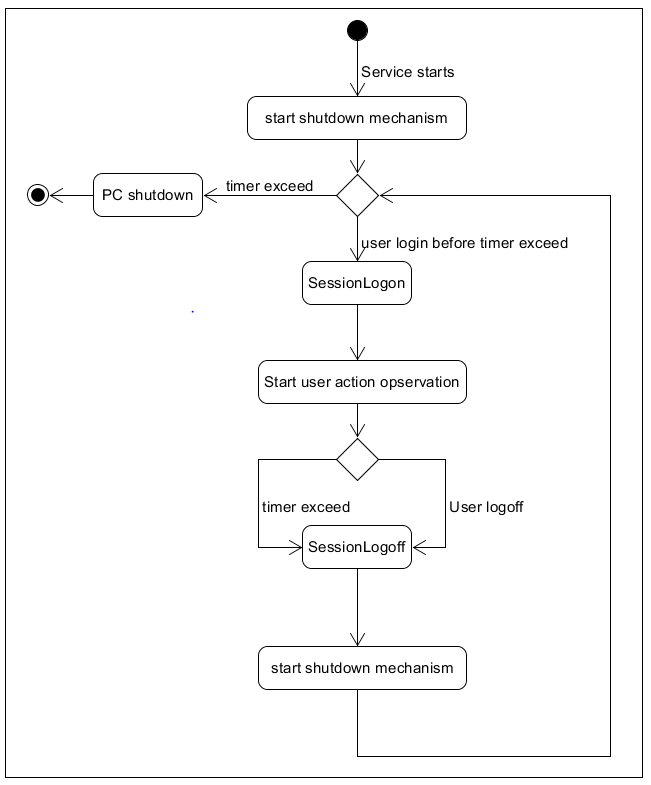
Beim erstmaligen Verbindungsaufbau wird der Status der Verbindung als SessionLogon angegeben. Erfolgt eine erneute Verbindung ändert sich der Status zu RemoteConnect. Loggt man sich aus erfolgt je nach dem Status zuvor entweder SessionLogoff oder RemoteDisconnect.



## Logout Mechanism

Um Rechner und Peripherie nicht unnötig zu belasten und um den Stromverbrauch zu senken werden die RemoteWorkstations nach einem bestimmten Ablaufe heruntergefahren.

Sobald der RemoteService gestartet ist, wird der Shutdown Mechanismus gestartet. Dabei wird ein Timer gestartet. Der Timeout des Timers wird aus dem Feld [ShutdownTimeout] der Collection [GeneralInformation] aus der Datenbank [remoteWorkstations] gelesen. Ist der Timer abgelaufen wird ein Event gesendet was dazu führt, dass der PC heruntergefahren wird. Der Grund für das sofortige starten des Shutdown Mechanismus liegt darin, dass ein versehentliches



Logt sich ein User vor dem Ablauf des Timers in den Rechner ein, wird der Shutdown Mechanismus beendet. Nun erfolgt der Start der User Observation. Auch hier wird erneut ein Timer gestartet, der aber bei jeder Aktion des Users (Tastatur- oder Mauseingaben) auf seinen Startwert zurückgesetzt wird. Der Timeout wird aus dem Feld [UserObservationTimeout] der Collection [GeneralInformation] aus der Datenbank [remoteWorkstations] gelesen.

Erfolgt keine Interaktion des Users mit der RemoteWorkstation innerhalb des Timeouts [UserObservationTimeout] statt, wird er automatisch aus dem Rechner ausgeloggt. Zu beachten ist das der Mechanismus nicht berücksichtigt, ob Programme mit umgespeicherten Änderungen laufen.

Um zu verhindern, dass man ausgeloggt wird gibt es die Möglichkeit des permanenten Logins. Dies deaktiviert den User Observation Mechanismus und verhindert, dass der User ausgeloggt wird. Nur einem Admin ist es jetzt noch möglich einen User aus einem Rechner zu werfen.

# Kommunikation

Daten werden unabhängig des verwendeten Protokolls in Form eines Kommandos verschickt.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| cmdId | src | dst | direction | parameter |

cmdid: Eindeutige ID des Kommandos. Liste aller Kommandos

src:

dst:

direction:

parameter:

Als Bestätigung schickt der Client das Kommando zurück und fügt noch hinzu ob die Ausführung erfolgreich war und etwaige Rückgabeparameter.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| cmdId | src | dst | direction | parameter | success | returnval |

## Remoteview/RemoteService Kommunikation

Die Kommunikation zwischen der RemoteView als Client und dem RemoteService als Server erfolgt über Websockets. Diese ermöglichen einen Socket ähnlichen Zugriff auf die Kommunikation wie von TCP oder UDP Socket bekannt ist.

Die Daten selbst werden im JSON Format übertragen. Dies ist ein kompaktes Datenformat in einer einfach lesbaren Textform zum Zweck des Datenaustauschs zwischen Anwendungen.

Beispiel für ein Kommando:

“cmd“ {

“cmdID“ : xxx,

“src“ : xxx,

“dst“ : xxx,

“direction“ : SET,

“parameter“ : [ “X1“, “X2“, “Xn“]

}

Beispiel für eine Kommandoantwort:

“cmd“ {

“cmdID“ : xxx,

“src“ : xxx,

“dst“ : xxx,

“direction“ : SET,

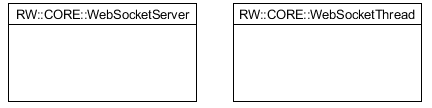
“parameter“ : [ “X1“, “X2“, “Xn“],

“success“ : true,

“returnval“ : [“X1“, “X2“ , “X2“ ]

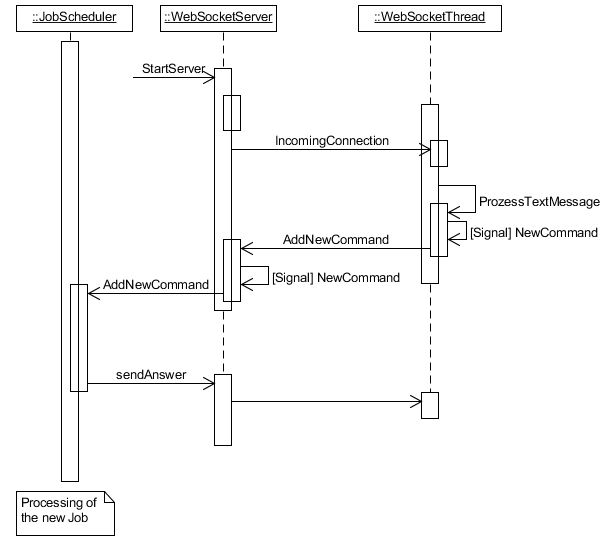
}

Der Remoteservice besitzt für die Kommunikation mit dem Client zwei Klassen



Der WebsocketServer kümmert sich um die Initialisierung des Websocketservers und stellt den entsprechenden Endpunkt für den Client zu Verfügung. Sobald ein Client sich mit dem Server verbindet, wir die eingehende Verbindung zu einem neuen Thread weitergeleitet. Dort erfolgt dann die Verarbeitung der Nachricht und wird nach dem Parsen durch auslösen eines Events an den Server gesendet. Dieser dient als Proxy für die Weiterleitung des Events an den Scheduler.

Der Scheduler schickt nach dem Ausführen des Kommandos eine Antwort an den Server der wiederum die Nachricht an den WebSocketThread weiterleitet. Als letzter Schritt bleibt dann noch die Übertragung der Antwort an den Client.



## RemoteApp/RemoteService Kommunikation

## JobScheduler

## Kommandoliste

|  |  |  |
| --- | --- | --- |
| USBCommand | 1 | Setzt oder liefert den Status des USB Ports |
| IOCommand | 2 | Setzt oder liefert den Status des IO Ports |
| RelayCommand | 3 | Setzt oder liefert den Status des Relays |
| ADCCommand | 4 | Liefert den aktuellen ADC Wert |
| DACCommand | 5 | Setzt den Wert für den DAC |
| FlashCommand | 6 | Flasht das Target mit der angebenden SoftwareVersion |
| GeneralInformation | 7 | Liefert allgemeine Informationen |
|  |  |  |

## FlashCommand

Parameter : AC Pfad, GC Pfad, Bootloader Pfad

Der Pfad in dem FHostSP versucht wird zu starten wird aus dem Feld [fhostsppath] der Collection [GeneralInformation] aus der Datenbank [remoteWorkstations] gelesen. Die Pfade für die zu flashenden Files werden aus dem Parameter gelesen.

Der Wert für das Timeout des Abbruches von FHostSP wird aus dem Feld [fhosttimeout] der Collection [GeneralInformation] aus der Datenbank [remoteWorkstations] gelesen.