|  |  |
| --- | --- |
| DESM Middleware |  |
| Sebastian Straube | sebastian.straube@desm.ch  +41 (0) 79 48 18 444  http://www.desm.ch |

# Inhaltsverzeichnis

[1 Inhaltsverzeichnis 1](#_Toc396338435)

[2 Kontaktliste 2](#_Toc396338436)

[3 Dokumentenhistorie 3](#_Toc396338437)

[4 Ausgangslage 4](#_Toc396338438)

[4.1 Systemansatz 4](#_Toc396338439)

[5 Ziele 5](#_Toc396338440)

[6 DESM Middleware 5](#_Toc396338441)

[6.1 Systemkommunikation 9](#_Toc396338442)

[6.1.1 RS232 9](#_Toc396338443)

[6.1.2 TCP/IP Protokoll 9](#_Toc396338444)

[6.1.3 DLL 9](#_Toc396338445)

[6.2 Kommunikationsarchitektur 9](#_Toc396338446)

[6.3 Locsim 10](#_Toc396338447)

[6.3.1 DLL Spezifikation 10](#_Toc396338448)

# Kontaktliste

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Verankerung** | **Kontakt** | **Aufgaben / Hintergrund DESM** |
| Dr. Jürg Suter | DESM Präsident | +41 31 931 3662 | Präsident Verein DESM,  Middleware Standardisierung |
| Sebastian Straube | DESM Vorstand IT | +41 79 48 18 444 | Vorstand Verein DESM,  Middleware Standardisierung,  Implementierung für verknüpfte Systemkomponenten |
| Maximilian Haupt | Privat | mail@maximilianhaupt.com | Implementierung Middleware |
| Fabian Riesen | Cisco Systems | +41 79 448 4700 | Dispatcher Implementierung Re 4/4 |
| Dr. Hansjürg Rohrer | Fachhochschule Biel | +41 32 321 63 73 | Eigentümer Simulation LOCSIM |

# Dokumentenhistorie

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | | **Datum** | **Name** | **Änderung** |
| 0.12 | 06.02.2013 | Sebastian Straube | Kapitel hinzugefügt: 2 Kontaktliste, 3 Dokumentenhistorie  Kapitel entfernt: „Kontaktpersonen“  Kapitel erweitert: 7.7.5.3 Events Simulation Transition (Start, Aufbau und Ende) Beschreibung aktualisiert „setTrainPosition“ |
| 0.13 | 24.02.2013 | Sebastian Straube | Kapitel erweitert: 2 Kontaktliste, 7.7.1 ToDo Schnittstelle |
| 0.14 | 02.04.2013 22.07.2013 | Sebastian Straube | Kapitel 7.7.3 erweitert: stw\_infoConnectionStatus Kapitel 6 angepasst  Kapitel 7.8.3 angepasst |
| 0.15 | 11.01.2013 | Sebastian Straube | Kapitel 2 angepasst  Kapitel 7.7.5 angepasst |
| 0.16 | 19.01.2013 | Sebastian Straube | Kapitel 7.7.5 angepasst |
| 0.17 | 20.03.2014 | Sebastian Straube | Kapitel 7.5 angepasst  Kapitel 7.7.5 angepasst |
| 0.18 | 23.03.2014 | Sebastian Straube | Kapitel 7.7.5 angepasst |
| 0.2 | 20.08.2013 | Sebastian Straube | Strukturänderung |

# Ausgangslage

Innerhalb der Promotionsarbeit von Jürg Suter wurde ein Forschungslabor aufgebaut, welches zu einem offiziellen Verein mit dem Namen DESM institutionalisiert ist. Das Forschungslabor besteht momentan aus zwei Fahrsimulatoren Re 4/4 und Re 460. Der DESM Simulator des Typs Re460 ist gegenwärtig in der Schweiz der einzige Vollsimulator dieser Art. Für weitere Details zu der Promotionsarbeit verweise ich auf die Vereinsseite: <http://www.desm.ch> .

Alle Untersuchungen der Forschungsarbeit beziehen sich auf qualitative und quantitative Analysen und der Erarbeitung von neuen Methoden, die im Forschungslabor durchgeführt werden. Dafür ist es nötig ein System aufzubauen, in dem diese Methoden erarbeitet und die Ergebnisse wissenschaftlich analysiert werden können. Sie finden weitere Details auf die Vereinsseite: <http://www.desm.ch>.

# Ziel - integrierte Simulation

Für die anstehenden Aufgaben wird es nötig sein, die Kommunikation zwischen bestimmten Systemelementen zu ermöglichen. Das heisst, es soll nicht nur der Lokführer in die Simulation eingebunden werden, sondern auch der Zugverkehrsleiter in der Betriebszentrale sowie die dazugehörigen Stellwerke auf der simulierten Strecke.

Das Projekt Middleware verfolgt das langfristige Ziel, inkompatible Systemkomponenten einer Simulationsumgebung über eine standardisierte Kommunikationsarchitektur miteinander zu verbinden. Dieses Vorhaben wird innerhalb des Vereins DESM umgesetzt. Die einzelnen Komponenten der Architektur sollen möglichst modular aufgebaut sein, um die Anwendung und Integration verschiedener Komponenten kurzfristig zu ermöglichen. Die Kommunikation zwischen den Komponenten wird anhand eines anerkannten Industriestandards umgesetzt werden.

Die Simulationsumgebung mit allen beteiligten Komponenten wird in diesem Dokument integrierte Simulation genannt.

# Simulation

Die Middleware Architektur erlaubt es eine Simulationskomponente modular einzubinden. Für die Integration eines Modules ist es notwendig die Schnittstelle zwischen der Middleware und den Simulationskomponenten zu definieren. Es werden alle wichtigen Komponenten unterstützt, wie in der folgenden Abbildung gezeigt wird.

Die Art der Simulation wird in den Kategorien Umwelt, Fahrzeug, Kabine und Stellwerk gegliedert. In den meisten Simulations-umgebungen übernimmt eine spezialisierte Software eine bestimmte Art der Simulation. Die Implementierungen sind meist sehr proprietär auf das vorhandene Simulationsmodell zugeschnitten. Die Middleware löst den proprietären Ansatz und bildet dabei eine Schnittstelle zwischen den Simulationsarten und der Implementierung. Im ersten Schritt stehen uns die Simulation der Umwelt von Locsim und ZUSI, sowie für den Führerstand die Kabine der Re420 und der Re4/4, und das Obermatt-Langnau Stellwerk zur Verfügung. Die folgende Abbildung zeigt die Zusammenhänge zwischen der Simulationsart und der vorhanden Implementierung sowie deren Schnittstelle zur Middleware.

|  |
| --- |
| **DLL**  **RS232**  **DESM Middleware**  Fahrzeug  LOCSIM  LOCSIM  Umwelt  ZUSI  Umwelt/ Fahrzeug  **TCP/IP**  Re 420  Kabine  **RS232**  Re 4/4  Kabine  **RS232**  **RS232**  Stellwerk  ObermattLangnau |

## Systemkommunikation

### RS232

### TCP/IP Protokoll

### DLL

## Kommunikationsarchitektur

Es stehen verschiedene Techniken zur Verfügung, die Daten über ein Protokoll auszutauschen. Es wurden folgende Technologien bzw. Frameworks evaluiert.

|  |  |  |  |
| --- | --- | --- | --- |
| **Protokollname** | **Beschreibung** | **Vorteil** | **Nachteil** |
| Winsocket ([TCP](http://www.codeproject.com/Articles/13071/Programming-Windows-TCP-Sockets-in-C-for-the-Begin)) | Eine „Winsocket“ Client-Server Übertragung. Das Übertragungsprotokoll soll lediglich ByteStreams empfangen und senden können, weil nur XML Daten transferiert werden sollen und keine RAW Datentypen. Dafür muss ein Magic Packet (Initialisierung) definiert werden. | * schnell * Abstraktion durch XML Struktur | * Integration kompliziert * eigenes Übertragungsprotokoll nötig |
| HTTP (high level) & [Json](http://www.json.org/json-de.html) | verschiedene libraries sind u.A. boostASIO, libHTTP, libEvent, libOV | * High Level Integration * ermöglich Steuerung über Webinterface | * Overhead gross * keine permanente Verbindung * zwei Webserver |
| [Open Sound Protokoll](http://gruntthepeon.free.fr/oscpkt/html/oscpkt__demo_8cc-example.html) |  | * library Unterstützung sehr umfangreich | * proprietäre Implementierung |
| ZeroMQ (0MQ) |  | * Framework Message Handling * Portierung auf andere Sprachen möglich |  |

## Locsim

Simulationssoftware Locsim wird von der FH Biel in der Schweiz entwickelt. Die Verbindung zwischen der Middleware und dem LOCSIM wird per Ethernet und Rs232 Verbindung hergestellt. Aus den gegebenen Anforderungen der Middleware ist eine modulare Lösung für die Implementierung der Anbindung nötig.

### Schnittstelle RS232

### Schnittstelle DLL

Die DLL wurde als „unmanaged“ Code in C und C++ geschrieben. Zum Laden der DLL wird kein MFC benötigt. Auf die Definition als COM Komponente wurde aus Vereinfachungsgründen verzichtet. Die Daten in der DLL werden in einem Cache gehalten und von dort weitergegeben oder abgeholt.

#### Präfix Definition

Es ist für programmatische Problemstellungen u.U. wichtig zu erkennen, in welcher Situation eine Funktion benutzt werden sollte um allen Anforderungen gerecht zu werden. Daher werden hier verschiedene Präfixe für Funktionen definiert, damit bereits vom Namen abgeleitet werden kann in welche Richtung der Kommunikationsweg vollzogen wird und ob z.B. der Cache beeinflusst wird. Die folgende Tabelle.

|  |  |
| --- | --- |
| **Präfix** | **Beschreibung** |
| stw\_set | Daten aus LOCSIM zum Stellwerk übertragen |
| stw\_get | Daten aus Stellwerk zum LOCSIM übertragen |
| stw\_on | Funktionsaufruf während bestimmter Events |
| stw\_info | DLL Informationen |

#### Events und Funktionsaufrufe LOCSIM

Die folgende Tabelle zeigt eine Übersicht bei welchem LOCSIM Event bestimmte Funktionen aufgerufen werden sollten. Die Übersicht beschränkt sich auf genau eine Instanz des LOCSIM.

|  |  |  |
| --- | --- | --- |
| **Event** | **Funktionsaufruf Anzahl** | **Aktion** |
| Start Programm | einmalig  beliebig | stw\_onStartProgramm  stw\_infoVersion  stw\_infoConnectionStatus |
| Lade Strecke  (Lade neue Strecke) | Beliebig  beliebig  beliebig  beliebig  beliebig  beliebig  beliebig  beliebig | stw\_onLoadStrecke  stw\_setTrack  stw\_setTrackConnection  stw\_setSignal  stw\_setBalise  stw\_setIsolierstoss  stw\_setKilometerDirection  stw\_setLoop |
| Start Simulation | Einmalig  beliebig  beliebig  beliebig  beliebig  beliebig  beliebig | stw\_onStartSimulation  stw\_getEvents  stw\_getSignal  stw\_getBalise  stw\_getWeiche  stw\_getLoop  stw\_setTrainPosition |
| Stopp Simulation | Einmalig | stw\_onStopSimulation |
| Stopp Programm | Einmalig | stw\_onStopProgramm |

#### DLL Error

|  |  |
| --- | --- |
| **Error Code** | **Beschreibung** |
| 0 | ERROR\_OK |
| 1 | ERROR\_FATAL |
| 2 | ERROR\_API\_MISUSE |
| 3 | ERROR\_UNKNOWN\_ID |

#### Event DLL laden

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_onStartProgramm (char\* configPath) | |
| **Funktionsbeschreibung** | * beim Laden der DLL wird der Netzwerkserver gestartet | |
| **seit Version** | 0.1 | |
| **Parameter** | configPath | der Pfad zum gemeinsamen Konfigurationsverzeichnisses wo die Konfigurationsdatei abgelegt ist |
| **Rückgabepointer** |  |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 1 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_onStopProgramm (void) | |
| **Funktionsbeschreibung** | * Wenn LOCSIM beendet wird ist die Simulation gestoppt, die DLL wird entladen und der Netzwerkserver heruntergefahren. Es wird sichergestellt, dass die Verbindung zwischen Server und Client ordnungsgemäss getrennt wird. | |
| **seit Version** | 0.1 | |
| **Parameter** |  |  |
| **Rückgabepointer** |  |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** |  |  |

#### Event DLL ist geladen

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | const int stw\_infoVersion(char\* versionBuf, int versionBufLen, int\* versionStrLen) | |
| **Funktionsbeschreibung** | * Version der DLL, zur Behandlung von Versionskonflikten | |
| **seit Version** | 0.1 | |
| **Parameter** | int versionBufLen | Die Anzahl der möglichen Zeichen |
| **Rückgabepointer** | char\* versionBuf  int\* versionStrLen | Pointer auf die Zeichenkette  Pointer für die Länge der Zeichenkette |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** |  |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | const char\* stw\_infoConnectionStatus(char\* statusBuf, int statusBufLen, int\* statusStrLen) | |
| **Funktionsbeschreibung** | * gibt den Status der Netzwerkverbindung von Middleware Server und Client zurück | |
| **seit Version** | 0.1 | |
| **Parameter** | int statusBufLen | Die Anzahl der möglichen Zeichen |
| **Rückgabepointer** | char\* statusBuf  int\* statusStrLen | Pointer auf die Zeichenkette  Pointer für die Länge der Zeichenkette |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** |  |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | const char\* stw\_infoName(char\* nameBuf, int nameBufLen, int\* nameStrLen) | |
| **Funktionsbeschreibung** | * gibt die Bezeichnung der DLL zurück | |
| **seit Version** | 0.1 | |
| **Parameter** | infoNameLen | Die Anzahl der Zeichen vom Rückgabewert als Pointer |
| **Rückgabepointer** | char\* nameBuf  int\* nameStrLen | Pointer auf die Zeichenkette  Pointer für die Länge der Zeichenkette |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** |  |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | const char\* stw\_infoDescription(char\* descBuf, int descBufLen, int\* descStrLen) | |
| **Funktionsbeschreibung** | * gibt die Beschreibung der DLL zurück | |
| **seit Version** | 0.1 | |
| **Parameter** | descriptionLen | Die Anzahl der Zeichen vom Rückgabewert als Pointer |
| **Rückgabepointer** | char\* descBuf  int\* descStrLen | Pointer auf die Zeichenkette  Pointer für die Länge der Zeichenkette |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** |  |  |

#### Events Simulation Transition (Start, Aufbau und Ende)

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_onStartSimulation (void) | |
| **Funktionsbeschreibung** | * markiert den Start der Simulation * die weitere Datenverarbeitung wird durch mögliche „sets“ ermöglicht * Nach dem Aufruf dieser Funktion ist das dirty flag sämtlicher Signale zu setzen, damit die Grundstellung von LOCSIM erkannt wird | |
| **seit Version** | 0.1 | |
| **Parameter** |  |  |
| **Rückgabepointer** |  |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_onStopSimulation (void) | |
| **Funktionsbeschreibung** | * markiert das Ende der Simulation * erwartet danach „get“ Funktionen oder die „setTrainPosition“ Funktion | |
| **seit Version** | 0.1 | |
| **Parameter** |  |  |
| **Rückgabepointer** |  |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_onLoadStrecke (void) | |
| **Funktionsbeschreibung** | * beim erneuten Laden einer Strecke werden alle Transferdaten in der Stellwerk DLL gelöscht | |
| **seit Version** | 0.1 | |
| **Attribute** |  |  |
| **Rückgabepointer** |  |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_setTrack (int gleisId, double von, double bis, double abstand, char\* name, int nameLen) | |
| **Funktionsbeschreibung** | * Definition von zu Hauptgleis parallelen Gleisabschnitten * das Hauptgleis ist die Standardfahrstrasse der Simulation | |
| **seit Version** | 0.1 | |
| **Parameter** | gleisId  von  bis  abstand  name  nameLen | locsim-interne ID  als Meterangabe  als Meterangabe  Abstand von Hauptgleis( in Meter)  Name vom Track  Die Anzahl der Zeichen vom Attribut name |
| **Rückgabepointer** |  |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_getTrack (int gleisId, double\* von, double\* bis, double\* abstand, char\* nameBuf, int nameBufLen, int\* nameStrLen) | |
| **Funktionsbeschreibung** | * Definition von zu Hauptgleis parallelen Gleisabschnitten * das Hauptgleis ist die Standardfahrstrasse der Simulation | |
| **seit Version** | 0.1 | |
| **Parameter** | gleisId  von  bis  abstand  nameBuf  nameBufLen  nameStrLen | locsim-interne ID  als Meterangabe  als Meterangabe  Abstand von Hauptgleis( in Meter)  positiv (rechts), negativ (links), Gleisnummer gemäss Sicherungsanlage, z.B. A1 (kann auch leer sein)  Buffer Länge von nameBuf  die Anzahl der Zeichen von der Variable nameBuf (exkl. Terminator String) |
| **Rückgabepointer** | double\* von  double\* bis  double \*abstand  char\* nameBuf  int\* nameStrLen |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_setTrackConnection (int gleisBasisId, int gleis1Id, int gleis2Id,  double von, double bis, char\* name, int nameLen, int weiche1Id, int weiche2Id) | |
| **Funktionsbeschreibung** | * definiert die Verbindungen zwischen parallelen Gleisabschnitten * weiche1Id, weiche2Id = ID der Weiche * ID ist eine eindeutige LOCSIM-interne Nummerierung * wenn ID=0 dann keine Weiche (wenn z.B. connection2 in gleis2 übergeht) * pro Längsposition dürfen max. 20 verschiedene Gleise definiert sein (ID= 1…20) * Die Gleisnummer (ID) definiert eine Gleisverbindung ist wie ein Gleis (mit ID), aber anstelle eines definierten Abstandes zum Basisgleis geht sie von einem Gleis (gleis1id) zu einem anderen Gleis (gleis2id). Wenn ID=gleis1id oder ID=gleis2id hat es dort keine Weiche, d.h. es ist eine „gebogene“ Verbindung zu einem Gleis mit derselben Gleisnummer. Ein „Gleis“ ist bei uns immer parallel zum Basisgleis. Das brauchen zwingend auch, wenn sie einen Stellwerk-Plan zeichnen müssen. | |
| **seit Version** | 0.1 | |
| **Parameter** | gleisBasisId  gleis1Id  gleis2Id  von  bis  name  nameLen  weiche1Id  weiche2Id | Definiert eine gebogene Verbindung wenn id = gleis1Id oder gleis2Id  Eindeutige id vom Gleis1  Eindeutige id vom Gleis2  Positionsangabe in Meter  Positionsangabe in Meter  Name vom Gleis  Die Anzahl der Zeichen vom Attribut name  ID der ersten Weiche  ID der zweiten Weiche |
| **Rückgabepointer** |  |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_getTrackConnection (int\* gleisBasisId, int gleis1Id, int gleis2Id,  double\* von, double\* bis, char\* nameBuf, int nameBufLen, int\* nameStrLen, int\* weiche1Id, int\* weiche2Id) | |
| **Funktionsbeschreibung** | * definiert die Verbindungen zwischen parallelen Gleisabschnitten * weiche1Id, weiche2Id = ID der Weiche * ID ist eine eindeutige LOCSIM-interne Nummerierung * pro Längsposition dürfen max. 20 verschiedene Gleise definiert sein (ID= 1…20) * Die Gleisnummer (ID) definiert eine Gleisverbindung ist wie ein Gleis (mit ID), aber anstelle eines definierten Abstandes zum Basisgleis geht sie von einem Gleis (gleis1id) zu einem anderen Gleis (gleis2id). Wenn ID=gleis1id oder ID=gleis2id hat es dort keine Weiche, d.h. es ist eine „gebogene“ Verbindung zu einem Gleis mit derselben Gleisnummer. Ein „Gleis“ ist bei uns immer parallel zum Basisgleis. Das brauchen zwingend auch, wenn sie einen Stellwerk-Plan zeichnen müssen. | |
| **seit Version** | 0.1 | |
| **Parameter** | gleisBasisId  gleis1Id  gleis2Id  von  bis  nameBuf  nameBufLen  nameStrLen  weiche1Id  weiche2Id | Definiert eine gebogene Verbindung wenn id = gleis1Id oder gleis2Id  Eindeutige id vom Gleis1  Eindeutige id vom Gleis2  Positionsangabe in meter  Positionsangabe in meter  Name der Gleisverbindung  Grösse des Buffers zur Übertragung  Die Anzahl der Zeichen vom Attribut name  ID der ersten Weiche  ID der zweiten Weiche |
| **Rückgabepointer** | glaisBasisId  von  bis  nameBuf  nameStrLen  weiche1Id  weiche2Id |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

Id gleis ort signaltyp höhe distanz name namelen wirkrichtung

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_setSignal (int signalId, char\* name, int stellung) | |
| **Funktionsbeschreibung** | * wird pro Signal aufgerufen | |
| **seit Version** | 0.1 | |
| **Parameter** | signalId  name  stellung | eindeutige Signal Id  name des Signals  positiv dann Richtung +1, negativ dann Richtung -1 |
| **Rückgabepointer** |  |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_setBalise (int baliseId, int gleisId,  double position, int stellung, int beeinflussendeSignalId1, int beeinflussendeSignalId2) | |
| **Funktionsbeschreibung** | * definiert eine Balise: auf bestimmten Gleis, an bestimmter Position | |
| **seit Version** | 0.1 | |
| **Parameter** | baliseId  gleisId  position  stellung  beeinflussendeSignalId1 beeinflussendeSignalId2 | eindeutige Balise ID  eindeutige Gleis ID  Position der Balise  Stellung positiv dann Richtung +1 sonst negativ dann Richtung -1  Beeinflussende Signal 1  Beeinflussende Signal 2 |
| **Rückgabewert (Attribut)** |  |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_setLoop (int baliseId, int gleisId,  double positionVon, double positionBis, int stellung, int beeinflussendeSignalId1, int beeinflussendeSignalId2) | |
| **Funktionsbeschreibung** | * ein Loop ist eine Antenne, die wie eine Balise wirkt, aber nicht punktuell, sondern über eine bestimmte Länge (von-bis) | |
| **seit Version** | 0.13 | |
| **Parameter** | baliseId  stellung  char\* protokoll  int protokollLen | eindeutige Balise ID  Stellung positiv dann Richtung +1 sonst negativ dann Richtung -1  Protokoll – welches Protokoll wird hier verwendet?  Die Anzahl der Zeichen vom Attribut protokoll |
| **Rückgabewert (Attribut)** |  |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_setIsolierstoss (int isolierstossId, int gleisId, double position) | |
| **Funktionsbeschreibung** | * Ist in den locsim-Streckendaten bis jetzt nicht drin, könnte aber hinzugefügt werden | |
| **seit Version** | 0.1 | |
| **Parameter** | isolierstossId  gleisId  position | Eindeutige Isolierstoss id  Eindeutige Gleis id  Position vom Isolierstoss |
| **Rückgabepointer** |  |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_getIsolierstoss (int isolierstossId, int\* gleisId, double\* position) | |
| **Funktionsbeschreibung** | * Ist in den locsim-Streckendaten bis jetzt nicht drin, könnte aber hinzugefügt werden | |
| **seit Version** | 0.15 | |
| **Parameter** | isolierstossId  gleisId  position | Eindeutige Isolierstoss id  Eindeutige Gleis id  Position vom Isolierstoss |
| **Rückgabepointer** | Int\* gleisId  double\* position |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_setKilometerDirection (int richtung) | |
| **Funktionsbeschreibung** | * Gibt an ob die Kilometer inkrementiert oder dekrementiert werden | |
| **seit Version** | 0.13 | |
| **Parameter** | Richtung | bei einem positiven Wert wird inkrementiert (+1)  bei einem negativen Wert wird dekrementiert (-1) |
| **Rückgabepointer** |  |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_getKilometerDirection (int\* richtung) | |
| **Funktionsbeschreibung** | * Gibt an ob die Kilometer inkrementiert oder dekrementiert werden | |
| **seit Version** | 0.15 | |
| **Parameter** | Richtung | bei einem positiven Wert wird inkrementiert (+1)  bei einem negativen Wert wird dekrementiert (-1) |
| **Rückgabepointer** | int\* richtung |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

#### Event Simulation gestartet

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_getEvents(int\* anzahlEvents, int\*\* typeList, int\*\*\* paramList) | |
| **Funktionsbeschreibung** | * LOCSIM fragt DLL, welche events vom Stellwerk ausgelöst wurden * Beispiel aufruf:   anzahlEvents = 2;  typeList = {1,2};  paramList = {{SignalId}, {WeicheID}}. | |
| **seit Version** | 0.1 | |
| **Parameter** | anzahlEvents  typeList  paramList | Anzahl Events (Arraygrösse): 0 = keine Events vorhanden  Event typen sind 1 (Signal), 2 (Balise), 3 (Loop) und 4 (Weiche)  Eine Liste mit IDs zu der korrespondierenden typeList, siehe Funktionsbeschreibung |
| **Rückgabepointer** | Int\* anzahlEvents  Int\*\* typeList  Int\*\*\* paramList | Beinhaltet die Anzahl der Events (Buffer länge) |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 1  2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_getSignal (int signalId, char\*\* nameBuf, int nameBufLen, int\* nameStrLen, int\* stellung) | |
| **Funktionsbeschreibung** | * gibt die Stellung eines bestimmten Signals zurück | |
| **seit Version** | 0.1 | |
| **Parameter** | signalId  nameBuf  nameBufLen  nameStrLen  stellung | ID des Signals  Eindeutiger Name vom Signal  grösse des Buffers zur Übertragung  Die Anzahl der Zeichen vom Attribut name |
| **Rückgabepointer** | Int\* gleisId  char\* nameBuf  int\* nameStrLen  int\* stellung |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_setWeiche (int weicheId, int gleisId) | |
| **Funktionsbeschreibung** | * gibt die Stellung einer bestimmten Weiche zurück | |
| **seit Version** | 0.1 | |
| **Parameter** | weicheId  gleisId | ID gemäss set\_trackConnection  Gleisnummer der Stellung (im stumpfen Bereich) |
| **Rückgabepointer** |  |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_getWeiche (int weicheId, int\* gleisId) | |
| **Funktionsbeschreibung** | * gibt die Stellung einer bestimmten Weiche zurück | |
| **seit Version** | 0.1 | |
| **Parameter** | weicheId  gleisId | ID gemäss set\_trackConnection  Gleisnummer der Stellung (im stumpfen Bereich) |
| **Rückgabepointer** | int\* gleisId | ist rückgabewert, als Integer Pointer |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_setTrainPosition (int trainTyp, int direction, double\* positionList, int positionListLen, int\* gleisList, int gleisListLen) | |
| **Funktionsbeschreibung** | * Übergibt pos1 – gleis1 – pos2 – gleis2 – pos3 - … des Zuges * Zug an DLL wenn die Zugspitze oder der Zugschluss einen Isolierstoss überfährt * Was bedeutet in diesem Fall die Position – Zuganfang, Zugschluss oder Achse? | |
| **seit Version** | 0.1 | |
| **Parameter** | trainTyp  direction  positionList  poitionListLen  gleisList  gleisListLen | 0=simulierter Zug, 1 und weitere=andere Züge  1=vorwärts, -1=rückwärts  Position „von, bis“ als Objekt von „setTrack“  Gleisnummern „gleisId“ als Objekt von „setTrack“  Anzahl der Elemente in gleisList |
| **Rückgabepointer** |  |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_getTrainPosition (int trainTyp, int\* direction,  double\*\* positionList, int\* positionListLen, int\*\* gleisList, int\* gleisListLen) | |
| **Funktionsbeschreibung** | * Übergibt pos1 – gleis1 – pos2 – gleis2 – pos3 - … des Zuges * Zug an DLL wenn die Zugspitze oder der Zugschluss einen Isolierstoss überfährt * Was bedeutet in diesem Fall die Position – Zuganfang, Zugschluss oder Achse? | |
| **seit Version** | 0.1 | |
| **Parameter** | trainTyp  direction  positionList  positionListLen  gleisList  gleisListLen | 0=simulierter Zug, 1 und weitere=andere Züge  1=vorwärts, -1=rückwärts  Position „von, bis“ als Objekt von „setTrack“    Gleisnummern „gleisId“ als Objekt von „setTrack“ |
| **Rückgabepointer** | int\* direction  double\*\* positionList  int\* positionListLen  int\*\* gleisList  int\* gleisListLen |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_getLoop (int baliseId, int\* stellung, char\* protokollBuf, int protokollBufLen, int\* protokollStrLen)) | |
| **Funktionsbeschreibung** | * ein Loop ist eine Antenne, die wie eine Balise wirkt, aber nicht punktuell, sondern über eine bestimmte Länge (von-bis) | |
| **seit Version** | 0.15 | |
| **Parameter** | baliseId  stellung  protokollBuf  protokollBufLen  protokollStrLen | eindeutige Balise ID  Stellung positiv dann Richtung +1 sonst negativ dann Richtung -1 |
| **Rückgabepointer** | int\* gleisId  double\* von  double\* bis  int\* stellung  int\* beeinflussendeSignalId1  int\* beeinflussendeSignalId2 |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Signatur** | int stw\_getBalise (int baliseId, int\* stellung, char\* protokollBuf, int protokollBufLen, int\* protokollStrLen)) | |
| **Funktionsbeschreibung** | gibt eine Stellung und das Protokoll einer bestimmten Balise zurück | |
| **seit Version** | 0.1 | |
| **Parameter** | baliseId  stellung  protokollBuf  protokollBufLen  protokollStrLen | eindeutige Balise ID  Stellung positiv dann Richtung +1 sonst negativ dann Richtung -1 |
| **Rückgabepointer** | int\* stellung  char\* protokollBuf  int\* protokollStrLen |  |
| **Rückgabewert (OK)** | 0 |  |
| **Rückgabewert (ERROR)** | 2 |  |