

**Model Animation**

**Server**



|  |  |  |  |
| --- | --- | --- | --- |
| Version | Datum | Autor | Kommentar |
| v0.1 | 15.11.2015 | Grischa Rutishauser | Erstellung |
| V0.2 | 04.01.2016 | Grischa Rutishauser | elmStartType, elmEndType |

|  |  |
| --- | --- |
|  | Actifsource Architecture |
| Notation | * To do * Information * **Bold**: Terms from actifsource or other technologies and tools * **Bold underlined**: actifsource Resources * Underlined: User Resources * *UnderlinedItalics*: Resource Functions * Monospaced: User input * *Italics*: Important terms in current situation |
| Disclaimer | The authors do not accept any liability arising out of the application or use of any information or equipment described herein. The information contained within this document is by its very nature incomplete. Therefore the authors accept no responsibility for the precise accuracy of the documentation contained herein. It should be used rather as a guide and starting point. |
| Contact | **actifsource GmbH**  Täfernstrasse 37  5405 Baden-Dättwil  Switzerland  [www.actifsource.com](http://www.actifsource.com) |
| Trademark | **actifsource** is a registered trademark of **actifsource GmbH** in Switzerland, the EU, USA, and China. Other names appearing on the site may be trademarks of their respective owners. |
| Revision | * 2014/01/30 [rc] 5.11.0 |

Table of Content

[1 JSON Schnittstellendefinition Server ⬄ Target 4](#_Toc443751968)

[1.1 Involvierte Komponenten 4](#_Toc443751969)

[1.1.1 Modul CIP 5](#_Toc443751970)

[1.1.2 Modul DEC 6](#_Toc443751971)

[1.1.3 Target Ring-Buffer 6](#_Toc443751972)

[1.1.4 Target Router 6](#_Toc443751973)

[1.1.5 PC Server 6](#_Toc443751974)

[1.1.6 PC Browser 6](#_Toc443751975)

[1.2 JSON Telegramme zwischen Server ⬄ Target 7](#_Toc443751976)

[1.2.1 Bind-Request 7](#_Toc443751977)

[1.2.2 Unbind-Request 7](#_Toc443751978)

[1.2.3 BindValue-Request 7](#_Toc443751979)

[1.2.4 UnbindValue-Request 7](#_Toc443751980)

[1.2.5 Action-Request 8](#_Toc443751981)

[1.2.6 Update-Response 9](#_Toc443751982)

# JSON Schnittstellendefinition Server ⬄ Target

In diesem Kapitel werden die Schnittstellen zwischen Server und Target definiert. Diese Schnittstellen werden für CIP-Animationen (Trumpf) verwendet.

## Involvierte Komponenten



Abbildung 4 Involvierte Komponenten

In den folgenden Kapiteln werden die einzelnen Komponenten und deren Verhalten genauer beschrieben.

### Modul CIP

Das CIP-System hat die Aufgabe alle Änderungen der zu animierenden Elemente (State oder Transition) dem Ring-Buffer mitzuteilen.

Jedes animierte Element (z.B: {"id":[2,4,2,8],"st":1, seq":12}) besitzt folgende Informationen:

|  |  |
| --- | --- |
| ID:  (id) | * Jedes dieser Elemente ist über eine eindeutige ID innerhalb des gesamten Systems (über alle Module) identifizierbar. Diese ID setzt sich aus den folgenden Sub-IDs zusammen:  1. Sub-ID1 -> Modul\_ID 2. Sub-ID2 -> Process\_ID 3. Sub-ID2 -> MultiplicityProcessIndex\_ID 4. Sub-ID4 -> Element\_ID  * Der Sub-ID Wert „0“ dient als Broadcast Adresse für die entsprechende Stufe. Dieser wird z.B. für einen Mode-Change verwendet. |
| State:  (st) | * *How the element should be displayed.* |
| Time:  (time) | * Der Zeitstempel könnte für die Synchronisation zwischen den Modulen verwendet werden, da sich der Undo-Kontext auf ein Modul bezieht. |
| Sequence:  (seq) | * Die Sequence-Number ist eindeutig innerhalb eines Moduls. Der Undo-Kontext bezieht sich auf ein Modul. |

*Information eines Elements*

Über die Funktion „mvaSetElementState“ wird der Ring-Buffer vom CIP-System benachrichtigt. Alle übergebenen Elemente besitzen die gleiche Sequence-Number. Nach jedem Aufruf dieser Funktion, muss die Sequence inkrementiert werden.

|  |
| --- |
| mvaSetElementState(unsigned char elementCount, t\_AnimationElement elements[]); |

*Cip Function-Call.*

|  |
| --- |
| typedef struct {  unsigned int processId; // Prozess-ID, eindeutig innerhalb des Moduls  unsigned char multiplicityIndex; // Array-Index des Prozesses  unsigned int elementId; // Element-ID, eindeutig innerhalb des Prozesses (State oder Transition) unsigned char state; // State des Elementes (z.B: Rot oder Grün…)  } t\_AnimationElement; |

*Element-Struct*

Über die Funktion „mvaGetProcessSnapshot“ kann der gesamte Snapshot aus dem System abgefragt werden.

|  |
| --- |
| mvaGetSystemSnapshot( t\_AnimationElement \*processesElemente); |

*Cip System-Snapshot.*

### Modul DEC

Über die DEC-Komponente werden nun die Animation Elemente an das Bussystem weitergeleitet. Ankommende Meldungen werden entsprechend interpretiert.

### Target Ring-Buffer

Der Ring-Buffer dient als History-Speicher, um in der Zeit zurück zu gehen.

### Target Router

Der Router hat die Aufgabe ankommende JSON Telegramme (PC -> Router) in ein Proprietäres Protokoll umzuwandeln und an das richtigen Modul weiterzuleiten. Das gleiche gilt für die Gegenrichtung.

### PC Server

Der Server wird in C oder Java geschrieben. Er hat die Aufgabe Informationen, welche vom Browser ([2 Kommunikation Server <-> Browser](#_JSON_Schnittstellendefinition_Brows)) benötigt werden, beim Modul anzufordern und zwischen zu speichern.

### PC Browser

[1 Design der Benutzeroberfläche](#_Design_der_Benutzeroberfläche)

## JSON Telegramme zwischen Server ⬄ Target

Dieses Kapitel beinhaltet alle JSON Telegramme welche zwischen Server (PC) und Target (Modul) ausgetauscht werden.

### Bind-Request

Registriert der Browser beim Server eine neue statusbehaftete Komponente (SK). Leitet der Server diese Anfrage an das entsprechende Modul weiter, solange dieses Modul noch nicht registriert wurde. (Die „id“ im JSON Bind-Request entspricht der Modul-ID).

|  |
| --- |
| {  "bind":[  {"id":[4]}  ]  } |

*JSON Bind-Request*

### Unbind-Request

Wird ein Modul (alle SKs eines Moduls) vom Browser nicht mehr benötigt, wird dieses Modul über den Unbind-Request abgemeldet.

|  |
| --- |
| {  "unbind":[  {"id":[4]}  ]  } |

*JSON Unbind-Request*

### CRC- Request

Der Browser hat die Möglichkeit den Application CRC-Wert beim Server anzufordern. Diese Anfrage erfolgt über WebSocket.

|  |
| --- |
| {  "crc":"get", "id":[4]  } |

*JSON Anfrage get crc*

### CRC-Response

Anhand der Informationen im CRC-Response wird der Browser nun überprüft, ob der Browser und der Server synchron sind.

|  |
| --- |
| {  "crc":14345345435, "id":[4]  } |

*JSON CRC-Response*

### BindValue-Request

Über diesen Request kann der Browser Value-Daten beim Modul anfordern. (Die „id“ im JSON-Request entspricht der Modul-ID und Value-ID). (time entspricht der periode)

|  |
| --- |
| {  "bindValue":[  {"id":[4,2],"cycletime":55}  ]  } |

*JSON BindValue-Request*

### UnbindValue-Request

Benötigt der Browser den Value nicht mehr, wird dieser über den UnbindValue-Request beim Modul abgemeldet.

|  |
| --- |
| {  "unbindValue":[  {"id":[4,2]}  ]  } |

*JSON UnbindValue-Request*

### Action-Request

Über den Browser können Action an das Target abgesendet werden. (Die „id“ im JSON-Request entspricht der Modul-ID und Action-ID).

|  |
| --- |
| {  "action":[  {"id":[4,3], "value":0}  ]  } |

*JSON Action-Request*

### Tree-Request

Die dynamischen Komponenten des Naviagtion-Baums werden über WebSocket automatisch nach dem Erhalten des explorer-data.json Files vom Browser angefordert (Tree-Request). Dieser Request erfolgt nur einmal.

|  |
| --- |
| {  "tree":"get"  } |

*JSON Tree-Request*

### Tree-Response

Dieses Telegramm erfolgt als Respons auf einen Tree-Request. Es enthält alle dynamischen Tree-Nodes, welche von der Default-Initialisierung abweichen. Dieses Telegramm kann zu jedem Zeitpunkt beim Browser eintreffen, wodurch der Browser die Änderungen aktualisiert.

|  |
| --- |
| {  "tree":[  {"id":[“aa9ad4c4-1424-11e5-8041-a1180c924d9f”, 4],"st":0,"name":"NewName"},  {"id":[“aa9ad4c4-1424-11e5-8042-a1180c924d9f”, 5],"st":1}  ]  } |

*JSON Tree-Node Config*

### Update-Response

Wird eine Bind-Request an ein Modul gesendet, leitet dieses Modul folgende Schritte ein:

1. Als erstes werden alle History-Elemente gesendet vom ältesten zum neusten (elmStartType: history).
2. Danach folgt der Snapshot des Systems. Es müssen nur jene Elemente übermittelt werden, welche vom Init-Wert abweichen. Diese Elemente besitzen keine Sequence-Number (elmStartType: snapshot).
3. Nach dem Snapshot folgen alle zukünftigen Ereignisse (elmStartType: update).
4. Dies wird solange fortgesetzt bis der Unbind-Request eintrifft.
5. Trifft nach einem Unbind-Request wieder ein Bind-Request ein, beginnt das Ganze von vorne.

Die Übermittlung kann in unterschiedliche Telegramme unterteilt sein. Allerdings darf die Reihenfolge der Elemente auf dem Weg zum Server nicht verändert werden.

|  |
| --- |
| {  "elmStartType":{"id":[4],"type":"history"},  "elm":[  {"id":[4,2,3,1],"st":1,"seq":220},  {"id":[4,2,3,3],"st":0,"seq":219},  {"id":[4,2,3,4],"st":0,"seq":115},  ],  "value":[  {"id":[4,1],"st":1,"value":1},  {"id":[4,2],"st":1,"value":9}  ],  "error":[  {"nr":1,"msg":"Error message"}  ],  "log":[  {"time":16666, "st": 1,"msg":"Test Message"}  ]  } |

*JSON Update-Response*

|  |
| --- |
| {  "elmStartType":{"id":[4],"type":"snapshot"},  "elm":[  {"id":[4,2,3,1],"st":1,"seq":230},  {"id":[4,2,3,3],"st":0,"seq":230},  {"id":[4,2,3,2],"st":1,"seq":230},  {"id":[4,2,3,4],"st":0,"seq":230}  ],  } |

*JSON Update-Response*

Auch möglich:

|  |
| --- |
| {  "elmStartType":{"id":[4],"type":"update"},  "elm":[  {"id":[4,2,3,1],"st":1,"seq":230},  {"id":[4,2,3,3],"st":0,"seq":230},  {"id":[4,2,3,2],"st":1,"seq":230},  {"id":[4,2,3,4],"st":0,"seq":230}  ],  }  **--New Json Object--**  {  "elm":[  {"id":[4,2,3,5],"st":1,"seq":230},  {"id":[4,2,3,6],"st":0,"seq":230},  {"id":[4,2,3,7],"st":1,"seq":230},  {"id":[4,2,3,8],"st":0,"seq":230}  ],  }  **--New Json Object--**  {  "elm":[  {"id":[4,2,3,11],"st":1,"seq":230},  {"id":[4,2,3,13],"st":0,"seq":230},  {"id":[4,2,3,12],"st":1,"seq":230},  {"id":[4,2,3,10],"st":0,"seq":230}  ],  } |

*JSON Update-Response*

In der gleichen Update-Response können Elemente, Values und Errors übermittelt werden.

### Test-Bench

|  |
| --- |
| {  "testbenchState":"reset "  } |

*JSON –Request*

Browser -> Server

|  |
| --- |
| {  "testbenchEvent":{"id":[3], "value":0}  } |

*JSON Test Case Event*

Server -> Browser

|  |
| --- |
| {  "testbenchAction": {"id":[2], "value":{"f1":44, "f2":66, "f3":{"f1":88}}}  } |

*JSON Test Case Action*

Complete : reset and Event

|  |
| --- |
| {  " testbenchState ":"complete"  } |

*JSON –Response (Event )*

Server -> Browser

|  |
| --- |
| {  "requiredInput": {"id":[3]}  } |

*JSON Test Case Action*

Browser -> Server

|  |
| --- |
| {  "requiredOutput":{"id":[2], "value":{"f1":44, "f2":66, "f3":{"f1":88}}}  } |

*JSON Test Case Event*