







## Inhaltsverzeichnis

<b>1</b>	<b>Allgemeines</b>	<b>3</b>
<b>2</b>	<b>Scripte</b>	<b>4</b>
2.1	Syntax	4
2.2	Operatoren:	5
2.3	Lokale Variablen :	5
<b>3</b>	<b>Objektaufistung</b>	<b>5</b>
<b>4</b>	<b>Komponenten</b>	<b>7</b>
4.1	Kommunikationskomponente (TATSPLC) 	7
4.1.1	Eigenschaften von TATSPLC (properties)	7
4.2	Programmlader (TATSProgLoader) 	8
4.2.1	Eigenschaften von TATSProgLoader (properties)	8
4.3	Fehlermeldung (TATSError) 	10
4.3.1	Eigenschaften von TATSError (properties)	11
4.4	Hinweise (TATSHint) 	11
4.4.1	Eigenschaften von TATSHint (properties)	12
4.5	Programmierbare Ein/Ausgänge (TATSProgEAs) 	12
4.5.1	Eigenschaften von TATSProgEAs (properties)	12
4.6	Schrittendebedingungen (TATSEndOfStepCond) 	13
4.6.1	Eigenschaften von TATSEndOfStepCond (properties)	13



<b>4.7</b>	<b>Makros für Schrittketten (TATSProgMakro)</b> 	<b>13</b>
4.7.1	Eigenschaften von TATSProgMakro (properties)	14
<b>4.8</b>	<b>Zeit und Scriptmanagement (TATSTimerScript)</b> 	<b>15</b>
4.8.1	Eigenschaften von TATSTimerScript (properties)	15
<b>4.9</b>	<b>Datenprotokollierung (TATSDataLogger)</b> 	<b>15</b>
4.9.1	Eigenschaften von TATSDatalogger (properties)	16



# 1 Allgemeines

VisConATS2004 dient zur Visualisierung und Steuerung von Prozessabläufen. Sie kommuniziert über ein Schnittstellenprogramm (MultiSPS2004) mit der jeweiligen Steuerung (SPS). Die Kommunikation wird über eine Socket Verbindung hergestellt. Damit können VisConATS2004 und MultiSPS2004 auf verschiedenen Rechnern ausgeführt werden. Wenn der Steuerungsrechner im Internet "bekannt" ist, können VisConATS2004 und MultiSPS2004 auch über das Internet Daten austauschen.

Mit folgenden Steuerungstypen kann im Moment kommuniziert werden:

- Siemens S7 MPI (Prodave Mini von Siemens wird benötigt)
- Allen Bradley DF1 Protokoll (SLC 50x, PLC 5, Micro Logix)
- Kloeckner Moeller SUCOM A

Bei Bedarf werden weitere Kommunikationsprotokolle implementiert.

Die Anzahl der Variablenpunkte ist unbegrenzt. Das Programm ist Runtime und Konfigurator zugleich. Der Konfigurator ist in der Regel unsichtbar und kann über einen speziellen Service Schalter vom Projekt angezeigt werden. Wird die VisConATS2004 ohne Projekt gestartet, ist der Konfigurationsmodus automatisch aktiv.

Das Projekt wird im Delphi (.dfm) Format gespeichert, sie können mit Delphi geöffnet und bearbeitet werden.

VisConATS2004 ist Mehrsprachenfähig. Jedes Objekt, das einen Text beinhaltet, besitzt ein Stringlistenfeld, in dem die verschiedenen Sprachen gespeichert werden. Sämtliche Texte können in eine Textdatei exportiert werden, die mit Excel (Textdatei mit Tabstops getrennt) geöffnet und bearbeitet werden kann. Somit können Übersetzungen einfach zum Übersetzen weitergeleitet werden. Die fertig übersetzte Datei, kann dann wieder importiert werden.

Mit VisConATS2004 ist eine Schrittkettenprogrammierung möglich. Dem Kunden kann diese Option gegeben werden. Diese Schrittketten können zur SPS übertragen und dort dementsprechend verarbeitet werden. Diese Option bietet sich dann an, wenn eine Anlage viele unterschiedliche Prozesse sequentiell abwickeln muss (z.B. einmal mit Einlegen von Teilen, einmal ohne, Kernzüge unterschiedlich ...). Das SPS – Programm muss entsprechend angepasst sein.

Viele Objekte haben die Möglichkeit ein Script (einfacher Befehlsinterpreter) auszuführen. Damit können SPS unabhängige Aktionen ausgeführt werden, die die Bedienung und SPS Programmierung vereinfachen können. Man kann zum Beispiel mit Hilfe der Scripts eine Prozesswertüberwachung integrieren ohne Änderung des SPS – Programms.

In den nächsten Kapiteln werden die einzelnen Objekte aufgeführt und erklärt.



## 2 Scripte

Viele Komponenten dieser Software sind in der Lage ein Script auszuführen. Die Scripte werden bei Schaltern ausgeführt, wenn der Bediener diesen Schalter betätigt oder Scripte werden ausgeführt, wenn sich der Wert eines Eingabefeldes ändert. Damit kann man auf Aktionen des Bedieners oder Veränderungen in der SPS reagieren.

Ein Beispiel:

In der SPS wird eine Fehlermeldung ausgelöst, wenn ein Sicherheitsendschalter (E 5.0) auslöst. Auf der Oberfläche gibt es eine Seite, die die Funktionalität dieses Sicherheitsendschalters beschreibt. Tritt der Fehler auf, soll sofort auf diese Seite gesprungen werden, damit der Bediener sofort eine Hilfestellung zur Hand hat.

Fehlermeldungen sind Komponenten, die auch ein Script abarbeiten, wenn die Meldung von FALSE auf TRUE wechselt. In diesem Beispiel, kann in diesem Script ein Schalter aktiviert werden, der dafür zuständig ist um auf besagte Seite zu wechseln.

SCRIPT:

```
if self.zIsActive = True then  
  ATSTJumpToErrorPage.zBtnClick := True
```

### 2.1 Syntax

Scripte müssen einer gewissen Syntax folgen:

1. Eventuell zuweisen von Komponenten (z.B. Progloader : zFindTimer), für eine Komponente immer eine Zeile
2. Sequenz muss immer mit einer IF Abfrage beginnen
3. IF Abfrage muss wie folgt aufgebaut sein :  
IF Komponentename.Eigenschaft =(>, <>, <, >=, <=) Wert then  
Anstatt Komponentename kann auch self genommen werden, wenn es die Komponente ist, die dieses Script beinhaltet.
4. Befehle
5. Befehle müssen wie folgt aufgebaut sein:  
Komponentename.Eigenschaft := Wert  
Komponentename.Eigenschaft := Komponentename.Eigenschaft  
Komponentename.Eigenschaft := Komponentename.Eigenschaft +(-, \*, /, div) Wert  
Komponentename.Eigenschaft := Komponentename.Eigenschaft + (-, \*, /, div)  
Komponentename.Eigenschaft  
exit (Script wird beendet)
6. ELSE Zweige kann es geben
7. Befehle

Beispiel:

```
compVar_1 := ProgLoader_Mc.findTimer([16,17,18],[])
```

```
if self.Tag <> 0 then  
  self.Tag := 0
```

```
if compVar_1 then  
  self.zAddressFileNumber := compVar_1.zAddressFileNumber  
  self.zAddressNumber := compVar_1.zAddressNumber  
  self.LabelCaption := compVar_1.LabelCaption  
  self.zAdvise := True  
  self.Visible := True  
  ATSTrackbar_Fuellzeit.Visible := True
```



```
ATSText_Fuellzeit.Visible := True
ATSText_Fuellzeit.Caption := self.Text + s
variantVar := False
self.Tag := 1
exit
else
self.Visible := False
ATSTrackbar_Fuellzeit.Visible := False
ATSText_Fuellzeit.Visible := False
```

## 2.2 Operatoren:

- +, -, \*, /
- div : Division mit anschließender Rundung (Wert wird ein Integer)
- mod : Modulo
- shl : Shift left
- shr : Shift right
- or, and, xor

## 2.3 Lokale Variablen :

- variantVar :  
Sie kann als String, Integer, Float, Bool benutzt werden
- compVar\_1, compVar\_2, compVar\_3, compVar\_4 :  
Ihr kann eine Komponente zugewiesen werden. (z.B. Progloader : zFindTimer)

## 3 Objektauflistung

Folgende Objekte stehen zur Verfügung:

1. Komponenten
  - Kommunikationskomponente
  - Programmlader (Speichern und Übertragen von Programmen, Rezepturen)
  - Fehlermeldung
  - Hinweise
  - Programmierbare Ein- Ausgänge für Schrittketten
  - Schritteendebedingungen für Schrittketten
  - Makros für Schrittketten
  - Speichermanager (Festplattenspeicherprüfung)
  - Zeit und Scriptmanagement (Zeitabhängige Befehle ausführen)
  - Datenprotokollierung (Prozessdatenerfassung)
  - Dokumentverwaltung (z.B. Öffnen und Anzeigen von PDF Dateien als Hilfe)
  - Lizenzmanager (Eingabe einer Lizenznummer)



- Freies Erstellen von Protokolldateien (Zyklisches Schreiben von speziellen Protokolldateien im Textformat)
  - Komponente zum automatischen Ausloggen.
2. Visuelle Objekte :
- Seiten
  - Menü
  - PopUp Menü
  - Eingabefeld
  - Eingabefelder mit Scriptfunktion
  - Textfeld
  - Schalter
  - Messwerverfassung
  - Listbox
  - Shapes (einfache Rechtecke, Kreise ...)
  - Images (Grafiken)
  - Schieberegler
  - Bediener (User) Verwaltung
  - Bargraphen
  - Fehlermeldungsanzeige
  - Auftragsverwaltung
  - Wartungstool

## 4 Komponenten

### 4.1 Kommunikationskomponente (TATSPLC)

TATSPLC stellt die Verbindung zu dem Programm MultiSPS2004 her und verwaltet das komplette Lesen, Schreiben und „Refreshen“ aller Objekte die dieser Komponente zugeordnet sind. Objekte die Daten von der SPS benötigen müssen mit der TATSPLC Komponente verknüpft sein.

Es können mehrer TATSPLC Komponenten im Projekt angelegt werden, d.h. ein Projekt kann mit mehreren Steuerungen kommunizieren und sie verwalten.

#### 4.1.1 Eigenschaften von TATSPLC (properties)

1. Name (String):  
Name der Komponente
2. Tag (Integer):  
frei verwendbar
3. zAutoConnect (Bool):  
True : Bei Programmstart wird automatisch Kommunikation mit MultiSPS2004 gestartet  
False: Bei Programmstart wird versucht den zuletzt eingestellten Zustand einzunehmen.
4. zBaudrate (selection):  
Auswahl der Baudrate mit dem über die serielle Schnittstelle kommuniziert werden soll. Dieser Wert wird bei der Verbindungsaufnahme mit MultiSPS2004 weitergeleitet.
5. zCheckType (selection)  
Auswahl des Protokollchecks (z.b bei SLC 503 → BCC oder CRC).
6. zComport (selection)  
Auswahl der seriellen Schnittstelle über die kommuniziert werden soll (Com1 bis Com9)
7. zConnect (Bool)  
True = Verbindung zu MultiSPS2004  
False = Keine Verbindung zu MultiSPS2004
8. zHostName (String)  
Eingabe auf welchem Host MultiSPS2004 angesprochen werden soll. „localhost“ entspricht immer dem eigenen Rechner. Der Hostname kann als IP Adresse oder Name (Computername) gewählt werden.
9. zLanguageNumber (Integer)  
Gerade aktive Sprache
10. zNetAddress (Integer)  
Falls die Steuerungen über Adressen verfügen (z.B. S7 MPI Adresse), wird diese hier angegeben.
11. zType (selection)  
Auswahl der Steuerung (S7, SLC ...)

## 4.2 Programmloader (TATSProgLoader)

Die Komponente TATSProgloader verwaltet das Übertragen von Programmen zur Steuerung und das Lesen von Programmen von der Steuerung. Außerdem stellt sie Funktionen zur Verfügung, um eine Analyse der Schrittkette durchführen zu können, z.B. um einen speziellen Timer zu finden oder einen Schritt zu finden, in dem eine gewisse Funktion ausgeführt wird. Dazu müssen dann Scripte ausgeführt werden, Scriptaufbau siehe Kap. 3.10.

Startet das Programm neu oder wird eine Schrittkette (Rezeptur) neu übertragen, sucht die Komponente automatisch nach visuellen Komponenten die direkt mit einer Schrittkette in Verbindung stehen, z.B. ein Eingabefeld für einen Schrittwert (Timer, Position ..), und weist ihnen entsprechende Texte (Text für einen Schritt) oder Werte zu, oder macht sie eventuell unsichtbar, wenn sie in der aktuellen Schrittkette nicht gebraucht werden (z.B. es stehen 25 Eingabefelder als Timer zur Verfügung, aber es werden für die aktuelle Schrittkette nur 10 gebraucht).

### 4.2.1 Eigenschaften von TATSProgLoader (properties)

1. EndValNr (Integer)  
Da jeder Schritt bis zu 6 Endwerte haben kann (z.B. Zeit oder Temperatur → zwei Endwerte) muss man angeben an welcher Stelle der Endwert für die jeweilige Endbedingung steht, wenn man Werte von einem gewissen Schritt lesen will.
2. findPosition (TComponent) (only read)  
Da die Schrittketten frei programmierbar sind, kann sich eine spezielle Position, die man immer braucht verändern, sie kann einmal im Schritt 1, einmal im Schritt 4 sein. Um diese Position immer zu finden kann man in den Scripten die Funktion findPosition aufrufen und ihr Argumente mitgeben, welche Ausgänge programmiert und welche nicht programmiert sein dürfen. Anhand dieser Argumente findet diese Funktion dann ein visuelles Eingabefeld das den Endwert des Schrittes anzeigt und gibt diese ‚Komponente‘ zurück.
3. findPressure (TComponent) (only read)  
s. 2.
4. findTemperature (TComponent) (only read)  
s. 2.
5. findTimer (TComponent) (only read)  
s. 2.
6. GetStepCondition (Integer) (read only)  
Gibt den Wert der Schrittabbedingung für den ausgewählten Schritt zurück
7. GetStepSetValue (String) (read only)  
Gibt den Sollwert der ausgewählten Schrittabbedingung (EndValNr) des aktuellen Schrittes zurück
8. GetStepText (String) (read only)  
Gibt den Schrittext des ausgewählten Schrittes zurück
9. Name (String)  
Name der Komponente
10. readStepNr (Integer)  
Schritt der ausgewählt werden soll.
11. Tag (Integer)  
Frei verwendbar
12. zAddressFileNumberAction (Integer)  
Gibt das Datenfile an. Wird eine Schrittkette über den Progloader in die Steuerung übertragen, wird diese Aktion der Steuerung mitgeteilt, dadurch kann in der Steuerung reagiert werden und andere Programme, die auch online mit der Steuerung verbunden sind, können synchronisiert werden.
13. zAddressFileNumberNrOfSteps (Integer)  
Gibt das Datenfile an, in dem die Anzahl der Schritte der aktuellen Schrittkette steht.



14. zAddressFileNumberStation (Integer)  
Gibt das Datenfile an, in der die aktuelle Stationsnummer (zStation) + ein 4Bit Zaehler + 4Bit Aktion steht.

4 BIT Aktionszähler	8 BIT Stationsnummer	4 BIT Aktion
------------------------	-------------------------	-----------------

Die Stationsnummer wird automatisch an den Progloader bei Verbindungsstart verteilt.

15. zAddressFileNumberSteps0 (Integer)  
Gibt das Datenfile an, in dem die ersten 256 Worte der Schrittkette abgelegt sind.
16. zAddressFileNumberSteps1 (Integer)  
Gibt das Datenfile an, in dem die zweiten 256 Worte der Schrittkette abgelegt sind.
17. zAddressFileNumberSteps2 (Integer)  
Gibt das Datenfile an, in dem die dritten 256 Worte der Schrittkette abgelegt sind.
18. zAddressFileNumberSteps3 (Integer)  
Gibt das Datenfile an, in dem die vierten 256 Worte der Schrittkette abgelegt sind.
19. zAddressFileNumberSteps4 (Integer)  
Gibt das Datenfile an, in dem die fünften 256 Worte der Schrittkette abgelegt sind.
20. zAddressFileNumberStepText0 (Integer)  
Gibt das Datenfile an, in dem die ersten 256 Worte der Schritttex te abgelegt werden. Pro Schritt kann ein Text von 32 Zeichen zugeordnet werden → Pro Schritt werden 16 Worte benötigt.
21. zAddressFileNumberStepText0 (Integer)  
Gibt das Datenfile an, in dem die ersten 256 Worte der Schritttex te abgelegt werden.
22. zAddressFileNumberStepText1 (Integer)  
s. 21.
23. zAddressFileNumberStepText2 (Integer)  
s. 21.
24. zAddressFileNumberStepText3 (Integer)  
s. 21.
25. zAddressFileNumberStepText4 (Integer)  
s. 21.
26. zAddressFileNumberStepText5 (Integer)  
s. 21.
27. zAddressFileNumberStepText6 (Integer)  
s. 21.
28. zAddressFileNumberStepText7 (Integer)  
s. 21.
29. zAddressNumberAction (Integer)  
Gibt die Adresse an in der die Aktionsnummer liegt.
30. zAddressNumberNrOfSteps (Integer)  
Gibt die Adresse an, in der die Anzahl der Schritte der aktuellen Schrittkette steht.
31. zAddressNumberSteps0 (Integer)  
Adresse ab der die ersten 256 Worte der Schrittkette abgelegt werden.
32. zAddressNumberSteps1 (Integer)  
s. 31.
33. zAddressNumberSteps2 (Integer)  
s. 31.
34. zAddressNumberSteps3 (Integer)  
s. 31.
35. zAddressNumberSteps4 (Integer)  
s. 31.
36. zAddressNumberStepText0 (Integer)  
Gibt die Adresse an, ab der die ersten 256 Worte der Schritttex te abgelegt werden.
37. zAddressNumberStepText1 (Integer)  
s. 36.
38. zAddressNumberStepText2 (Integer)  
s. 36.

- 39. zAddressNumberStepText3 (Integer)  
s. 36.
- 40. zAddressNumberStepText4 (Integer)  
s. 36.
- 41. zAddressNumberStepText5 (Integer)  
s. 36.
- 42. zAddressNumberStepText6 (Integer)  
s. 36.
- 43. zAddressNumberStepText7 (Integer)  
s. 36.
- 44. zDownLoad (Bool)  
Wird diese Eigenschaft auf TRUE gesetzt, wird der Download Prozess gestartet. Das Standardfenster von Windows zum Öffnen von Dateien erscheint, damit kann die Datei ausgewählt werden, in der die gewünschte Schrittkette (Rezeptur) abgelegt ist. Ist keine DownLoadAccess Variable (s. 45) eingetragen oder ist diese ungleich 0 erscheint eine Fehlermeldung und der Vorgang wird abgebrochen.
- 45. zDownLoadAccess (TATSText)  
Zugeordnete visuelle Variable (z.B. Schrittnummer des Prozesses), die die Freigabe zum Downloaden gibt (=0).
- 46. zLanguageNumber (Integer)  
Gerade aktive Sprache.
- 47. zLength (Integer) (only read)  
ohne Funktion
- 48. zOutPutWords (Integer)  
Anzahl der programmierbaren Ausgangsworte pro Schritt (z.B. 8 → 8 Worte = 128 Ausgänge oder Funktionen). Es kommen jeweils 2 Worte pro Schritt dazu, Wert der Schrittedebedingung und Wert des Endschalters.
- 49. zPLC (TATSPLC)  
Steuerung mit der eine Verbindung hergestellt werden soll.
- 50. zProgList (TStringList)  
Hier wird die aktuelle Schrittkettenstruktur in Textform gespeichert.
- 51. zProgNameField (TATSText)  
Falls der Programmname ebenfalls in der Steuerung abgelegt werden soll, muss eine entsprechende visuelle Komponente hier zugeordnet sein.
- 52. zScriptAfterLoad (TStringList)  
Nach einem Download, wird diese Script abgearbeitet.
- 53. zTextByteChanged (Bool)  
Ist diese Eigenschaft True, werden HighByte und LowByte der Schrittttexte jeweils getauscht.
- 54. zShowVirtKeyboard (Bool)  
Ist die Eigenschaft True, wird beim speichern eines Programs eine Bildschirmtastatur eingeblendet mit der ein Name eingegeben werden kann (Touchscreen).
- 55. zStation (Integer)  
Zeigt die automatisch ermittelte Stationsnummer an.
- 56. zTextBytesChanged (Bool)  
Ist diese Eigenschaft True, werden beim Download die Zeichen der Schrittttexte Byteweise vertauscht (z.B. Siemens S7 Konvention).
- 57. zUpLoad (Bool)  
Wird diese Eigenschaft auf True gesetzt, wird das im Progloader aktuelle Programm über den Standardspeicher – Dialog zum Speichern angeboten.

### 4.3 Fehlermeldung (TATSError)

TATSError ist eine Komponente, die eine Fehlermeldung darstellt, die entweder von der SPS gelesen wird oder von einem Script beschrieben wird. Sie hält nur den aktuellen Text (gerade aktive Sprache), verwaltet die verschiedenen Sprachen.

### 4.3.1 Eigenschaften von TATSError (properties)

1. Name (String)  
Name der Komponente
2. Tag (Integer)  
Frei verwendbar
3. Text (String)  
Gibt den aktuellen Fehlertext der Komponenten zurück (ausgewählte Sprache).
4. zActivationTime (String)  
Gibt das Datum an, wann diese Meldung das letzte Mal auf aktiv gewechselt hat.
5. zAddressFileNumber (Integer)  
Gibt das Datenfile an in dem das Fehlerbit sitzt (DB, N ...).
6. zAddressNumber (Integer)  
Gibt das Datenwort an in dem das Fehlerbit sitzt.
7. zAdvise (Bool)  
True = das Bit wird ständig aus der SPS gelesen (zRefreshTime).  
False = das Bit wird nur einmal aus der SPS gelesen.
8. zATSHint (TATSHint)  
Zugeordnete Hinweis- (Text) Komponente.
9. zBitNumber (Integer)  
Gibt die Bitnummer des Fehlers an.
10. zErrorNumber (Integer)  
Man kann hier eine definierte Nummer für diesen Fehler festlegen.
11. zFileType  
Bei einzelnen Steuerungen muss ein Filetyp angegeben werden, z.B. bei Allen Bradley ob es ein Integer oder Real file ist.
12. zIsActive (Bool)  
True = Fehler ist aktiv.  
False = Fehler ist nicht aktiv.
13. zLanguageList (TStringList)  
Jede Zeile dieser Liste steht für eine Sprache (z.B. 0 = deutsch, 1 = englisch ...).
14. zLanguageNumber (Integer)  
Gerade aktive Sprache.
15. zPLC (TATSPLC)  
Steuerung mit der eine Verbindung hergestellt werden soll.
16. zRefreshTime (Integer) in ms  
Zeit nach der der Wert aktualisiert wird, wenn zAdvise = True ist.  
TATSPLC ermittelt für ein File immer die kleinste Refreshzeit aller Komponenten und aktualisiert alle die auf dieses File zugreifen mit dieser Zeit.
17. zScriptOnError (TStringList)  
Bei kommendem Fehler wird das hier abgelegte Script abgearbeitet.
18. zSelected (Bool)  
True = Fehler ist gerade ausgewählt.
19. zType (selection)  
Fehler kann als Fehler, Warnung, Hinweis, Kommunikationsfehler deklariert werden.
20. zWriteState (Bool)  
Wird die Eigenschaft auf True gesetzt, wird versucht das Fehlerbit in der SPS auf 1 zu setzen (einmalig). Wird sie auf False gesetzt, wird versucht das Fehlerbit in der SPS auf 0 zu setzen (einmalig).

### 4.4 Hinweise (TATSHint)

Hinweise sind einfache „Textspeicher“, die gleichzeitig verschiedenen Komponenten zugeordnet werden können, als „Hint“ für Schalter, Eingabefelder usw. oder als Erklärung für Fehlermeldungen die

als extra Text in Verbindung mit einer Fehlertabelle (TATSErrorGrid) angezeigt werden. Sie können mehrere Zeilen lang sein.

#### 4.4.1 Eigenschaften von TATSHint (properties)

1. Name (String)  
Name der Komponente
2. Tag (Integer)  
Frei verwendbar
3. Text (String)  
Eingabemöglichkeit für den Text der aktuell in der Komponenten angewählten Sprache.
4. zLanguageList (TStringList)  
Liste aller Sprachtexte (kann bei einzeiligen Texten auch zur Bearbeitung genommen werden).
5. zLanguageNumber (Integer)  
Gerade aktive Sprache.

#### 4.5 Programmierbare Ein/Ausgänge (TATSProgEAs) E/A

Um eine Schrittkette programmieren zu können, muss der Anwender „programmierbare“ Ein/Ausgänge (Funktionen) und Schrittenebedingungen zur Verfügung haben. Die Ein/Ausgänge werden über die Komponente TATSProgEAs definiert.

##### 4.5.1 Eigenschaften von TATSProgEAs (properties)

1. ActText (String) (only read)  
Eine reine Leseeigenschaft, in der der aktuelle Text der gerade aktiven Sprache steht (Kann zu Abfragen genutzt werden)
2. Glyph (TBitmap)  
Der Komponente kann ein Bitmap (Icon) zugeordnet werden, das beim „Programmieren“ angezeigt wird.
3. Name (String)  
Name der Komponente
4. Tag (Integer)  
Frei verwendbar
5. zLanguageList (TStringlist)  
Liste aller Sprachtexte (kann bei einzeiligen Texten auch zur Bearbeitung genommen werden).
6. zLanguageNumber (Integer)  
Gerade aktive Sprache.
7. zNumber (Integer)  
Entspricht der laufenden Bitnummer des Ein/Ausgangs. Jeder Schritt wird als Block übertragen. In den ersten Worte (abhängig von der Anzahl der Ausgänge) werden die Ausgänge abgelegt.  
z.B. :  
zNumber = 0 → Ausgangswort 0 Bit 0  
zNumber = 17 → Ausgangswort 1 Bit 1  
zNumber = 31 → Ausgangswort 1 Bit 15
8. zPLC (TATSPLC)  
Steuerung mit der eine Verbindung hergestellt werden soll.
9. zType (selection)  
vrInput == Eingang  
vrOutput == Ausgang

## 4.6 Schrittabedingungen (TATSEndOfStepCond)

Um eine Schrittkette programmieren zu können, muss der Anwender „programmierbare“ Ein/Ausgänge (Funktionen) und Schrittabedingungen zur Verfügung haben. Die Schrittabedingungen werden über die Komponente TATSEndOfStepCond definiert.

### 4.6.1 Eigenschaften von TATSEndOfStepCond (properties)

1. ActText (String) (only read)  
Eine reine Leseeigenschaft, in der der aktuelle Text der gerade aktiven Sprache steht (Kann zu Abfragen genutzt werden)
2. Glyph (TBitmap)  
Der Komponente kann ein Bitmap (Icon) zugeordnet werden, das beim „Programmieren“ angezeigt wird.
3. Name (String)  
Name der Komponente
4. Tag (Integer)  
Frei verwendbar
5. zLanguageList (TStringlist)  
Liste aller Sprachtexte (kann bei einzeiligen Texten auch zur Bearbeitung genommen werden).
6. zLanguageNumber (Integer)  
Gerade aktive Sprache.
7. zNumber (Integer)  
Definiert die Schrittabedingung für die Steuerung. Jeder Schritt wird als Block übertragen. Im zweitletzten Wort steht immer der Wert von zNumber. Der Wert muss in der SPS dementsprechend ausgewertet werden.
8. zPLC (TATSPLC)  
Steuerung mit der eine Verbindung hergestellt werden soll.
9. zRightOfPointPositions (Integer)  
Anzahl der Nachkommastellen des Sollwertes dieser Endbedingung
10. zType1,2,3,4,5,6 (selection)  
vrConstant = Konstante (allgemeine Variable)  
vrTimer = Zeit (Schrittabedingung durch eine Zeit)  
vrPosition = Position (Schrittabedingung durch eine Position)  
vrTemperature = Temperatur (Schrittabedingung durch eine Temperatur)  
vrPressure = Druck (Schrittabedingung durch einen Druck)  
vrLimswitch = Endschalter (Schrittabedingung durch einen Eingang)  
vrNone = Kein Endwert (TComponent) (only read)  
Jede Schrittabedingungskomponente kann bis zu 6 verschiedene „SOLL – Endwerte“ haben, diese können in der SPS beliebig als ‚UND‘, ‚ODER‘ Bedingungen verarbeitet werden.

## 4.7 Makros für Schrittketten (TATSProgMakro)

Makros stellen Teilschrittketten dar, die aneinandergehängt für die jeweilige Steuerung eine korrekte Schrittkette ergeben. Der Programmierer kann damit dem Endnutzer ein Werkzeug zur Verfügung stellen um Standardprogramme (Rezepte) zu erstellen. In der Anwendung kann über einen Schalter ein Fenster (Autoloader) aktiviert werden, mit dessen Hilfe eine Schrittkette durch die Makros erstellt werden kann.

Die Makros können verschiedenen Gruppen und Untergruppen zugeordnet werden, die im „Autoloader“ in verschiedenen Spalten dann angezeigt werden. Jede Spalte entspricht dann einem Schrittkettenblock

Ein Beispiel:



3 Blöcke, aus jedem kann der Endnutzer ein Makro auswählen. Die drei ausgewählten ergeben dann eine komplette Schrittkette.

#### 4.7.1 Eigenschaften von TATSProgMakro (properties)

1. ActText (String) (only read)  
Eine reine Leseeigenschaft, in der der aktuelle Text der gerade aktiven Sprache steht (Kann zu Abfragen genutzt werden)
2. Glyph (TBitmap)  
Der Komponente kann ein Bitmap (Icon) zugeordnet werden, das beim „Programmieren“ angezeigt wird.
3. Name (String)  
Name der Komponente
4. Tag (Integer)  
Frei verwendbar
5. zLanguageList (TStringlist)  
Liste aller Sprachtexte (kann bei einzeiligen Texten auch zur Bearbeitung genommen werden).
6. zLanguageNumber (Integer)  
Gerade aktive Sprache.
7. zNumber (Integer)  
Frei verwendbar
8. zPLC (TATSPLC)  
Steuerung mit der eine Verbindung hergestellt werden soll.
9. zProgGroup (Integer)  
Identifiziert die Spalte für den „Autoloader“.  
min. : 1  
max.: 5
10. zProgList (TStringlist)  
Hier wird die Schrittkettenstruktur in Textform gespeichert.

11. zShowVirtKeyboard (Boolean)  
True = Wird die Schrittkette für das Makro bearbeitet erscheint bei Eingabefenstern eine Bildschirmstatur, damit der Wert über den Bildschirm eingegeben werden kann (Touchscreenanwendung).
12. zSubProgGroup (Integer)  
Wird im „Autoloader“ ein Makro in einer Spalte aktiviert, werden alle Makros in allen Spalten ausgeblendet, die ungleich dieser zSubProgGroup – Zahl und ungleich 0 (allgemein) sind. Damit kann eine „Führung“ zum Programm erstellen für den Endanwender angelegt werden.

## 4.8 Zeit und Scriptmanagement (TATSTimerScript)

TATSTimerScript Komponenten enthalten ein Scriptmanagement, das in ca. 50ms Intervallen immer abgearbeitet wird und eine gleichzeitige Timerfunktionalität, durch die man Zeitgesteuerte Funktionen oder Abfragen aus der Steuerung (SPS) in die Oberfläche nehmen kann.

Zum Beispiel kann man damit einen Drucksensor auf Funktionalität testen, ist eine Druckkammer geöffnet und ist der Wert des Drucksensors länger als 1 Sekunde > 0,1 bar, wird eine Fehlermeldung gesetzt. Im Script wird der Timer gestartet, gestoppt und abgefragt ob er abgelaufen ist, falls ja wird die Fehlermeldung aktiviert.

### 4.8.1 Eigenschaften von TATSTimerScript (properties)

1. Name (String)  
Name der Komponente
2. Tag (Integer)  
Frei verwendbar
3. zCurrentTime (Integer)  
Abgelaufene Zeit in Millisekunden
4. zEnabled (Bool)  
Ist diese Eigenschaft auf True läuft der Timer
5. zScript (TStringList)  
Befehle, die abgearbeitet werden sollen
6. zTime (Integer)  
Sollzeit des Timers in Millisekunden
7. zTimeDone (Bool)  
Ist die Eigenschaft auf True, ist der Timer abgelaufen
8. zTimeStop (Bool)  
Wird diese Eigenschaft auf True gesetzt, wird die ablaufende Zeit angehalten, bei False setzen, läuft die Zeit an dieser Stelle weiter.

## 4.9 Datenprotokollierung (TATSDataLogger)

Mit der Datenprotokollierung können alle Ereignisse, Wertänderungen, Bedieneraktionen erfasst und in Dateien tageweise auf dem Rechner gespeichert werden. Zusätzlich kann eine Verbindung zu einer visuellen Komponente, die zur Auftragsverwaltung dient, hergestellt werden, um jeder Speicherung den dazugehörigen Auftrag mit zu geben. Es werden pro Tag immer 3 Dateien im ASCII Format angelegt. Eine Zeile entspricht immer einem Eintrag und die einzelnen Punkte einer Zeile werden durch Tabs getrennt:

- **JJJJ\_MM\_TT.MS**  
In dieser Datei werden alle kommenden und gehenden Fehlermeldungen im folgenden Format abgelegt:  
DATUM und UHRZEIT aktueller Bediener      Maschine      Aktiver Auftrag      Zustand (True  
= gekommen, False = gegangen)      Fehlernummer      Fehlertext      Fehlername



- **JJJJ\_MMM\_TT.VL**

In dieser Datei werden alle Wertänderungen, z.B. Zykluszähler, Zykluszeiten ..., abgelegt.  
Format:

DATUM	UHRZEIT	aktueller Bediener	Maschine	Aktiver Auftrag
		Variablenbeschreibung	Wert	Variablenname

- **JJJJ\_MM\_TT.OP**

In dieser Datei werden alle Daten abgelegt, die durch eine Aktion des Bedieners verändert wurden, z.B. Änderung eines SOLL-Werts, Ein/Ausloggen.

Format:

DATUM	UHRZEIT	aktueller Bediener	Maschine	Aktiver Auftrag
		Variablenbeschreibung	Wert	Art der Variable
				Name der Variable

#### 4.9.1 Eigenschaften von TATSDatlogger (properties)

1. Name (String)  
Name der Komponente
2. Tag (Integer)  
Frei verwendbar
3. zActivated (Boole)  
Ist diese Eigenschaft wahr, dann arbeitet der Datalogger.
4. zEditValueList (TStringList)  
Hier werden die Komponentennamen angegeben, die in der .OP Datei gespeichert werden sollen. Steht in der ersten Zeile ein **All**, werden alle Aktionen gespeichert.
5. zErrorList (TStringList)  
Hier werden die Fehlernamen eingetragen die protokolliert werden sollen. Steht in der ersten Zeile ein **All**, werden alle Fehlermeldungen protokolliert.
6. zFilePath (String)  
Angabe, in welchem Ordner die Dateien gespeichert werden sollen (z.B. **C:\Orders\**). Dieser Ordner muss existieren, sonst werden keine Daten gespeichert.
7. zOrderManager (TATSOOrderGrid)  
Hier kann eine Auftragsverwaltungskomponente zugeordnet werden.
8. zTextValueList (TStringList)  
Hier werden die Komponentennamen eingegeben, die in der .VL Datei abgelegt werden sollen. Steht in der ersten Zeile ein **All**, werden alle Aktionen gespeichert.