Systemerweiterung
der Verkehrsrechnerzentrale
in Baden-Württemberg

Skriptsprache der Umfassenden Datenanalyse

Segment 6 (IBV), SWE 6.3 Umfassende Datenanalyse

Version	1.0
Stand	14.02.2008
Produktzustand	Vorgelegt
Datei	UdaDoc_SWE6.3_LosC1C2_VRZ3.doc

Projektkoordinator	Herr Dr. Pfeifle
Projektleiter	Herr Dr. Pfeifle
Projektträger	Regierungspräsidium Tübingen Landesstelle für Straßentechnik Heilbronner Straße 300 - 302 70469 Stuttgart
Ansprechpartner	Herr Dr. Pfeifle

0 Allgemeines

0.1 Verteiler

Organisationseinheit	Name	Anzahl Kopien	Vermerk
PG VRZ 3			Bereitstellung auf dem Dokumentenserver

0.2 Änderungsübersicht

Version	Datum	Kapitel	Bemerkungen	Bearbeiter
1.0	14.02.2008		Erstellung erster Entwurf	U. Peuker

0.3 Inhaltsverzeichnis

0.4 Abkürzungsverzeichnis

Die für das Projekt VRZ 3, Los C1+C2 relevanten Abkürzungen sind in einem separaten Dokument zusammengefasst.

0.5 Referenzierte Dokumente

[UdaDoc] Skriptsprache der Umfassenden Datenanalyse

0.6 Abbildungsverzeichnis

Fehler! Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

0.7 Tabellenverzeichnis

1 Zweck des Dokuments

In diesem Dokument ist die Sprache zur Erstellung von Skripten, die innerhalb der Umfassenden Datananalyse ausgeführt werden sollen dokumentiert. Das Handbuch

Folgende Typographie wird verwendet:

kursiv	Namen von Dateien, Ordnern und Benutzern
`Maschinenschrift`	Befehle und Texte die in der Kommandozeile oder einem graphischem Dialog eingeben werden
`Maschinenschrift im Fettdruck`	Teil eines Befehls oder Eingabetextes, der individuell angepasst werden muss

Tabelle 1.1: Typographie

Folgende Konventionen werden festgelegt:

$VRZ3_HOME	Das Verzeichnis in dem die Kernsoftware installiert ist
$VRZ3_SWE	Das Verzeichnis in dem diese SWE installiert wird

Tabelle 1.2: Konventionen

2 Die Sprache der Umfassenden Datenanalyse

2.1 Allgemein

Die Skriptsprache zur Umfassenden Datenanalyse ist eine für den Anwender einfach verständliche Sprache, um einfache Aufgaben der Verarbeitung von Daten eines verbundenen Datenverteilersystems zu beschreiben und die Daten auszuwerten.

- Bezeichner können Groß- und Kleinbuchstaben, Umlaute, den Buchstaben „ß“, den Unterstrich „_“ und Ziffern enthalten. Namen von Konstanten, Variablen, Datenstrukturen, Containern, Funktionen und Methodenaufrufen sind Bezeichner. Ein Bezeichner darf nicht mit einem Ziffernzeichen beginnen.
- Die Schlüsselwörter der Skriptsprache sind in deutscher Sprache
- Die syntaktischen Konstrukte der Sprache sind einfach gehalten und fast umgangssprachlich lesbar
- Über entsprechende Konstrukte innerhalb der Sprache kann auf die Daten innerhalb der Konfiguration eines verbundenen Datenverteiler-Systems zugegriffen werden. Außerdem stehen Methoden zur Verfügung, um in einer an die Datenverteilerapplikationsfunktionen angelehnten Art und Weise auf Onlinedaten schreibend und lesend zuzugreifen.

Die Umfassende Datenanalyse stellt einen Interpreter zur Verfügung, der ein Skript nach den notwendigen syntaktischen Überprüfungen direkt ausführt. Damit können schnell erste lauffähige Skriptversionen erstellt werden und diese Programme in kurzen Entwicklungszyklen verfeinert werden. Ein UDA-Skript kann außerdem auf einem UDA-Server abgelegt werden, der dessen Ausführung zeitgesteuert, zyklisch oder manuell angestossen übernimmt.

2.2 Aufbau eines Skriptes

Ein Skript besteht aus beliebig vielen Anweisungen, die nacheinander vom Interpreter ausgewertet werden. Neben den Anweisungen können in speziellen Blöcken Funktionen definiert werden, die innerhalb der Anweisungen aufgerufen werden können.
Über das spezielle Konstrukt der "Benutze-Anweisung" können Skripte beliebig hierarchisch in mehrere Teilskripte unterteilt werden. Damit können beispielweise Funktionsdefinitionen in eine eigene Skriptdatei ausgegliedert werden und an entsprechender Stelle des Skriptes hinzugeladen werden. Die „Benutze“-Anweisung ist nur nutzbar, wenn das Skript von der Softwareeinheit „Umfassende Datenanalyse“ im Serverbetrieb ausgeführt wird, da auf die untergeordneten Teilskripte über ihren Namen zugegriffen wird.
An jeder Stelle des Skriptes können Kommentare zur Dokumentation angegeben werden. Ein Kommentar wird durch das Zeichen "#" eingeleitet und gilt bis zum Zeilenende.

2.3 Konstanten, Literale

Konstanten sind Datenobjekte mit einem festen, d.h. zeitlich unveränderlichen Wert. Der Typ der Konstanten ergibt sich aus seiner Schreibweise. Folgende Arten von Konstanten werden unterstützt.

Ganze Zahlen

Ein Literal, das nur aus Ziffernzeichen steht wird als „Ganze Zahl“ interpretiert. Optional kann als erstes Zeichen das Zeichen '-' verwendet werden, um negative Zahlen auszudrücken. Eine ganze Zahl wird in der Umfassenden Datenanalyse durch einen 64-Bit-Wert repräsentiert, was zu einem Wertebereich von –9223372036854775808 bis 9223372036854775807 führt.

Beispiele für ganze Zahlen sind: -13, 0, 255.

Fließkommazahlen

Ein Literal, das mit einem Ziffernzeichen beginnt und zusätzlich die Zeichen „.“, „e“ und/oder „E“ enthält, wird als Fliesskommazahl betrachtet. Optional kann das Zeichen '-' vorangestellt werden, wenn ein negativer Wert beschrieben werden soll.
Das Zeichen „e“ wird verwendet, wenn die Zahl in Exponentialschreibweise dargestellt werden soll. Dem Exponenten kann ebenfalls optional ein „-„-Zeichen vorangestellt werden.
Fließkommazahlen werden in der Umfassenden Datenanalyse durch einen 64 Bitt-Wert dargestellt. Der Wertebereich liegt zwischen ±4.94065645841246544e-324 und ±1.79769313486231570e308.
Beispiele für Fließkommazahlen sind: -13.25, 0.95, 1e10, 1.6e-19.

Logische Konstanten

Als logische Konstanten sind die Werte **`wahr`** und **`falsch`** zulässig.

Zeichenketten

Eine Zeichenkette besteht aus beliebigen Zeichen, die durch Anführungszeichen (") begrenzt sind. Wenn innerhalb einer Zeichenkette das Anführungszeichen benutzt werden soll, muss dieses Zeichen durch einen Rückwärtsschrägstrich (\) eingeleitet werden. Mit dem Zeichen '\n' kann ein Zeilenumbruch veranlasst werden. Ein Rückwärtsschrägstrich wird durch "\\" dargestellt.
Beispiele für Zeichenketten sind: "MQ 1", "VRZ Leverkusen"

Zeitstempel und relative Zeitangaben

Zeitangaben erfolgen innerhalb der Umfassenden Datenanalyse millisekundengenau.
Zeitstempel werden in der Umfassenden Datenanalyse durch einen 64-Bit-Wert als vergangene Millisekunden seit dem 1. Januar 1970, 0:00:00,000 Uhr GMT, dargestellt.

Ein Zeitstempel wird durch eine Zeichenkette im Format „dd.mm.yyyy HH:MM:SS.sss“ angegeben. Um einen Zeitstempel von einer Zeichenkette zu unterscheiden muss die angegebene Zeichenkette an die Funktion „zeitstempel“ übergeben werden, um den eigentlichen Zeitstempelwert zu erzeugen. Die Angabe des Zeitpunktes ist fest an die vorgegebene Struktur gebunden. Folgende Abweichungen sind erlaubt:

- das Datum kann entfallen, in diesem Fall wird das aktuelle Daten eingesetzt
- bei der Datumsangabe kann das Jaht zweistellig angegeben werden, zum angegebenen Wert wird dann 2000 addiert
- Monat, Tag und Stunde können einstellig angegeben werden
- die Zeitangabe kann um beliebige Angaben am Ende reduziert werden, der entsprechende Anteil wird auf den Wert 0 gesetzt
- die Angabe der Millisekunden kann 1 … 3-stellig erfolgen, die Angabe wird intern auf drei Stellen mit nachfolgenden Nullen ergänzt.

Wird die Funktion zeitstempel() ohne Parameter verwendet, liefert die den aktuellen Zeitstempel. Optional steht für den aktuellen Zeitstempel auch das Schlüsselwort „aktuell“ zur Verfügung.

Beispiele für Zeitstempel:

```
z1 = zeitstempel("1.12.2002 14:15:35,756")
z2 = zeitstempel()
z3 = aktuell
```

Eine relative Zeitangabe wird durch eine Angabe der gewünschten Zeit mit einer beliebigen Anzahl von Paaren <wert> <zeiteinheit> definiert.
Als Einheiten sind erlaubt: `Millisekunde(n)`, `Sekunde(n)`, `Minute(n)`, `Stunde(n)`, und `Tag(e)`.
Die Angaben werden nicht auf Reihenfolge und Singularität überprüft, d.h. eine Angabe „1 Stunde 1 Stunde“ liefert eine relative Zeitdauer von 2 Stunden.

Beispiele für relative Zeitangaben:

```
z1 = 25 Stunden 19 Minuten 53 Sekunden 800 Millisekunden
z2 = 15 Minuten
```

Die Verrechung von Zeitwerten kann mit den Operatoren „+“ und „-„ erfolgen. Dabei sind folgende Kombinationen erlaubt.

zeitstempel + zeitstempel	= Argumentfehler
zeitstempel + zeitdauer	= zeitstempel (um zeitdauer verschoben)
zeitdauer + zeitstempel	= zeitstempel (um zeitdauer verschoben)
zeitdauer + zeitdauer	= zeitdauer
zeitstempel - zeitstempel	= zeitdauer (die Differenz zwischen den Zeiten)
zeitstempel - zeitdauer	= Argumentfehler
zeitdauer - zeitstempel	= zeitstempel (um zeitdauer verschoben)
zeitdauer - zeitdauer	= zeitdauer

Über die Namen der einzelnen Bestandteile `tag`, `monat`, `jahr`, `stunde`, `minute`, `sekunde`, `millisekunde` kann auf die einzelnen Werte des Zeitstempels zugegriffen bzw. können diese modifiziert werden.
Die Ausgabe eines Zeitstempels kann durch `format` beeinflusst werden, wobei die Stellen des Datums durch Großbuchstaben („TT.MM.JJJJ“) und die Stellen der Uhrzeit durch Kleinbuchstaben („hh:mm:ss,ttt“) vertreten werden. Folgende Regeln sind dabei einzuhalten:

- Tag, Monat, Stunde und Minute sind zweistellig oder kommen in der Formatzeichenkette nicht vor.
- Das Jahr ist zwei- oder vierstellig oder kommt in der Ausgabe nicht vor.
- Die Millisekunden sind dreistellig oder kommen in der Ausgabe nicht vor.
- Entspricht die Formatzeichenkette nicht den obigen Regeln, so wird die Standarddarstellung „TT.MM.JJJJ hh:mm:ss,ttt“ verwendet.

Beispiele für die Formatierung von Zeitstempeln:

```
tag = z2.tag                        # tag = 30
monat = z2.monat                          # monat = 11
jahr = z2.jahr                            # jahr = 2002
z2.jahr = 1982
ausgabe z2.format("TT.MM.JJ, hh:mm")    # 30.11.82, 16:09
ausgabe z2.format("M.JJ hh:mm:ss")      # 30.11.1982 16:09:03,091
ausgabe z2.format("ttt JJJJ)            # 091 1982
```

Zeitbereiche

Zeitbereiche definieren, wie der Name sagt einen zeitlichen Bereich mit einem Anfangs- und Endpunkt. Sie werden beispielsweise verwendet, um Daten aus dem Archiv abzufragen.

Ein Zeitbereich wird mit Hilfe der Funktion zeitbereich(anfangswert, endwert, inklusive) erzeugt, wobei die Parameter folgende Bedeutung haben:

Parameter	Beschreibung
anfangswert	der Anfangswert des Zeitbereiches. Er wird als Zeitstempel übergeben und ist selbst immer Bestandteil des erzeugten Bereiches.
endwert	der Endwert kann als absoluter Zeitstempel oder als relative Zeitdauer übergeben werden.
inlusive	der Boolwert definiert, ob der Endzeitpunkt selbst Bestandteil des Bereiches ist oder lediglich die Grenze des Bereiches bildet.

Auf die Daten eines Zeitbereiches kann über seine Attribute zugegriffen werden.

Attribute

Name	Beschreibung
anfangszeit	der Anfangszeitpunkt des Zeitbereiches als Zeitstempel.
endzeit	der Endzeitpunkt des Zeitbereiches als Zeitstempel.
inklusive	ein Boolscher Wert, der definiert, ob der Endzeitpunkt selbst Bestandteil des Bereiches ist oder lediglich die Grenze des Bereiches bildet

Der „Nichtwert“ undefiniert

`undefiniert` kennzeichnet, daß eine Variable oder ein Attribut bzw. das Ergebnis einer Funktion oder eines Vergleichs undefiniert ist.
Variablen und Attribute sind initial im Zustand `undefiniert`. Erst durch eine gültige Wertzuweisung erhalten sie einen Wert aus ihrem zugehörigen Wertebereich. Auch in Datensätzen, die mit der Methode `datenLesen()` vom Datenverteiler ermittelt werden, können undefinierte Attribute enthalten.
Der Wert "unbekannt" kann in UDA in Zuweisungen, Vergleichen und innerhalb einer Zeichenketten-Konkenation verwendet werden. Der Versuch mit dem Wert "undefiniert" zu rechnen führt zu einem Fehler.

Eine Variable oder ein Attribut kann darauf überprüft werden, ob sie einen definierten Wert, oder den Zustand `undefiniert` hat. Damit kann vor einem Ausdruck geprüft werden, ob er fehlerfrei ausgeführt werden kann. Das kann optional durch das Attribut „.definiert“ oder durch den Vergleich mit dem Wert „undefiniert“ erfolgen.

Durch Zuweisen des Wertes „undefiniert“ zu einer Variable wird diese typenlos und kann wieder mit einem beliebigen Wert belegt werden.

2.4 Variablen

In Variablen können Informationen innerhalb eines Skriptlaufes zwischengespeichert und zu einem späteren Zeitpunkt wieder abgefragt werden.
Variablen werden nicht deklariert, sie entstehen an jeder Stelle beliebigen Stelle eines Skripts durch Zuweisung.
Variablen können Daten mit beliebigen Datentypen speichern von einfachen Variablentypen (z.B. Ganz-, Fließkommazahl und Zeichenketten) als auch komplexe Typen wie Container, Datenstrukturen und Objekte.
Eine Variable erhält durch die Zuweisung einen Typ, im folgenden Ablauf des Skripts kann dieser Variablen nur ein zu diesem Typ kompatibler Datenwert zugewiesen werden. Durch Zuweisung des Wertes „undefiniert“ wird der Typ einer Variable entfernt.

Allgemein gelten für Variablenzuweiseung folgende Festlegungen:

- Zahlen sind ineinander konvertierbar
- jeder Typ ist ein eine Zeichenkette konvertierbar
- jeder Typ ist in ein Fehlerobjekt konvertierbar
- jedem Typ kann der Wert undefiniert zugewiesen werden, womit die entsprechende Variable ihren Typ verliert

Variablen sind per se immer lokal im Gültigkeitsbereich einer Funktionsdefinition oder des Hauptprogramms verfügbar. Wenn die Variable global verfügbar sein soll, muss sie über ein spezielles Objekt angelegt werden (siehe folgende Kapitel).

2.5 Datenstrukturen

Mit Hilfe von Datenstrukturen können Objekte konstruiert werden, die mehrere unterschiedliche Werte (Elemente) enthalten. Eine Datenstruktur wird durch Auflistung der jeweiligen Elementnamen definiert. Die Definition einer neuen Datenstruktur geschieht über einen Funktionsaufruf und kann somit innerhalb von Ausdrücken an beliebigen Stellen des Skriptes erfolgen. Auf die einzelnen Werte eines Datensatzes der definierten Datenstruktur kann mit Hilfe der Elementnamen zugegriffen werden.
Zur Konstruktion einer neuen Datenstruktur wird die Funktion `Datenstruktur` aufgerufen. Dabei werden die Namen der Elemente übergeben.

```
RaumPunkt= Datenstruktur( "xKoordinate", "yKoordinate", "zKoordinate")
```

Als Ergebnis liefert die Funktion ein Objekt zurück, das die so definierte Datenstruktur beschreibt.
Ein neuer Datensatz dieser Datenstruktur kann mit der Methode `datensatz()` aus der Datenstruktur erzeugt werden.

```
p1= RaumPunkt.datensatz()
```

Anschließend kann auf die Elemente des Datensatzes (die initial `undefiniert` sind) direkt zugegriffen werden.

```
p1.xKoordinate=1
p1.yKoordinate=3
p1.zKoordinate=7
```

Beim Aufruf der Methode `datensatz()` können aber auch initiale Werte für die Elemente des neuen Datensatz mitübergeben werden.

```
p2= RaumPunkt.datensatz( 2, 1, 4)
```

Um einen generischen Zugriff auf eine Datenstruktur zu ermöglichen, kann dieser auch als Container von Zeichenketten mit den Namen der definierten Elemente aufgefasst werden.

```
ausgabe RaumPunkt[3]    # gibt die Zeichenkette "zKoordinate" aus
```

Um einen generischen Zugriff auf einen Datensatz zu ermöglichen, kann dieser auch als Container der Elementwerte aufgefasst werden.

```
ausgabe p2[3] # gibt den Wert des dritten Elements des
              # Datensatzes aus, also den Wert der zKoordinate, also 4
```

Außerdem kann ausgehend von einem Datensatz die zugehörige Datenstruktur mit der Methode `datenstruktur()` bestimmt werden.

```
ausgabe p2.datenstruktur()[3]  # gibt die Zeichenkette "zKoordinate" aus
```

2.6 Container

Im Gegensatz zu Datensätzen, die eine feste Struktur haben, können Container eine beliebige Anzahl von Objekten zusammenfassen. Die einzelnen Objekte eines Containers können dabei beliebigen Typs sein (Container können auch Container enthalten; die Typen der Objekte eines Containers können unterschiedlich sein)
Von der Umfassenden Datenanalyse werden die Containertypen Feld, Liste und Schlüsselmenge bereitgestellt.

2.6.1 Felder und Listen

Felder und Listen bieten die Möglichkeit Objekte in einer vorgegebenen Reihenfolge zu speichern und indiziert auf einzelne Objekte des Containers zuzugreifen.

2.6.1.1 Konstruktion von Feldern und Listen

Ein neues Feld kann durch Aufruf der Funktion `Feld()` konstruiert werden. Über einen optionalen Parameter kann der Funktion eine initiale Größe übergeben werden, d.h. eine Anzahl von Objekten für die von vornherein Platz im Container zur Verfügung stehen soll.

```
feld1= Feld()
feld2= Feld(1000)
```

Wenn an die Funktion Feld() ein anderer Container als Parameter übergeben wird, dann wird das Feld mit den definierten Elementen des anderen Containers gefüllt.

```
feld3= Feld(feld2)
```

Durch den Einsatz von Feldinitialisierern können Felder in einem Schritt konstruiert und mit Werten vorbesetzt werden. Feldinitialisierer werden durch die beiden Zeichen '[' und ']' begrenzt. Zwischen den Klammern kann eine Liste von Ausdrücken mit Komma getrennt angegeben werden. Die Ergebnisse dieser Ausdrücke werden in der gegebenen Reihenfolgen als Elemente in das Feld aufgenommen.

```
zustände= ["frei", "dicht", "zäh", "stau"]
```

Eine neue Liste kann durch den Aufruf der Funktion Liste() konstruiert werden.

```
liste1= Liste()
```

Wenn an die Funktion `Liste()` ein anderer Container als Parameter übergeben wird, dann wird die Liste mit den definierten Elementen des anderen Containers gefüllt.

```
liste2= Liste(feld3)
```

Durch den Einsatz von Listeninitialisierern können Listen in einem Schritt konstruiert und mit Werten vorbesetzt werden. Listeninitialisierer werden durch die beiden Zeichen '{' und '}' begrenzt. Zwischen den Klammern kann eine Liste von Ausdrücken mit Komma getrennt angegeben werden. Die Ergebnisse dieser Ausdrücke werden in der gegebenen Reihenfolgen als Elemente in die Liste aufgenommen.

```
zustandsListe = {"frei", "dicht", "zäh", "stau"}
```

2.6.1.2 Operationen auf Feldern und Listen

2.6.1.2.1 Anzahl Elemente

Über das Attribut `anzahl` eines Containers kann die Anzahl der Elemente im Container bestimmt werden.

```
anzahlZustände= zustände.anzahl
```

Die Anzahl der Elemente eines Feldes entspricht dem größten verwendeten Index im Feld.

2.6.1.2.2 Zugriff auf ein einzelnes Element

Mit dem folgenden Konstrukt kann auf ein beliebiges Element eines Containers zugegriffen werden:

```
container[index]
```

Wenn der Index `1` ist, dann wird auf das erste Element des Containers zugegriffen; wenn der Index `2` ist, dann wird auf das zweite Element des Containers zugegriffen; usw.
Wenn der Index `-1` ist, dann wird auf das letzte Element des Containers zugegriffen; wenn der Index `-2` ist, dann wird auf das vorletzte Element des Containers zugegriffen; usw.
Die einzelnen Elemente verhalten sich wie Variablen, das heißt man kann ihnen Werte zuweisen (schreibender Zugriff) und man kann die Werte abfragen um sie in weiteren Berechnungen zu verwenden (lesender Zugriff).
Der Index 0 und ein Index, der größer als die Anzahl der Elemente oder kleiner als die negative Anzahl der Elemente ist, führt beim lesenden Zugriff zu einem `ContainerZugriffsFehler`.
Ein schreibender Zugriff mit einem Index von 0 oder einem Index, der kleiner als die negative Anzahl Elemente ist, führt zu einem `ContainerZugriffsFehler`. Wird beim schreibenden Zugriff ein Index benutzt, der größer als die Anzahl Elemente ist, dann wird bei einem Feld die Anzahl der Elemente entsprechend erhöht und alle Elemente mit einem Index zwischen der bisherigen Anzahl und der neuen Anzahl werden auf den Wert `undefiniert` gesetzt. Bei einer Liste wird bei schreibenden Zugriff mit einem Index, der um eins größer als die Anzahl der Elemente ist, ein neues Element am Ende der Liste angefügt. Eine größerer Index führt beim schreibenden Zugriff auf eine Liste zu einem `ContainerZugriffsFehler`.

```
primzahlen= [2, 3, 5, 7, 11]
ausgabe primzahlen[3]   # gibt 5 aus
ausgabe primzahlen[-2]  # gibt 7 aus
ausgabe primzahlen[6]   # gibt eine Fehlermeldung aus
primzahlen[7]= 13
ausgabe primzahlen[3]   # gibt 5 aus
ausgabe primzahlen[-3]  # gibt 11 aus
ausgabe primzahlen[6]   # gibt undefiniert aus
ausgabe primzahlen[7]   # gibt 13 aus
```

2.6.1.2.3 Einfügen und Anhängen von Elementen

```
container.einfügen( index, element)
container.einfügen( element)
container.anhängen( index, element)
container.anhängen( element)
```

Die Methode `einfügen` fügt den Parameter `element` in den Container ein. Mit dem Parameter `index` kann der Index eines Elementes im Container angegeben werden, vor dem das neue Element eingefügt werden soll. Negative Werte von Index spezifizieren dabei Elemente vom Ende des Containers aus gesehen. Der Index 0 und ein Index, der größer als die Anzahl der Elemente ist, fügt das neue Element als letztes Element in den Container ein. Ist der Index kleiner als die negative Anzahl der Elemente, wird das neue Element als erstes Element in den Container eingefügt. Wenn der Parameter `index` nicht angegeben wurde, wird der Wert `1` angenommen, d.h. das neue Element wird vor allen anderen Elementen des Containers eingefügt.
Die Methode `anhängen` fügt den Parameter `element` in den Container ein. Mit dem Parameter `index` kann der Index eines Elementes im Container angegeben werden, hinter dem das neue Element eingefügt werden soll. Negative Werte von Index spezifizieren dabei Elemente vom Ende des Containers aus gesehen. Ein Indexwert von 0 fügt das Element vor allen anderen Elementen in den Container ein. Indexwerte, die größer als die Anzahl der Elemente sind fügen das neue Element als letztes Element hinter den vorhandenen Elementen in den Container ein. Ist der Index kleiner als die negative Anzahl der Elemente, wird das neue Element als erstes Element in den Container eingefügt. Wenn der Parameter `index` nicht angegeben wurde, wird der Wert `-1` angenommen, d.h. das neue Element wird hinter allen vorhandenen Elementen des Containers eingefügt.

```
container= [ 1, 2, 3]
container.einfügen( 4)      # [4,1,2,3]
container.einfügen( 2, 5)   # [4,5,1,2,3]
container.einfügen( -3, 6)  # [4,5,6,1,2,3]
container.anhängen( 7)      # [4,5,6,1,2,3,7]
container.anhängen( 6, 8)   # [4,5,6,1,2,3,8,7]
container.anhängen( -3, 9)  # [4,5,6,1,2,3,9,8,7]
```

2.6.1.2.4 Suchen von Elementen

```
element= container.element( index)
```

Mit der Methode `element` kann das Element mit dem Index `index` eines Containers bestimmt werden. Das gefundene Objekt wird von der Funktion zurückgeliefert. Negative Werte des Parameters `index` spezifizieren dabei Elemente vom Ende des Containers aus gesehen. Ein Index, der 0, größer als die Anzahl Elemente oder kleiner als die negative Anzahl der Elemente ist, führt zu einem `ContainerZugriffsFehler`.

2.6.1.2.5 Löschen von Elementen

```
element= container.löschen( index)
```

Die Methode `löschen()` entfernt das Element mit dem Index `index` aus dem Container. Negative Werte des Parameters `index` spezifizieren dabei Elemente vom Ende des Containers aus gesehen. Ein Index, der 0, größer als die Anzahl Elemente oder kleiner als die negative Anzahl der Elemente ist, führt zu einem `ContainerZugriffsFehler`.
Das aus dem Container entfernte Element wird von der Methode zurückgeliefert.

2.6.1.2.6 Selektion von Objekten

Durch Anwendung eines Selektionskonstrukts können aus den Elementen eines Containers Objekte oder Datensätze ausgewählt werden, die bestimmte Attribute mit bestimmten Werten besitzen. Es wird zwischen Mehrfach-Selektion und Einzelselektion unterschieden.

2.6.1.2.7 Mehrfach-Selektion

```
container{attribut1:wert1, attribut2:wert2, ...}
```

Das Mehrfach-Selektionskonstrukt enthält in geschweiften Klammern hinter einem Container eine durch Komma getrennte Liste von Tupeln, die einen Attributnamen und einen Wert durch Doppelpunkt getrennt spezifizieren. Es werden alle Elemente aus dem Container, die die angegebenen Attribute mit den entsprechenden Werten haben, ausgewählt und in einer Liste zurückgegeben.
Der Ausdruck

```
container{richtung: "N", spuren: 2}
```

liefert als Ergebnis eine Liste, mit denjenigen Datensätzen oder Objekten aus dem Container `container`, die ein Attribut `richtung` besitzen, dessen Wert gleich `"N"` ist und ein Attribut `spuren` besitzen, dessen Wert gleich `2` ist.
Wenn kein Objekt selektiert wurde, dann wird eine leere Liste zurückgegeben.

2.6.1.2.8 Einzel-Selektion

```
container[attribut1:wert1, attribut2:wert2, ...]
```

Das Einfach-Selektionskonstrukt enthält in eckigen Klammern hinter einem Container eine durch Komma getrennte Liste von Tupeln, die einen Attributnamen und einen Wert durch Doppelpunkt getrennt spezifizieren. Es wird das erste Element des Containers, das die angegebenen Attribute mit den entsprechenden Werten hat, ausgewählt und zurückgegeben.
Der Ausdruck

```
container[richtung: "N", spuren: 2]
```

liefert als Ergebnis das erste Element aus dem Container `container` das ein Attribut `richtung` besitzt, dessen Wert gleich `"N"` ist und ein Attribut `spuren` besitzt, dessen Wert gleich `2` ist.
Wenn kein Objekt selektiert wurde, dann wird ein `ContainerZugriffsFehler` zurückgegeben

2.6.1.2.9 Beispiele zur Selektion.

```
mqInfo= Datenstruktur("name","typ" , "autobahn", "richtung", "spuren")
mq1=mqInfo.datensatz("a", "induktiv",3,"N",3)
mq2=mqInfo.datensatz("b", "radar"   ,4,"O",3)
mq3=mqInfo.datensatz("c", "radar"   ,3,"S",2)
mq4=mqInfo.datensatz("d", "induktiv",3,"N",3)
mq5=mqInfo.datensatz("e", "induktiv",4,"W",2)
mq6=mqInfo.datensatz("f", "induktiv",4,"W",2)
querschnitte=[ mq1, mq2, mq3, mq4, mq5, mq6]
```

In obigem Beispiel wird ein Container mit 6 Datensätzen konstruiert, die jeweils bestimmte Informationen eines Messquerschnitts enthalten. Folgende Tabelle zeigt an einigen Beispielen die Möglichkeiten der Selektion

Ausdruck	Ergebnis
`querschnitte[name: "f"]`	`mq6`
`querschnitte{name: "f"}`	`{mq6}`
`querschnitte[autobahn:4]`	`mq2`
`querschnitte{autobahn:4}`	`{mq2, mq5, mq6}`
`querschnitte[autobahn:5]`	`ContainerZugriffsFehler`
`querschnitte{autobahn:5}`	`{ }`
`querschnitte[typ: "radar",spuren:2]`	`mq3`

Ausdruck	Ergebnis
`querschnitte{typ: "radar",spuren:2}`	`{mq3}`

2.6.1.3 Iteratoren auf Feldern und Listen

Iteratoren sind Objekte, die zum Navigieren und Iterieren innerhalb eines Containers benutzt werden können. Sie referenzieren i.a. ein Element des Containers enthalten aber zusätzlich genügend Informationen, um auf benachbarte Elemente zuzugreifen. Mit Hilfe von Iteratoren können manche Algorithmen auf Containern einfacher implementiert werden (einfachere Anwendung), zum anderen sind bestimmte Operationen wie Positionieren, Einfügen und Löschen effizienter realisiert, weil Iteratoren interne Zusatzinformationen enthalten, die bestimmte Operationen (z.B. Suche nach einem Elementen mit einem bestimmten Index in einer Liste) überflüssig machen (effizientere Implementierung).

2.6.1.3.1 Konstruktion von Iteratoren

Ausgehend von einem Container können durch folgende Methoden Iteratoren des Containers erzeugt werden.

2.6.1.3.1.1 Iterator vor das erste Element

```
iterator= container.anfang()
```

Die Methode `anfang()` eines Containers liefert einen Iterator zurück, der vor das erste Element zeigt.

2.6.1.3.1.2 Iterator auf das erste Element

```
iterator= container.erstes()
```

Die Methode `erstes()` eines Containers liefert einen Iterator zurück, der auf das erste Element zeigt.

2.6.1.3.1.3 Iterator auf beliebiges Element

```
iterator= container.iterator(index)
```

Die Methode `iterator()` eines Containers liefert einen Iterator zurück, der auf das Element mit dem Index `index` zeigt. Negative Indizes spezifizieren Elemente des Containers vom Ende aus gesehen. Ein Index von 0 positioniert den Iterator vor das erste Element. Ein Index, der größer als die Anzahl Elemente ist positioniert den Iterator hinter das letzte Element des Containers. Ein Index der kleiner als die negative Anzahl der Elemente ist positioniert den Iterator vor das erste Element.

2.6.1.3.1.4 Iterator auf das letzte Element

```
iterator= container.letztes()
```

Die Methode `letztes()` eines Containers liefert einen Iterator zurück, der auf das letzte Element positioniert ist.

2.6.1.3.1.5 Iterator hinter das letzte Element

```
iterator= container.schluss()
```

Die Methode `schluss()` eines Containers liefert einen Iterator zurück, der hinter das letzte Element positioniert ist.

2.6.1.3.2 Attribute und Operationen auf Iteratoren

2.6.1.3.2.1 Abfrage des zugehörigen Containers

```
container= iterator.container()
```

Die Methode `container()` liefert den container der dem Iterator zugeordnet ist, zurück.

2.6.1.3.2.2 Abfragen des referenzierten Elements

```
element= iterator.element()
```

Die Methode `element()` liefert das Element des Containers zurück, auf das der Iterator positioniert ist. Wenn der Container vor das erste Element oder hinter das letzte Element positioniert ist, dann liefert die Methode einen `ContainerZugriffsFehler` zurück.

2.6.1.3.2.3 Ändern des referenzierten Elements

```
altesElement= iterator.element(neuesElement)
```

Die Methode `element()` ersetzt das bisherige Element auf das der Iterator zeigt durch ein neues Element. Die Methode liefert das ursprüngliche Element zurück. Wenn der Container vor das erste Element oder hinter das letzte Element positioniert ist, dann liefert die Methode einen `ContainerZugriffsFehler` zurück.

2.6.1.3.2.4 Bestimmung des Vorgängers

```
iterator= iterator.vorgänger()
```

Die Methode `vorgänger()` liefert einen neuen Iterator zurück, der auf das Element innerhalb des Containers, das unmittelbar vor dem Element des ursprünglichen Iterators liegt, positioniert ist. Wenn der ursprüngliche Iterator hinter das letzte Element zeigt, liefert die Methode einen Iterator auf das letzte Element zurück. Wenn der ursprüngliche Iterator auf das erste Element zeigt, liefert die Methode einen Iterator vor das erste Element zurück. Wenn der ursprüngliche Iterator vor das erste Element zeigt, liefert die Methode auch einen Iterator vor das erste Element zurück. Der ursprüngliche Iterator bleibt durch die Methode unverändert.

2.6.1.3.2.5 Bestimmung des Nachfolgers

```
iterator= iterator.nachfolger()
```

Die Methode `nachfolger()` liefert einen neuen Iterator zurück, der auf das Element innerhalb des Containers, das unmittelbar nach dem Element des ursprünglichen Iterators liegt, positioniert ist. Wenn der ursprüngliche Iterator vor das erste Element zeigt, liefert die Methode einen Iterator auf das erste Element zurück. Wenn der ursprüngliche Iterator auf das letzte Element zeigt, liefert die Methode einen Iterator hinter das letzte Element zurück. Wenn der ursprüngliche Iterator hinter das letzte Element zeigt, liefert die Methode auch einen Iterator hinter das letzte Element zurück. Der ursprüngliche Iterator bleibt durch die Methode unverändert.

2.6.1.3.2.6 Positionierung auf den Vorgänger

```
elementVorhanden= iterator.zurück()
```

Die Methode `zurück()` positioniert den Iterator auf den Vorgänger des Iterators (siehe Kapitel 2.6.1.3.2.4 "Bestimmung des Vorgängers"). Die Methode liefert einen Wahrheitswert zurück, der angibt, ob der Iterator innerhalb des Cointainers positioniert ist (siehe Kapitel 2.6.1.3.2.11 "Ist der Iterator auf ein Element innerhalb des Containers positioniert?").

2.6.1.3.2.7 Positionierung auf den Nachfolger

```
elementVorhanden= iterator.weiter()
```

Die Methode `weiter()` positioniert den Iterator auf den Nachfolger des Iterators (siehe Kapitel 2.6.1.3.2.5 "Bestimmung des Nachfolgers"). Die Methode liefert einen Wahrheitswert zurück, der angibt, ob der Iterator innerhalb des Cointainers positioniert ist (siehe Kapitel 2.6.1.3.2.11 "Ist der Iterator auf ein Element innerhalb des Containers positioniert?").

2.6.1.3.2.8 Einfügen eines Elements

```
neuerIterator= iterator.einfügen(neuesElement)
```

Die Methode `einfügen()` fügt ein neues Element vor dem Element, auf das der Iterator positioniert ist, in den Container ein. Wenn der Iterator vor dem ersten Element positioniert ist, dann wird das neue Element als erstes Element in den Container eingefügt. Wenn der Iterator hinter das letzte Element des Containers zeigt, dann wird das neue Element als letztes Element in den Container

eingefügt. Der ursprüngliche Iterator behält seine Position bei. Die Methode liefert einen neuen Iterator zurück, der auf das eingefügte Element positioniert ist.

2.6.1.3.2.9 Anhängen eines Elements

```
neuerIterator = iterator.anhängen(neuesElement)
```

Die Methode `anhängen()` fügt ein neues Element hinter dem Element, auf das der Iterator positioniert ist, in den Container ein. Wenn der Iterator vor dem ersten Element positioniert ist, dann wird das neue Element als erstes Element in den Container eingefügt. Wenn der Iterator hinter das letzte Element des Containers zeigt, dann wird das neue Element als letztes Element in den Container eingefügt. Der ursprüngliche Iterator behält seine Position bei. Die Methode liefert einen neuen Iterator zurück, der auf das angehangene Element positioniert ist.

2.6.1.3.2.10 Löschen eines Elements

```
element= iterator.löschen()
```

Die Methode `löschen()` entfernt das Element, auf das der Iterator positioniert ist, aus dem Container. Das entfernte Element wird von der Methode zurückgegeben. Wenn der Iterator nicht innerhalb des Containers positioniert ist, liefert die Methode einen `ContainerZugriffsFehler` zurück. Durch eine erfolgreiche Lösch-Operation wird der Iterator nicht verändert, d.h. er ist nach wie vor auf das Element das gelöscht wurde positioniert. Die Methode `element()` kann weiterhin zum Abrufen des Elements benutzt werden. Da dieses Element nicht mehr zum Container gehört, liefert die Methode `innerhalb()` nach dem Löschen allerdings den Wert `falsch` zurück. Der Methoden `vorgänger()` bzw. `nachfolger()` liefern Iteratoren auf die Elemente zurück, die vor dem Löschen Vorgänger bzw. Nachfolger gewesen sind. Nach dem Löschen kann der Iterator mit den Methoden `zurück()` und `weiter()` auf die Elemente positioniert werden, die vor dem Löschen Vorgänger bzw. Nachfolger waren.

2.6.1.3.2.11 Ist der Iterator auf ein Element innerhalb des Containers positioniert?

```
innerhalb= iterator.innerhalb()
```

Die Methode `innerhalb()` liefert `wahr` zurück, wenn der Iterator auf ein Element innerhalb des Containers positioniert ist. Wenn der Iterator vor das erste oder hinter das letzte Element zeigt oder auf ein gelöschtes Element des Containers, dann liefert die Methode `falsch` zurück.

2.6.2 Schlüsselmengen

Schlüsselmengen (assoziative Arrays) bieten die Möglichkeit ein Schlüsselobjekt zusammen mit einem Datenobjekte zu speichern. Auf die Datenobjekte kann mit Angabe des Schlüsselobjekts zugegriffen werden.

2.6.2.1 Konstruktion von Schlüsselmengen

Eine neue Schlüsselmenge kann durch Aufruf der Funktion `Schlüsselmenge()` konstruiert werden.

```
s= Schlüsselmenge()
```

2.6.2.2 Operationen

Mit dem folgenden Konstrukt kann auf ein beliebiges Element einer Schlüsselmenge zugegriffen werden:

```
s[schlüsselobjekt]= datenobjekt     # schreibender Zugriff
datenobjekt= s[schlüsselobjekt]     # lesender Zugriff
```

Als `schlüsselobjekt` kann ein beliebiges Objekt übergeben werden.
Die einzelnen Elemente verhalten sich wie Variablen, das heißt man kann ihnen Werte (Datenobjekte) zuweisen (schreibender Zugriff) und man kann die Werte (Datenobjekte) abfragen um sie in weiteren Berechnungen zu verwenden (lesender Zugriff).

Ein lesender Zugriff mit einem Schlüsselobjekt, das nicht in der Schlüsselmenge enthalten ist, führt zu einem `ContainerZugriffsFehler`. Ein schreibender Zugriff mit einem Schlüsselobjekt, das nicht in der Schlüsselmenge enthalten ist, fügt ein neues Datenobjekt in die Schlüsselmenge ein.

```
nachfolger= Schlüsselmenge()
nachfolger["mq1"]= "mq2"
nachfolger["mq2"]= "mq7"
nachfolger["mq7"]= "mq4"
vorgänger= Schlüsselmenge()
vorgänger["mq2"]= "mq1"
vorgänger["mq7"]= "mq2"
vorgänger[nachfolger["mq7"]]="mq7"
ausgabe nachfolger["mq4"]                          # gibt einen Fehler aus
ausgabe nachfolger["mq2"]                          # gibt mq7 aus
ausgabe nachfolger[nachfolger["mq2"]]              # gibt mq4 aus
ausgabe vorgänger[nachfolger[nachfolger["mq2"]]] # gibt mq7 aus
```

2.6.2.2.1 Einfügen von Elementen

```
container.einfügen( schlüssel, element)
```

Die Methode `einfügen` fügt den Parameter `element` in den Container ein. Mit dem Parameter `schlüssel` wird das Schlüsselobjekt angegeben unter dem das Datenobjekt `element` gespeichert wird. Wenn schon ein Element unter dem angegebenen Schlüssel gespeichert ist, dann wird der Container nicht verändert und ein `ContainerZugriffsFehler` zurückgegeben.

2.6.2.2.2 Suchen von Elementen

```
element= container.element( schlüssel)
```

Mit der Methode `element` kann ein Container nach einem Element durchsucht werden, das mit dem Schlüsselobjekt `schlüssel` in den Container eingefügt wurde. Das gefundene Objekt wird von der Funktion zurückgeliefert. Wenn der Schlüssel nicht gefunden wurde, dann wird ein `ContainerZugriffsFehler` zurückgegeben.

2.6.2.2.3 Löschen von Elementen

```
element= container.löschen( schlüssel)
```

Die Methode `löschen()` entfernt das Element mit dem Schlüssel `schlüssel` aus dem Container. Das aus dem Container entfernte Element wird von der Methode zurückgeliefert. Wenn der Schlüssel nicht gefunden wurde, dann wird ein `ContainerZugriffsFehler` zurückgegeben.

2.6.2.2.4 Selektion von Objekten

Durch Anwendung eines Selektionskonstrukts können aus den Elementen eines Containers Objekte oder Datensätze ausgewählt werden, die bestimmte Attribute mit bestimmten Werten besitzen. Es wird zwischen Mehrfach-Selektion und Einzelselektion unterschieden.
Syntax und Semantik der Selektionskonstrukte für Schlüsselmengen entspricht den entsprechenden Konstrukten für Listen und Felder (siehe Kapitel 2.6.1.2.6 "Selektion von Objekten").

2.6.2.3 Iteratoren auf Schlüsselmengen

Iteratoren sind Objekte, die zum Navigieren und Iterieren innerhalb eines Containers benutzt werden können. Sie referenzieren i.a. ein Element des Containers enthalten aber zusätzlich genügend Informationen, um auf benachbarte Elemente zuzugreifen.
Die Reihenfolge der Elemente ist in Schlüsselmengen durch die implizite Sortierreihenfolge der Schlüsselobjekte vorgegeben.

2.6.2.3.1 Konstruktion von Iteratoren

Zur Konstruktion von Iteratoren in Schlüsselmengen stehen die gleichen Operatoren wie bei Feldern und Listen zur Verfügung (siehe Kapitel 2.6.1.3.1 "Konstruktion von Iteratoren").

```
iterator= container.anfang()
iterator= container.erstes()
iterator= container.iterator(schlüssel)
iterator= container.letztes()
iterator= container.ende()
```

Der einzige Unterschied ist bei der Methode iterator() gegeben. Diese erhält als Parameter nicht einen Index (wie bei Listen und Feldern) sondern einen Schlüssel. Der Iterator wird auf das Element mit dem gegebenen Schlüssel positioniert.

2.6.2.3.2 Attribute und Operationen auf Iteratoren

2.6.2.3.2.1 Analogie zu Listen und Feldern

Folgende Operationen auf Iteratoren von Schlüsselmengen verhalten sich analog zu den entsprechenden Methoden auf Iteratoren von Feldern und Listen (siehe Kapitel 2.6.1.3.2 "Attribute und Operationen auf Iteratoren"):

```
container= iterator.container()
element= iterator.element()
altesElement= iterator.element(neuesElement)
iterator= iterator.vorgänger()
iterator= iterator.nachfolger()
elementVorhanden= iterator.zurück()
elementVorhanden= iterator.weiter()
element= iterator.löschen()
innerhalb= iterator.innerhalb()
```

2.6.2.3.2.2 Keine Analogie beim Einfügen und Anhängen

Für die Methoden `einfügen()` und `anhängen()` gibt es bei Schlüsselmengen keine Analogie, da die Position eines Elements in einer Schlüsselmenge einzig und allein durch das Schlüsselobjekt (und die implizite Sortierreihenfolge der Schlüsselobjekte) gegeben ist.

2.6.2.3.2.3 Zugriff auf das Schlüsselobjekt

Als zusätzliche Methode haben die Iteratoren von Schlüsselmengen die Möglichkeit auf das Schlüsselobjekt zuzugreifen.

```
schlüssel= iterator.schlüssel()
```

Die Methode Schlüssel liefert den Schlüssel des Elements zurück, auf das der Iterator positioniert ist. Wenn der Iterator vor das erste Element oder hinter das letzte Element positioniert ist, liefert die Methode einen `ContainerZugriffsFehler` zurück. Die Methode kann auch benutzt werden um über einen Iterator den Schlüssel eines zuvor mit der Methode `iterator.löschen()` aus dem Container entfernten Elements zu bestimmen.
Die Schlüsselobjekte von Elementen eines Containers können nicht verändert werden, da sich dadurch die Position im Container verändern würde.

2.7 Spezielle Objekte

In der Skriptsprache zur Umfassenden Datenanalyse existieren spezielle Objekte als Container für den Zugriff aug globale Daten, die Konfiguration und die Parameter mit denen ein Skript ausgeführt wurde.

2.7.1 "global" – Objekt

In diesem Objekt sind alle globalen Variablen des Skriptes zusammengefasst. Dabei kann über die Syntax

```
global.a = 42
```

beispielsweise der globalen Variable a der Wert 42 zugewiesen werden.
Der Zugriff auf das „global“-Objekt ist von jeder beliebigen Stelle des Skripts möglich.

2.7.2 "konstant" – Objekt

In diesem Objekt sind alle Konstanten des Skriptes zusammengefasst. Dabei kann über die Syntax

```
konstant.a = 42
```

beispielsweise der Konstanten a der Wert 42 zugewiesen werden. Die Zuweisung einer Konstanten ist natürlich nur einmal möglich. Der Interpreter prüft zur Laufzeit, ob einer Konstanten mehrfach ein Wert zugewiesen wurde und bricht die Programmausführung für den Fall, dass einer Konstanten unterschiedliche Werte zugewiesen wurden, mit einer entsprechenden Fehlermeldung ab. Konstanten sind global verfügbar.
Folgende Konstanten sind in der Umfassenden Datenanalyse vordefiniert:

Konstante	Beschreibung
konstant.pi	π (3.141593...)
konstant.e	Eulersche Konstante e (2.718282...)
konstant.GrößteZahl	Größte darstellbare Fließkommazahlen
konstant.GrößteGanzeZahl	Größte darstellbare ganze Zahl
konstant.KleinsteZahl	Kleinste darstellbare Fließkommazahlen
konstant.KleinsteGanzeZahl	Kleinste darstellbare ganze Zahl

2.7.3 "parameter" – Objekt

In diesem Objekt sind alle Aufrufparameter des Skriptes zusammengefasst. Dabei kann über die Syntax

```
# Inhalt des Aufrufparameters –konfiguration=... auslesen
uzIdentifikation= parameter.konfiguration
```

beispielsweise auf den Aufrufparameter -konfiguration="UZ Leverkusen" lesend zugegriffen und der Variablen `uzIdentifikation` zugewiesen werden.

Die übergebenen Parameter werden in der Reihenfolge „Ganze Zahl“, „Fliesskommazahl“, „Zeichenkette“ interpretiert und innerhalb des „parameter“-Objekts bereitgestellt.

2.7.4 "konfiguration" – Objekt

Das „konfiguration“-Objekt stellt ein virtuelles Abbild der Konfiguration des Datenverteilers dar. Es bietet einen einfachen Zugriff auf alle benötigten Informationen der Systemkonfiguration.

Ausgehend vom Schlüsselwort **konfiguration** kann der Benutzer auf alle im Abschnitt „Zugriff auf den Datenverteiler“ beschriebenen Informationen zur Systemkonfiguration zugreifen.

2.8 Fehler und Fehlerbehandlung

Zur Fehlerbehandlung werden in der Umfassenden Datenanalyse Fehler-Objekte eingesetzt. Die Fehlerobjekte werden bei entsprechendem Fehlverhalten automatisch erzeugt und mit den notwendigen Informationen initialisiert. Über diese Objekte können zur Laufzeit Fehler erkannt und ausgewertet werden.
Ein Fehlerobjekt setzt sich aus folgenden Attributen zusammen:

Attribut	Bedeutung
typ	Mit dem Attribut typ wird der Fehlertyp klassifiziert.
zusatztext	Beschreibender Zusatztext zum Fehler.
funktion	Hier wird, für den Fall, dass der Fehler innerhalb einer Funktion aufgetreten ist, die Funktion angegeben.
skript	Name des Skripts, in dem der Fehler aufgetreten ist.
zeile	Zeile im Skript, wo der Fehler aufgetreten ist.

Attribut	Bedeutung
ursprünglicherFehler	Mit diesem Attribut wird der vorherige Fehler referenziert, falls ein Fehlerobjekt den Fehler ausgelöst hat.

Tabelle 2-3: Attribute eines Fehler-Objektes

Die Art des Fehlers lässt sich in verschiedene Typen klassifizieren. Die folgende Tabelle gibt einen Überblick der möglichen Fehlertypen, die zum Teil noch feiner unterteilt sind. Die, in der Spalte "Weitere Unterteilung", fett gedruckten Texte werden in der Fehlerbeschreibung als Zusatztext aufgeführt.

Fehlertyp	Weitere Unterteilung
ArithmetischerFehler	Unter diesem Fehlertyp sind arithmetische Fehler zusammengefasst: • **Division durch 0** • **Overflow** • etc.
SymbolUndefiniertFehler	Unter diesem Fehlertyp sind Zugriffe auf undefinierte Symbole zusammengefasst: • **Variable nicht definiert** Dieser Fehler tritt auf, wenn auf eine Variable zugegriffen wird, der noch nichts zugewiesen wurde. • **Funktion nicht definiert** Es wurde eine unbekannte Funktion aufgerufen.
ObjektZugriffsFehler	Unter diesem Fehlertyp sind Zugriffe auf nicht existente Methoden oder Elemente von Objekten zusammengefasst: • **Methode nicht definiert** Dieser Fehler tritt auf, wenn zu einem konkreten Objekt die angegebene Methode nicht existiert. • **Element nicht definiert** Dieser Fehler tritt auf, wenn zu einem konkreten Objekt das angegebene Element nicht existiert.
ContainerZugriffsFehler	Unter diesem Fehlertyp sind falsche Containerzugriffe und fehlerhafte Selektionen zusammengefasst: • **Containerzugriff mit ungültigem Index** Dieser Fehler tritt auf, wenn der Index außerhalb des gültigen Bereichs liegt. • **Containerselektion ohne Ergebnis** Dieser Fehler tritt auf, wenn z.B. bei einem Konfigurationszugriff eine unbekannte pid angegeben wurde `uz= konfiguration.objekte[pid:"UZ GibtsNicht"]`

Fehlertyp	Weitere Unterteilung
ArgumentFehler	Unter diesem Typ sind Fehler bei der Argumentübergabe zusammengefasst: • **Falsche Anzahl von Argumenten** Dieser Fehler tritt auf, wenn die Anzahl der übergebenen Argumente nicht mit der geforderten Anzahl übereinstimmt. • **Funktion oder Operator nicht für diesen Argumenttyp definiert** Dieser Fehler tritt auf, wenn eine Funktion oder ein Operator für die konkreten Typen eines Arguments nicht definiert ist. z.B. x= "hallo" * 4 • **Argument außerhalb eines definierten Wertbereichs** Dieser Fehler tritt auf, wenn der Wert des Arguments außerhalb des definierten Wertbereichs liegt. z.B. log(-4)

Fehlertyp	Weitere Unterteilung
DatenZugriffsFehler	Unter diesem Typ sind fehlerhafte Zugriffe auf Daten zusammengefasst: • **falsche Attributgruppe** Dieser Fehler tritt auf, wenn zu einem Objekt eine Attributgruppe gelesen oder geschrieben werden soll, die nicht für diesen Objekttyp vorgesehen ist. • **falscher Aspekt** Dieser Fehler tritt auf, wenn zu einem Objekt eine Attributgruppe unter einem nicht zulässigen Aspekt gelesen oder geschrieben werden soll. • **keine Rechte** Dieser Fehler tritt auf, wenn der Zugriff auf Daten für dieses Objekt aufgrund fehlender Zugriffsrechte verwehrt wurde. • **keine Quell-Applikation angemeldet** Dieser Fehler tritt auf, wenn bei einer Leseanmeldung oder beim Lesen ohne explizite Anmeldung festgestellt wird, dass zur Zeit keine Quelle für die gewünschten Daten zur Verfügung steht. • **Quell-Applikation kann keine Daten liefern** Dieser Fehler tritt auf, wenn bei einer Leseanmeldung oder beim Lesen ohne explizite Anmeldung festgestellt wird, dass die Quelle zur Zeit die gewünschten Daten nicht liefern kann. • **Archivzugriff gescheitert** Dieser Fehler tritt auf, wenn keine Verbindung zur Datenhaltung hergestellt werden konnte oder wenn diese die gewünschten Daten nicht liefern konnte. • **Zugriff auf undefinierte Attribute** Dieser Fehler tritt auf, wenn auf ein Attribut aus einem gelesenen Datensatz zugegriffen wird das z.B. "fehlerhaft" oder "nicht ermittelbar" ist. • **Zugriff auf ein nicht vorhandenes Attribut** Dieser Fehler tritt auf, wenn bei einem gelesenen Datensatz auf ein nicht vorhandenes Attribut zugegriffen wird. • **keine Senke angemeldet** Dieser Fehler tritt auf, wenn bei einer Schreibanmeldung als Sender oder beim Schreiben ohne explizite Anmeldung festgestellt wird, dass zur Zeit keine Senke für die gewünschten Daten zur Verfügung steht.
KonfigurationsFehler	Unter diesem Typ sind fehlerhafte Zugriffe auf Konfigurationsobjekte zusammengefasst: • **Zugriff auf nicht existierende Konfigurationsobjekte** Dieser Fehler tritt auf, wenn versucht wird auf Objekte zuzugreifen, die in der angesprochenen Konfiguration nicht definiert sind. • **Änderung von nichtdynamischen Mengen versucht** Dieser Fehler tritt auf, wenn versucht wird, Elemente aus Mengen zu entfernen oder in Mengen zu stecken, die in der angesprochenen Konfiguration nicht als dynamische Mengen definiert sind.
Fehler	Unter diesem Fehlertyp sind sonstige nicht näher klassifizierte Fehler zusammengefasst.

2.8.1 Erzeugung neuer Fehler-Objekte

Die Fehlerobjekte werden bei entsprechendem Fehlverhalten automatisch erzeugt und mit den notwendigen Informationen initialisiert. Neben der automatischen Erzeugung können die Fehlerobjekte in der Umfassenden Datenanalyse auch bei Bedarf konstruiert werden. Dazu stehen drei Möglichkeiten zur Verfügung:

- *fehlertyp*(*zusatzText*, *ursprünglicherFehler*)
- *fehlertyp*(*zusatzText*)
- *fehlertyp*()

Dabei steht *fehlertyp* für den Namen der Funktion, der einen in der Spalte für die Fehlertypen aufgeführten Namen annehmen kann. Je nach Auswahl wird der entsprechende Fehlertyp klassifiziert (z.B. `ArgumentFehler()` erzeugt ein neues Fehlerobjekt vom Typ ArgumentFehler). Als Argument kann dem Fehlerobjekt ein Text übergeben werden, der in das Attribut `zusatztext` übernommen wird (z.B. `ArgumentFehler("Argument nicht vom Typ Zahl")` erzeugt eine neues Fehlerobjekt vom Typ ArgumentFehler; das Attribut `zusatztext` enthält dann beispielsweise "Argument nicht vom Typ Zahl"). Als weiteres Argument kann für den Fall eines Folgefehlers der ursprüngliche Fehler, d.h. der Fehler, der zu diesem neuen Fehler geführt hat, übergeben werden.

2.8.2 Abfrage von Fehlern

Zur Laufzeit kann in der Umfassende Datenanalyse geprüft werden, ob an einer Stelle im Skript ein Fehler-Objekt zurückgegeben wurde. Dazu steht die Funktion `fehler(x)` zur Verfügung, der als Argument ein Ausdruck übergeben wird. Wenn der übergebene Ausdruck ein Fehlerobjekt darstellt gibt die Funktion den Wert `wahr` zurück. Das folgende Beispiel erläutert den Gebrauch der Fehlerprüfungsfunktion:

```
funktion quadrat(x)

  wenn fehler(x) dann
    # Fehlerbehandlung
    rückgabe ArgumentFehler("Fehler als Argument übergeben",x)
  ende wenn
  wenn x.klasse <> "Zahl" dann
    # Fehlerbehandlung
    rückgabe ArgumentFehler("Argument nicht vom Typ Zahl")
  sonst
    rückgabe x*x
  ende wenn
ende funktion
```

2.8.3 Abfrage des Fehlermeldungstextes

Die Methode `meldung()` eines Fehlers liefert eine Zeichenkette zurück, die in Textform sämtliche Attribute des Fehlers enthält. Falls im Attribut `ursprünglicherFehler` auf einen anderen Fehler referenziert, wird auch dieser durch einen rekursiven Aufruf der Methode in eine Zeichenkette umgewandelt, d.h. das Ergebnis enthält schließlich eine Beschreibung der ganzen Fehlerkette bis zum ersten Fehler der Kette.

```
fehlermeldung= fehler.meldung()
fehlermeldung= fehler.meldung(anzahlUrsprünglicherFehler)
```

Wenn an die Methode `meldung(...)` zusätzlich eine Zahl als Parameter übergeben wird, dann gibt diese die maximale Anzahl der in Text zu konvertierenden Fehler in der Fehlerkette an. Der Wert `0` führt zur gleichen Verhaltensweise wie ein Aufruf ohne Parameter.

2.9 Ausdrücke

2.9.1 Allgemein

Ausdrücke sind Variablen, Konstanten, Parameter, andere spezielle Objekte oder komplexere Ausdrücke, die durch Verknüpfung einfacher Ausdrücke über Operatoren, Quantoren etc. gebildet werden.
In der Skriptsprache zur Umfassenden Datenanalyse kann ein Ausdruck ein Basisausdruck, ein erweiterter Ausdruck, ein Negationsausdruck, ein Multiplikationsausdruck, ein Additionsausdruck, ein Vergleichsausdruck, ein Gleichheitsausdruck, ein Und-Ausdruck, ein Oder-Ausdruck oder ein Quantor-Ausdruck sein.

Basisausdruck

Unter einem Basisausdruck werden

- Konstanten
  - Zahlen 123.34
  - Zeichenketten "Staumanagement"
- Variablen x
- Funktionsaufruf f(...)
- Geklammerte Ausdrücke (...)

verstanden.

Erweiterter Ausdruck

Zu den erweiterten Ausdrücken zählen:

- Containerzugriffe a[...]
- Objektelement zugriffe objekt.element
- Methodenaufrufe objekt.methode(...)

Negationsausdruck

Zu den Negationsausdrücken gehört die arithmetische und logische Negation:

- Arithmetische Negation - 5
- Logische Negation nicht ...

Multiplikationsausdruck

Die Multiplikationsausdrücke werden durch die Multiplikation, Division, Ganzzahl-Division und Modulo-Operation beschrieben:

- Multiplikation x * y
- Division x / y
- Ganzzahl-Division x div y
- Modulo-Operation x modulo y

Additionsausdruck

Hier sind Addition, Subtraktion und Zeichenkettenkonkatenation zusammengefasst.

- Addition x + y
- Subtraktion x – y
- Zeichenkettenkonkatenation “x“ & “y“

Vergleichsausdruck

- Kleiner x < y
- Kleiner-Gleich x <= y
- Größer x > y
- Größer-Gleich x >= y

Gleichheitsausdruck

- Gleich x = y
- Ungleich x <> y

Und-Ausdruck

- a und b

Oder-Ausdruck

- a oder b

Quantor-Ausdruck

für <elemente einer menge> gilt eine <bedingung>

Ein Ausdruck repräsentiert immer einen Wert innerhalb des Skriptes, der Wert eines Ausdrucks wird durch die Auswertung desselben ermittelt.

2.9.2 Operatoren

Operatoren sind Sprachelemente der Umfassenden Datenanalyse, die andere Ausdrücke miteinander verknüpfen und so wieder einen komplexeren Ausdruck bilden. Die unten stehende Tabelle zeigt die in UDA zur Verfügung stehenden Operatoren mit ihrer Priorität, d.h. der Reihenfolge, in der sie die ihnen zugeordneten Ausdrücke zusammenfassen.

Operator	P	Bedeutung	Beispiel
[] .	1	Containerzugriff Elementzugriff	a[5] objekt.element
- nicht	2	Vorzeichen logisches Komplement	-210 nicht x <> wahr
* / div modulo	3	Multiplikation Division Ganzzahldivision Divisionsrest	a * y a / y x div y x modulo y
+ - &	4	Addition Subtraktion Zeichenkettenkonkatenation	x + 5 x – 5 ("hallo" & "Welt")
< <= >= >	5	kleiner kleiner gleich größer gleich größer	x < y x <= y x >= y x > y
= <>	6	gleich ungleich	x = y x <> y
und	7	logische Und-Verknüpfung	x < y und y < z
oder	8	logische Oder-Verknüpfung	x < y oder y < z

Operator	P	Bedeutung	Beispiel
für ..	9	Quantoren	für alle x in M gilt x < y

2.9.3 Quantoren

Als Quantoren werden in der Umfassenden Datenanalyse spezielle Operatoren bezeichnet, die als Operanden einen Ausdruck, der einen Container repräsentiert, und einen Bedingungsausdruck verknüpfen. Das Ergebnis der Verknüpfung ist ein Boolscher Wert
Der Begriff Quantor ist insofern erweitert, das der übergebene Container nicht nur als Menge betrachtet wird, sondern auch Felder und Listen umfassen kann, in denen die Reihenfolge der darin enthaltenen Objekte eine Bedeutung erlangt, so dass auch Quantoren wie: „FürJedenDritten“, … definiert werden können.

Folgende Quantoren sind in der Umfassenden Datenanalyse vordefiniert:

- Für alle
- Für ein
- Für mindestens
- Für höchstens

Jeder Quantor-Ausdruck wird mit dem Schlüsselwort **für** eingeleitet. Es folgen, je nach Ausprägung die Schlüsselwörter:

für **fuer**	→	**jede**, **jedes**, **jeden** oder **alle** wenn die Bedingung für jedes Element der Menge erfüllt sein muss	→	*ObjektName* **in** *MengeName* *ObjektName* **in der** *MengeName* *ObjektName* **in dem** *MengeName* **der elemente** *ObjektName* **in** *MengeName* **der elemente** *ObjektName* **in der** *MengeName* **der elemente** *ObjektName* **in dem** *MengeName* **der** *ObjektName* **in** *MengeName* **der** *ObjektName* **in der** *MengeName* **der** *ObjektName* **in dem** *MengeName*
		ein, **eine** oder **einen** wenn die Bedingung für ein Element der Menge erfüllt sein muss		
		mindestens x oder **mindestens** x % wenn die Bedingung für mindestens x Elemente der Menge oder x % der Menge erfüllt sein muss		
		höchstens x oder **höchstens** x **%** wenn die Bedingung für höchstens x Elemente der Menge oder x % der Menge erfüllt sein muss		

Damit ist die zu untersuchende Menge von Objekten und der betrachtete Anteil der Objekte aus dieser Menge, der die folgende Bedingung erfüllen muss, spezifiziert. Die Vielzahl der Schlüsselwortvarianten ergibt sich aus der Forderung, dass der Quantor-Ausdruck möglichst umgangssprachlich formuliert werden soll.
Es folgt der Bedingungsteil des Quantors, der für den spezifizierten Anteil der Menge erfüllt sein muss:

(**die**) **bedingung** *Ausdruck* **erfüllt** (**ist**)
gilt *Ausdruck*

Wobei *Ausdruck* ein weiterer Quantor-Ausdruck sein kann (Schachteln von Quantoren).
Ist der Ausdruck `undefiniert` bzw. ein Fehlerobjekt, so entspricht dies einem *nicht* `wahr`.
Das folgende Beispiel zeigt verschiedene Quantoren, die in einer Wenn-Anweisung ausgewertet werden:

```
wenn für alle x in der Menge gilt f(x)>0 dann ..
wenn für alle x in der Menge die bedingung f(x)>0 erfüllt ist dann ..
wenn für jeden mq in der Menge gilt f(mq)>0 dann ..
wenn für einen mq in der Menge gilt f(mq)>0 dann ..
wenn für höchstens 123 der elemente x in Menge gilt f(x)>0 dann ..
wenn für mindestens 7 % der elemente x in Menge gilt f(x)>0 dann ..
```

Ein Quantor-Ausdruck gibt als Ergebnis `wahr` (d.h. Bedingung wird für die angegebene Anzahl von Objekten in der Menge erfüllt) oder `falsch` (d.h. Bedingung wird nicht für die angegebene Anzahl von Objekten in der Menge erfüllt) zurück.

Neben den vordefinierten Quantoren können innerhalb eine UDA-Skripts eigene Quantoren definiert werden (siehe Abschnitt: Erweiterung der Skriptsprache)

2.9.4 Basisfunktionen

Für numerische Berechnung stehen innerhalb der Umfassenden Datenanalyse folgende Funktionen zur Verfügung.

Potenzfunktionen

- potenz(basis, exponent)
- potenz(basis)
- wurzel (basis, exponent)
- wurzel (basis)

Die Funktionen ohne Angabe eines Exponenten nehmen implizit den Wert „2“ an. Die übergebenen Werte sind beliebige Zahlenwerte, d.h. die Wurzel könnte auch durch den Ausdruck potenz(4, 0.5) beschrieben werden. Das Ergebnis der Auswertung eine Potenzfunktion ist immer eine Fliesskommazahl.

Exponentialfunktion/Logarithmusfunktion

- exp(wert)
- ln(wert)

Die Funktionen liefern den Wert der Exponentialfunktion bzw. den natürlichen Logarithmus für die übergebenen Werte. Das Eregbnis ist immer eine Fleisskommazahl.

Trigonometrische Funktionen

- sin(wert)
- cos (wert)
- tan (wert)

Die Funktionen liefern die entsprechenden Werte der trigonometrischen Funktionen. Die Angabe des Wertes wird im Gradmaß erwartet. Um Berechnungen im Bogenmaß zu ermöglichen, stehen zusätzlich die Funktionen:

- inBogenmass(wert)
- inGrad (wert)

zur Verfügung, mit denen jeweils die Umrechnung von Grad in Bogenmaß oder umgekehrt ausgeführt werden kann.

Alle trigonometrischen Funktionen liefern als Ergebnis eine Fliesskommazahl.

Weitere Funktionen

- min (wert0, wert1, …….)
- max (wert0, wert1, …….)
- abs (wert)

Die Funktionen liefern das Maximum, das Minimum oder den Absolutwert der übergebenen numerischen Argumente. Die Anzahl der Argumente der Maximum- und Minimum-Funktion ist im Prinzip nicht begrenzt.
Das Eregbnis der hier aufgeführten Funktionen hängt vom Typ der übergebenen Argumente ab.

2.9.5 Vergangenheitsoperatoren (Trends)

Mit Hilfe von Vergangenheitsoperatoren kann festgestellt werden, ob der Verlauf eines Wertes in einem bestimmten Zeitbereich

- monoton steigend
- monoton fallend
- oder gleichbleibend

war.

Die Vergangenheitsoperatoren können auf Trendobjekte angewendet werden. Diese werden mit der Funktion `trend()` erzeugt.

```
t= trend("KFZ Anzahl, MQ4711")
```

Im Parameter kann der Funktion dabei optional ein Bezugsobjekt übergeben werden. Das Bezugsobjekt kann ein beliebiges Objekt beispielsweise ein Konfigurationsobjekt, eine Datenstruktur oder wie im obigen Beispiel ein Name sein. Eine Referenz auf das Bezugsobjekt wird im Trendobjekt gespeichert und kann mit der Methode `bezug()` wieder abgerufen werden.

```
ausgabe t.bezug()
```

Mit der Methode `wert()` kann (je nach Anzahl der Parameter) ein neuer Wert an das Trendobjekt übergeben werden oder der zuletzt übergebene Wert abgerufen werden.

```
mit fehlerausgabe  t.wert(30)
mit fehlerausgabe  t.wert(27, zeitstempel)
ausgabe            t.wert()
```

Bei Angabe von einem Parameter speichert die Methode den übergebenen Wert mit dem aktuellen Zeitpunkt. Über die Angabe eines Zeitstempels im zweiten Parameter kann auch ein historischer Wert gespeichert werden. Wenn die Methode mit einem Zeitstempel aufgerufen wird, der weiter in der Vergangenheit zurückliegt, als der zuletzt gespeicherte Wert, dann gibt die Methode einen `ArgumentFehler` zurück. Ohne Parameter liefert die Methode den zuletzt gespeicherten Wert zurück. Wenn noch kein Wert gespeichert wurde, wird `undefiniert` zurückgegeben.
Mit der Methode `zeit()` kann der Zeitstempel des zuletzt gespeicherten Wertes abgerufen werden.
Die Methoden `maxKonstant()`, `maxSteigend()` und `maxFallend()` gehen vom letzten Wert des Trends aus und zeigen an, über wie viele Werte die Folge konstant, monoton steigend bzw. monoton fallend war.

```
t = trend()
mit fehlerausgabe  t.wert(50)
mit fehlerausgabe  t.wert(50)
mit fehlerausgabe  t.wert(50)
mit fehlerausgabe  t.wert(40)
mit fehlerausgabe  t.wert(30)
a = t.maxKonstant()        # a = 1
a = t.maxSteigend()        # a = 1
a = t.maxFallend()         # a = 3
```

Wurden dem Trendobjekt noch keine Werte zugewiesen, so liefern alle drei Funktionen 0 zurück.
Mit den Vergangenheitsoperatoren, d.h. den Methoden steigend(), fallend() und konstant() kann abgefragt werden, ob die Veränderung des gespeicherten Wertes monoton steigend, fallend bzw. unverändert war. Dabei kann der jeweiligen Methode eine Anzahl von Intervallen oder eine relative Zeitangabe übergeben werden. Eine relative Zeitangabe bezeichnet dabei den Zeitraum, ausgehend vom aktuellen Zeitpunkt in die Vergangenheit, der betrachtet wird.

```
wenn t.fallend(5 Minuten) dann
   ausgabe t.bezug() & ": Wert über 5 Minuten fallend"
ende wenn
wenn t.steigend(5 Minuten) dann
    ausgabe t.bezug() & ": Wert über 5 Minuten steigend"
ende wenn
wenn t.konstant(5 Minuten) dann
    ausgabe t.bezug() & ": Wert über 5 Minuten konstant"
ende wenn

wenn t.fallend(5) dann
   ausgabe t.bezug() & ": Wert über 5 Intervalle fallend"
ende wenn
wenn t.steigend(5) dann
    ausgabe t.bezug() & ": Wert über 5 Intervalle steigend"
ende wenn
wenn t.konstant(5) dann
    ausgabe t.bezug() & ": Wert über 5 Intervalle konstant"
ende wenn
```

Wenn innerhalb der angegebenen Zeitspanne bzw. Anzahl Intervalle der gespeicherte Wert `undefiniert` war, liefern die Operatoren den Wert `falsch` zurück.
Zur Prüfung von historischen Daten kann den Vergangenheitsoperatoren optional ein Zeitstempel übergeben werden auf den sich die relative Zeitangabe bezieht.

```
wenn t.steigend(5 Minuten, zeitstempel) dann
    ausgabe t.bezug() & ": Wert war zum Zeitpunkt "
    ausgabe zeitstempel & " über 5 Minuten steigend"
ende wenn
```

Wenn der angegebene Zeitpunkt vor dem Zeitstempel des zuletzt gespeicherten Wertes liegt, liefert der Vergangenheitsoperator einen `ArgumentFehler` zurück.

2.9.6 Hysterese

Zur Klassifizierung von kontinuierlichen Eingangsgrößen in stufenorientierte Größen ist eine parametrierbare Klassifizierungsfunktion mit Hysterese vorzusehen. Dabei wird über die definierte Hysterese ein funktioneller Zusammenhang zwischen gemessener bzw. ermittelter Größe und neuer Stufe, in Abhängigkeit von der aktuellen Stufe spezifiziert.

2.9.6.1 Konstruktion einer Hysterese

Mit dem folgenden Konstrukt wird ein neues Hysterese-Objekt angelegt (die „\" deuten an, dass die Anweisung in der folgenden Zeile fortgesetzt wird).

```
h= Hysterese(\
   hystereseStufe(stufe1,0,x2),\
   hystereseStufe(stufe2,x1,x4),\
   hystereseStufe(stufe3,x3,255)\
)
```

Dabei wird jede Stufe der Hysterese mit ihrer Klassifizierung sowie dem Anfangs- und Endwert beschrieben (`hystereseStufe(`*`Klassifizierung`*`,`*`Anfangswert`*`,`*`Endwert`*`)`).
Der Zugriff auf die einzelnen Stufen erfolgt wie bei einem Container. Damit können die aktuellen Werte einfach ausgelesen oder neu gesetzt werden.

```
# Ausgabe des Anfangswertes der ersten Stufe:
ausgabe h.stufen[1].anfangswert

# Attribute der ersten Stufe einzeln ändern
h.stufen[1].stufe="kalt"
h.stufen[1].anfangswert=-30
h.stufen[1].endwert=25

# zweite und dritte Stufe anpassen
h.stufen[2]=hystereseStufe("warm",20,50)
h.stufen[3]=hystereseStufe("heiß",45,100)
```

Zur Erweiterung der Klassifizierung können zusätzliche Stufen in die Hysterese aufgenommen werden. Dazu stehen die beiden Methoden "einfügen" und "anhängen" zur Verfügung. Beide

Methoden erwarten als erstes Argument den Index (i), wo die Erweiterung erfolgen soll und als zweites Argument die Definition der zusätzlichen Hysteresestufe.

```
h.stufen.einfügen(i,hystereseStufe(Klassifizierung,Anfangswert,Endwert))
h.stufen.anhängen(i,hystereseStufe(Klassifizierung,Anfangswert,Endwert))
```

Beim Einfügen nimmt die neue Stufe den angegebenen Index an und alle weiteren Stufen werden um eine Position verschoben. Das Anhängen bewirkt, dass die neue Stufe hinter dem Eintrag der durch den Index identifizierten Stufe eingefügt wird und die folgenden Stufen um eine Position verschoben werden.
Wird ein Index verwendet, der größer als der größte bisher existierenede Index ist, so führt dies bei beiden Funktion dazu, dass die Stufe hinter dem letzten Eintrag angehägt wird. Wird der Index 0 oder ein negativer Index verwendet, so führt dies bei beiden Funktionen dazu, dass die Stufe vor dem ersten Eintrag eingefügt wird und alle Stufen um eine Position verschoben werden
Das folgende Beispiel zeigt die Erweiterung der Hysterese um die zusätzliche Stufe "lau".

```
# bisherige Stufen anpassen:
h.stufen[1]=hystereseStufe("kalt",-30,40)
h.stufen[2]=hystereseStufe("warm",20,60)
h.stufen[3]=hystereseStufe("heiß",50,100)
# neue Stufe "lau" vor Stufe "warm" einfügen:
mit fehlerausgabe h.stufen.einfügen(2,hystereseStufe("lau",25,45))
# damit ergibt sich:
ausgabe h.stufen[1].stufe # "kalt"
ausgabe h.stufen[2].stufe # "lau"
ausgabe h.stufen[3].stufe # "warm"
ausgabe h.stufen[4].stufe # "heiß"
```

2.9.6.2 Methoden der Hysterese

Letzten Zustand der Hysteresefunktion setzen oder abrufen: stufe(*stufe*)

Die Methode "stufe" setzt das Gedächtnis der Hysterese auf den angegebenen Wert. Neue Hysterese Objekte werden standardmäßig mit der ersten Stufe initialisiert. Als Rückgabewert wird die neu gesetzte Stufe oder für den Fall, dass die angegebene Stufe nicht definiert ist, ein Fehler (ArgumentFehler) zurückgegeben.

```
stufe = h.stufe("warm")    # setzt den Zustand der Hysterese auf "warm"
```

Die Methode "stufe" liefert die aktuelle Stufe zurück, wenn ihr kein Argument übergeben wurde:

```
stufe = h.stufe()          # liefert aktuelle Stufe zurück hier "warm"
```

Ermittelte Eingangsgröße in stufenorientierte Größen abbilden: klassifiziere(*x*)

Über die "klassifiziere"-Methode kann ein Eingangswert auf die stufenorientierte Größe abgebildet werden. Dazu wird jeweils der letzte Zustand der Hysterese (der zwischengespeichert ist) und der übergebene Wert x benötigt.

```
stufe= h.klassifiziere(x)
```

1. Wenn x im Bereich der letzten Stufe (zwischen Anfangs- und Endwert) liegt, bleibt die Stufe konstant.
2. Wenn x größer als der Endwert der letzten Stufe ist, wird die nächst höhere Stufe, deren Bereich den Wert x beinhaltet, als Ergebnis zurückgeliefert und diese Stufe bei der Hysterese als letzter Zustand vermerkt.
3. Wenn x kleiner als der Anfangswert der letzten Stufe ist, wird die nächst niedrigere Stufe, deren Bereich den Wert x beinhaltet als Ergebnis zurückgeliefert und diese Stufe bei der Hysterese als letzter Zustand vermerkt.
4. Wenn bei den Fällen 2. oder 3. keine definierte Stufe gefunden wurde, deren Bereich den Wert x enthält, wird ein Fehler (ArgumentFehler) zurückgegeben. Der Wert des letzten Zustands der Hysterese bleibt bestehen.

2.9.7 Fuzzy-Logik

Im Gegensatz zur Hysterese wird bei der Fuzzyfizierung von kontinuierlichen Eingangsgrößen keine klare Stufe zurückgeliefert, sondern es werden fließende (verwaschene) Übergänge zwischen den diskreten Ergebniszuständen (linguistische Terme) geliefert. Dies wird durch sogenannte Zugehörigkeitswerte der linguistischen Terme ausgedrückt. Linguistische Terme sind umgangssprachliche Bezeichnungen für bestimmte Attribute, die keine scharfen Grenzen haben wie z.B. "kalt", "kühl", "angenehm", "warm" und "heiß". Die Zugehörigkeit von Eingangswerten zu einem bestimmten linguistischen Term wird durch ein Fuzzy-Set beschrieben. Im Fuzzy-Set wird jedem Eingangswert eine Zugehörigkeit zu dem linguistischen Term in Form eines Wertes zugeordnet. Die Zugehörigkeitswerte liegen üblicherweise im Intervall [0,1] wobei 0 bedeutet, dass der Eingangswert nicht zum linguistischen Term gehört und 1 bedeutet, dass der Wert sicher zum jeweiligen linguistischen Term gehört. Mehrere linguistische Terme mit ihren jeweiligen Fuzzy-Sets werden zu einer linguistischen Variable zusammengefasst.

Bei der Fuzzyfizierung mit Hilfe von linguistischen Variablen wird einem Eingangswert ein Tupel zugeordnet, das für jeden linguistischen Term eine Zugehörigkeit angibt. Die einzelnen Zugehörigkeiten dieser sogenannten Fuzzywerte sind Zahlen und können mit den üblichen Operationen verarbeitet werden. Außerdem können spezielle Operationen der Fuzzy-Logik verwendet werden, um unscharfe logische Verknüpfungen durchzuführen. Ergebnisse dieser Verarbeitungen sind üblicherweise wieder Fuzzy-Werte von linguistischen Variablen, die zur Ausgabe dienen. Die Ergebnis-Fuzzywerte können mit Hilfe der Defuzzyfizierung bei Bedarf wieder in einzelne "scharfe" Werte (Zahlen) umgewandelt werden.

2.9.7.1 Konstruktion einer linguistischen Variablen

Mit folgendem Konstrukt wird eine neue linguistische Variable angelegt:

```
LinguistischeVariable(name, untereGrenze, obereGrenze, terme...)
```

Der Parameter `name` enthält die Bezeichnung der neuen linguistischen Variablen. Die untere Grenze des Wertebereichs, für den die linguistische Variable definiert sein soll, ist im Parameter `untereGrenze` enthalten. Die obere Grenze des Wertebereichs, für den die linguistische Variable definiert sein soll, ist im Parameter `obereGrenze` enthalten. Danach folgen die einzelnen linguistischen Terme, die folgendermaßen konstruiert werden können:

```
LinguistischerTerm(bezeichner, untereGrenze, obereGrenze, name)
```

Der Parameter `bezeichner` enthält den Bezeichner des neuen linguistischen Terms. Die untere Grenze des Bereichs, für den der linguistische Term eine Zugehörigkeit von eins haben soll, ist im Parameter `untereGrenze` enthalten. Die obere Grenze des Bereichs, für den der linguistische Term eine Zugehörigkeit von eins haben soll, ist im Parameter `obereGrenze` enthalten. `name` ist optional und enthält eine Zeichenkette, die dem linguistischen Term einen sprechenden Namen gibt. Wird `name` nicht angegeben, so wird `name` auf den gleichen Wert wie `bezeichner` gesetzt. Beispielsweise lässt sich die in **Fehler! Verweisquelle konnte nicht gefunden werden.** dargestellte linguistische Variable folgendermaßen definieren angelegt (die „\" deuten an, dass die Anweisung in der folgenden Zeile fortgesetzt wird):

```
v = LinguistischeVariable("Verkehrsstärke", 0, 2000,\
      LinguistischerTerm("sehrNiedrig", 0, 250, "sehr Niedrig"),\
      LinguistischerTerm("niedrig", 350, 500),\
      LinguistischerTerm("mittel", 700, 1000),\
      LinguistischerTerm("hoch",1200, 1500),\
      LinguistischerTerm("sehrHoch", 1700, 2000, "sehr Hoch")\
)
```

Die fließenden Übergänge in den Fuzzy-Sets von zwei benachbarten linguistischen Termen werden von der linguistischen Variable automatisch durch eine absteigende Rampe am Anfang, Trapeze und Dreiecke in der Mitte und eine aufsteigende Rampe am Ende ermittelt.

2.9.7.2 Attribute von linguistischenVariablen und Termen

Nach der Konstruktion einer linguistischen Variablen v kann mit

- `v.name` auf die Bezeichnung der linguistischen Variablen,

- `v.untereGrenze` auf die untere Grenze des Wertebereichs,
- `v.obereGrenze` auf die obere Grenze des Wertebereichs,
- `v.terme` auf den Container der zugehörigen linguistischen Terme,

zugegriffen werden.
Auf den linguistischen Term t kann mit

- `t.bezeichner auf den Bezeichner des linguistischen Terms t,`
- `t.untereGrenze` auf die untere Grenze des Bereichs, in dem der Term eine Zugehörigkeit von 1 hat,
- `t.obereGrenze` auf die obere Grenze des Bereichs, in dem der Term eine Zugehörigkeit von 1 hat,
- `t.name` auf den Namen des linguistischen Terms t,

zugegriffen werden.

2.9.7.3 Operationen auf linguistischen Variablen

Alle Operationen mit liguistischen Variablen basieren auf der Funktionalität der SWE „Funktionen Fuzzy“.

2.9.7.3.1 Fuzzyfizierung

```
w = v.fuzzyfiziere(qKfz)
```

Die Methode `fuzzyfiziere` einer linguistischen Variable fuzzyfiziert den übergebenen Eingangswert und liefert einen entsprechend gesetzten Fuzzywert zurück.
Ein Fuzzywert enthält neben den Bezeichnungen der linguistischen Terme jeweils einen Zugehörigkeitswert zu dem entsprechenden Term.

2.9.7.3.2 Erzeugung und Initialisierung neuer Fuzzywerte einer linguistischen Variablen

```
w2 = v2.fuzzywert()
```

Mit der Methode `fuzzywert` können neue Fuzzywerte einer linguistischen Variablen erzeugt werden. Die Zugehörigkeiten aller linguistischen Terme haben den Wert 0. Diese Methode ist zur Initialisierung eines Fuzzywerts notwendig, wenn dieser nicht durch Fuzzyfizierung eines Eingangswertes bestimmt wird, sondern aus anderen Fuzzywerten mit Fuzzy-Logik-Operatoren und/oder durch die üblichen Operationen auf Zahlen ermittelt wird.

2.9.7.3.3 Defuzzyfizierung

```
v.defuzzyfiziere(w)
```

Zur Defuzzifizierung, also zur Rückwandlung eines Fuzzywerts in eine einfache Zahl, die Ermittlung erfolgt auf Basis der Funktionalität der SWE Bibliothek Fuzzy.

2.9.7.3.3.1.1.1 Operationen auf Fuzzywerten bzw. Zugehörigkeiten einzelner Terme

2.9.7.3.4 Zugriff auf Fuzzywerte

Die Zugehörigkeit eines linguistischen Terms (beispielsweise "hoch") in einem Fuzzywert `w` kann mit

```
z= w.zugehörigkeit("hoch")
```

bestimmt werden.

Mit

```
w.zugehörigkeit("hoch", 0.8)
```

wird die Zugehörigkeit des Terms "hoch" auf den Wert 0,8 gesetzt.

Eine alternative Schreibweise ist der Zugriff auf Attribute des Fuzzywerts, wobei als Attributname der Bezeichner des linguistischen Terms benutzt werden kann:

```
z= w.hoch
w.hoch= 0.5
z = w.sehrHoch
w.sehrHoch = 0.3
```

Um einen generischen Zugriff auf Fuzzywerte zu ermöglichen kann der Fuzzywert auch als Container aufgefasst werden, dessen Elemente ein Attribut `name` (mit dem Namen des linguistischen Terms) und ein Attribut `zugehörigkeit` (mit dem Wert der Zugehörigkeit) haben. Dabei ist die Reihenfolge der Argumente im Container durch die Reihenfolge der entsprechenden linguistischen Terme bei Erzeugung der linguistischen Variablen festgelegt

Beispielsweise gibt folgende Anweisung

```
ausgabe w[4].name & ": " & w[4].zugehörigkeit
```

Den Text "`hoch: 0.5`" aus.

2.9.7.3.5 Fuzzy-Operationen

Neben den üblichen auf Zahlen definierten Operationen sind für die Verarbeitung von Fuzzywerten folgende Operationen erforderlich:

2.9.7.3.5.1 Fuzzy-Oder-Verknüpfungen

Für die unscharfe Oder-Verknüpfung kann die Maximums-Funktion `max(...)` oder die algebraische Summe `algSumme(...)` eingesetzt werden.

Beispiel:

```
w2.überdurchschnittlich= max(w.hoch, w.sehrHoch)
w2.überdurchschnittlich= algSumme(w.hoch, w.sehrHoch)
```

2.9.7.3.5.2 Fuzzy-Und-Verknüpfungen

Für die unscharfe Und-Verknüpfung kann die Minimums-Funktion `min(...)` oder das algebraische Produkt `algProdukt(...)` eingesetzt werden

Beispiel:

```
w3.hoch= min(w1.hoch, w2.hoch)
w3.hoch= algProdukt(w1.hoch, w2.hoch)
```

2.9.7.3.5.3 Gamma-Operator

```
gamma(g, zugehörigkeit1, zugehörigkeit2)
```

Mit dem Gamma-Operator kann ein Mittelweg zwischen algebraischer Summe und algebraischem Produkt beschritten werden. Der Operator ist über das Argument `g` das im Intervall [0,1] liegen muss parametrierbar. Wenn g den Wert 0 hat ergibt sich das algebraische Produkt und wenn g den Wert 1 hat ergibt sich die algebraische Summe.

2.9.7.3.5.4 Komplementbildung

```
komplement(zugehörigkeit)
```

Die unscharfe Negation oder Komplementbildung liefert als Ergebnis (`1-zugehörigkeit`) zurück.

2.9.8 Zugriff auf den Datenverteiler

Die Umfassende Datenanalyse bietet verschiedne Möglichkeiten des Zugriffs zum Datenverteiler. Es besteht die Möglichkeiten Informationen aus der Konfiguration abzurufen und in beschränktem Maß zu manipulieren. Andererseits können Daten versendet und empfangen werden.

2.9.8.1 Abfrage von Konfigurationsdaten

Der Zugriff auf die Daten der Konfiguration erfolgt über das spezielle Objekt „konfiguration“ als Ausgangspunkt. Die in der Konfiguration enthaltenen Elemente werden über Attribute oder Funktionen, die dieses Objekt bereitstellt abgerufen. Detailiertere Informationen zu den verscheidenen Objekttypen können dann über deren Attribute und Funktionen ermittelt werden.

2.9.8.1.1 Das Objekt „Konfiguration“

Attribute

Name	Beschreibung
objekte	das Attribut repräsentiert ein Feld mit allen in der Konfiguration für den aktuellen Zeitpunkt gültigen Objekten
typen	das Attribut repräsentiert ein Feld mit allen in der Konfiguration für den aktuellen Zeitpunkt gültigen Objekten, die einen Typ beschreiben
attributgruppen	das Attribut repräsentiert ein Feld mit allen in der Konfiguration für den aktuellen Zeitpunkt gültigen Objekten, die eine Attributgruppe beschreiben
aspekte	das Attribut repräsentiert ein Feld mit allen in der Konfiguration für den aktuellen Zeitpunkt gültigen Objekten, die einen Aspekt beschreiben

Die Auswertung des Attributs ergibt ein Feld entsprechend der Felddefinition gemäß Umfassender Datenanalyse. Einzelne Objekte können mit den entsprechenden Selektoren (wie im Kapitel Container beschrieben) ermittelt werden.
Bei der Verwendung dieser Art des Zugriffs auf die Konfiguration ist zu beachten, das entsprechend der Vorgehensweise des Interpreters immer zuerst das Feld mit allen potentiell zulässigen Objekten gefüllt wird, bevor eine Selektion erfolgen kann. Das ist auch dann der Fall, wenn beispielsweise nur ein bestimmter Typ gesucht wird, wie mit:

konfiguration.typen[pid: „typ.stau“]

Für den Zugriff auf einzelne Konfigurationsobjekte sollten daher, die nachstehend aufgeführten Funktionen verwendet werden, die einen effektiveren Zugriff auf die Konfiguration ermöglichen.

Funktionen

Name	Beschreibung
objekt(pid)	liefert das Objekt mit der angegebenen PID
typ(pid)	liefert den Typ mit der angegebenen PID
attributgruppe(pid)	liefert die Attributgruppe mit der angegebenen PID
aspekt(pid)	liefert den Aspekt mit der angegebenen PID

2.9.8.1.1.1 Systemobjekt

Systemobjekte existieren innerhalb der Umfassenden Datenanalyse in zwei Ausführungen, als Konfgurationsobjekt und als dynamisches Objekt.
Ein Systemobjekt kann man innerhalb eines UDA-Sktiptes auf verschiedene Art und Weise erhalten:

- ermitteltn eines Objekts aus der speziellen Objekt „Konfiguration“
- auslesen eines Systemobjekts aus einem Ergebnisdatensatz, der aus empfangenen Datenverteiler-Datensätzen entstanden ist
- Objekte, die aus einer Anmeldung für die Benachrichtigung bei Änderung von dynamischen Mengen an das Skript geliefert wurden

Auf Informationen eines Systemobjekts kann über die unten beschreibenen Attribute zugegriffen werden.

Attribute

Name	Beschreibung
pid	die PID mit dem das Objekt innerhalb der Konfiguration besteht. Das Attribut wird durch eine Zeichenkette repräsentiert
id	die ID die für das Objekt innerhalb der Konfiguration vergeben wurde. Das Attribut wird durch eine Ganzzahl repräsentiert
name	der Name des Objekts in der Konfiguration. Das Attribut wird durch eine Zeichenkette repräsentiert.
info	die innerhalb der Konfiguration abgelegten Informationen zum Objekt. Das Attribut wird durch ein spezielles strukturiertes Objekt „SystemObjektInformationen“ repräsentiert.
typ	der Typ des Objekts. Das Attribut ist selbst wieder ein Systemobjekt, genaugenommen ein TypObjekt
gültig	die Gültigkeit des Objekts innerhalb der Konfiguration. Das Attribut ist ein Boolscher Wert.
dynamisch	definiert, ob ein Objekt dynamisch ist. Das Attribut ist ein Boolscher Wert.
attributgruppen	ein Feld mit allen Attributgruppen, die für das Objekt verwendet werden können. Der Wert des Attributes wird durch ein Feld mit Systemobjekten in der Ausprägung AttributgruppenObjekt repäsentiert

2.9.8.1.1.1.1 Konfigurationsobjekt

Konfigurationsobjekte sind spezielle Ausprägungen von Systemobjekten. Sie repräsentieren alle innerhalb der Konfiguration erstellten statischen Objekte mit ihren Informationen.
Ein Konfigurationsobjekt hat alle Attribute, die von einem Systemobjekt zur Verfügung gestellt werden. Zusätzlich sind folgende Attribute verfügbar:

Attribute

Name	Beschreibung
mengen	die Mengen, die für das Objekt definiert sind. Das Attribut wird durch ein Feld repäsentiert, das die die Mengen repräsentierenden Systemobjekte enthält. Die Mengen selbst werden durch ein Objekt von Typ MengenObjekt dargestellt.

Konfigurationsobjekte werden innerhalb der Umfassenden Datenanalyse durch folgende speziellere Konfigurationsobjekte untersetzt: TypObjekt, AttributgruppenObjekt, AspektObjekt, AttributTypObjekt, AttributObjekt, MengenObjekt.

2.9.8.1.1.1.1.1 TypObjekt

Ein TypObjekt stellt innerhalb der Umfassenden Datenanalyse ein Systemobjekt dar, das innerhalb der Datenverteilerkonfiguration vom Basistyp „typ.typ“ abgeleitet ist.

Innerhalb eines Typobjekts stehen die gleichen Attribute zur Verfügung wie für ein KonfigurationsObjekt. Zusätzlich sind folgende Attribute verfügbar:

Attribute

Name	Beschreibung
objekte	die Objekte, die als Instanzen des Typs innerhalb der Datenverteilerkonfiguration zum aktuellen Zeitpunkt existieren. Das Attribut ist durch ein Feld repräsentiert, das die Objekte selbst als Objekte vom Typ SystemObjekt enthält.

2.9.8.1.1.1.1.2 AttributgruppenObjekt

Ein AttributgruppenObjekt stellt innerhalb der Umfassenden Datenanalyse ein Systemobjekt dar, das innerhalb der Datenverteilerkonfiguration die Definition einer Attributgruppe repräsentiert.

Innerhalb eines AttributgruppenObjekts stehen die gleichen Attribute zur Verfügung wie für ein KonfigurationsObjekt. Zusätzlich sind folgende Attribute verfügbar:

Attribute

Name	Beschreibung
aspekte	die Aspekte, mit denen die Attributgruppe verwendet werden kann. Das Attribut ist durch ein Feld repräsentiert, das die Objekte selbst als Objekte vom Typ AspektObjekt enthält.
attribute	die Definition der Attribute, die ein Datum der Attributgruppe beschreiben. Das Attribut ist durch ein Feld repräsentiert, das die Objekte selbst als Objekte vom Typ AttributObjekt enthält.

2.9.8.1.1.1.1.3 AspektObjekt

Ein AspektObjekt stellt innerhalb der Umfassenden Datenanalyse ein Systemobjekt dar, das innerhalb der Datenverteilerkonfiguration die Definition eines Aspekts repräsentiert.

Innerhalb eines AspektObjekts stehen die gleichen Attribute zur Verfügung wie für ein KonfigurationsObjekt. Zusätzliche Attribute sind nicht verfügbar. Der Typ dient jedoch der Erhöhung der Typsicherheit bei der Anwendung eines solchen Objekts für die Realisierung von Datenversand und –empfang.

2.9.8.1.1.1.1.4 AttributObjekt

Ein AttributObjekt stellt innerhalb der Umfassenden Datenanalyse ein Systemobjekt dar, das ein Attribut einer Attributgruppe bildet.

Innerhalb eines Attributobjekts stehen die gleichen Attribute zur Verfügung wie für ein KonfigurationsObjekt. Zusätzlich sind folgende Attribute verfügbar:

Attribute

Name	Beschreibung
typ	der Typ des Attributs. Dieses Attributs überlagert die Bedeutung des entsprechenden Attributs innerhalb eines Systemobjekts insofern, dass das Ergebnis ein Objekt vom Typ AttributTypeObjekt ist.
einheit	die innerhalb der Datenverteilerkonfiguration definierte Einheit für das Attribut. Das Attribut wird als Zeichenkette repräsentiert.
immerDefiniert	definiert, ob ein Standardwert für das Attribut existiert. Das Attribut ist als Boolscher Wert definiert.
istEinFeld	liefert einen Boolschen Wert, der definiert, ob das entsprechende Attribut ein Feld ist
wertebereich	liefert den Wertebereich des Attributs. Wertebereiche können innerhalb des Datenverteilers nur fir IntegerAttribute definiert werden. Ist ein solcher definiert, wird ein Objekt vom Typ IntegerValueRangeObjekt geliefert. In allen anderen Fällen hat das Attribut den Wert „undefiniert“.
attribute	liefert die Unterattribute eines Attributs, welches eine Struktur beschreibt. Wenn das Attribut keine Unterattribute besitzt, wird der Wert „undefiniert“ geliefert.

2.9.8.1.1.1.1.5 AttributTypObjekt

Ein AttributTypObjekt stellt innerhalb der Umfassenden Datenanalyse ein Systemobjekt dar, das den Typ eines Attributs beschreibt.

Innerhalb eines Attributobjekts stehen die gleichen Attribute zur Verfügung wie für ein KonfigurationsObjekt. Zusätzlich sind folgende Attribute verfügbar:

Attribute

Name	Beschreibung
einheit	die innerhalb der Datenverteilerkonfiguration definierte Einheit für das Attribut. Das Attribut wird als Zeichenkette repräsentiert.
immerDefiniert	definiert, ob ein Standardwert für das Attribut existiert. Das Attribut ist als Boolscher Wert definiert.
wertebereich	liefert den Wertebereich des Attributs. Wertebereiche können innerhalb des Datenverteilers nur fir IntegerAttribute definiert werden. Ist ein solcher definiert, wird ein Objekt vom Typ IntegerValueRangeObjekt geliefert. In allen anderen Fällen hat das Attribut den Wert „undefiniert“.
statusliste	liefert die für ein Attribut definierten Statuswerte. Statuswerte können nur für eine IntegerAttribut definiert werden. Das Attribut liefert ein Feld mit den definierten Statuswerten als Objekte des Type I ntegerValueStateObjekt, ein leeres Feld, wenn es sich um ein IntegerAttribut ohne definierte Statuswerte handelt oder „undefiniert“ in allen anderen Fällen.

Einen Sonderfall bildet der Typ AttributListDefinitionObjekt, der Attributdefinitionen beschreibt, die selbst wieder Attribute enthalten. Objekte dieses Typs stellen zusätzlich folgende Attribute zur Verfügung:

Attribute

Name	Beschreibung
attribute	die Attribute, die den AttributlistenTyp beschreiben als Feld.

2.9.8.1.1.1.6 MengenObjekt

Ein MengenObjekt stellt innerhalb der Umfassenden Datenanalyse ein Systemobjekt dar, das eine einem Systemobjekt zugeordnete Menge repräsentiert.

Innerhalb eines Mengenobjekts stehen die gleichen Attribute zur Verfügung wie für ein KonfigurationsObjekt. Zusätzlich sind folgende Attribute verfügbar:

Attribute

Name	Beschreibung
objekte	liefert ein Feld mit den in der Menge innerhalb der Konfiguration eingetragenen Systemobjekten.
gültigeObjekte	liefert ein Feld mit den in der Menge innerhalb der Konfiguration eingetragenen und aktuell gültigen Systemobjekten.

2.9.8.1.1.1.7 IntegerValueStateObjekt

Ein IntegerValueStateObjekt repräsentiert ein Systemobjekt der Konfiguration mit ein potentieller Status eines IntegerAttributs definiert ist.

Innerhalb eines IntegerValueStateObjekts stehen die gleichen Attribute zur Verfügung wie für ein KonfigurationsObjekt. Zusätzlich sind folgende Attribute verfügbar:

Attribute

Name	Beschreibung
wert	liefert den ganzzahligen Wert, durch den der Status repräsentiert wird.

2.9.8.1.1.1.8 IntegerValueRangeObjekt

Ein IntegerValueRangeObjekt repräsentiert ein Systemobjekt der Konfiguration mit der Wertebereich eines IntegerAttributs definiert ist.

Innerhalb eines IntegerValueRangeObjekts stehen die gleichen Attribute zur Verfügung wie für ein KonfigurationsObjekt. Zusätzlich sind folgende Attribute verfügbar:

Attribute

Name	Beschreibung
`minimum`	liefert den ganzzahligen minimalen Wert, dern den Bereich nach unten begrenzt.
`maximum`	liefert den ganzzahligen maximalen Wert, dern den Bereich nach oben begrenzt.
`faktor`	liefert den Faktor mit dem die Werte des zugeordneten Attributs skaliert werden als Fliesskommazahl
`einheit`	liefert die für den Wertebereich definierte Masseinheit als Zeichenkette

2.9.8.1.1.2 Dynamisches Objekt

Ein DynamischesObjekt stellt innerhalb der Umfassenden Datenanalyse ein Systemobjekt dar, das während der Laufzeit der Datenverteilerkonfiguration angelegt und/oder entfernt werden kann.

Innerhalb eines DynamischenObjekts stehen die gleichen Attribute zur Verfügung wie für ein SystemObjekt. Es werden keine zusätzlichen Attribute definiert. Ein dynamsiches Objekt wird lediglich anders behandelt, da es zur Laufzeit entstehen und verschwinden kann, im Gegensatz zu Konfigurationsobjekten.

2.9.8.1.1.2.1 Anlegen von dynamischen Objekten

Zum Anlegen eines dynamischen Objekts steht die Funktion „neuesObjekt“ am Objekt „konfiguration“ zur Verfügung.

Der Aufruf der Funktion erfolgt mit folgenden Parametern:

konfiguration.neuesObjekt(<typ>, <pid>, <name>, <konfigurationsdaten1>, <konfigurationsdaten2>, …)

wobei die Parameter die unten beschriebene Bedeutung haben.
Die Parameter pid, name und konfigurationsdaten sind optional.

Parameter	Beschreibung
typ	beschreibt, den Typ des neu anzulegenden Objekts. Der Typ kann als Zeichenkette, die die PID des gewünschten Typs enthält, oder als TypObjekt übergeben werden, das zuvor aus der Konfiguration ermittelt wurde. Das beschriebene TypObjekt sollte einen dynamischen Objekttyp definieren.
pid	enthält die PID, mit der das Objekt angelegt werden soll. Wird keine PID angegeben, wird eine leere Zeichenkette angenommen.
name	enthält den Name, den das neu anzulegende Objekt erhalten soll. Wird kein Name angegeben, wird eine leere Zeichenkette angenommen.
konfigurationsdaten	enthält die konfigurierenden Daten, die bei Bedarf beim Anlegen des Objekts mit an die Konfiguration übergeben werden können. Es kann eine beliebige Anzahl von Datensätzen übergeben werden. Ein Konfigurationsdatensatz muss als Objekt DataAndAtgUsageInformationObjekt übergeben werden. Wie ein solches Objekt angelegt wird, wird weiter unten beschrieben.

Das Ergebnis des Funktionsaufrufes ist das neu angelegte Objekt oder ein Fehlerobjekt, wenn das Objekt von der Konfiguration nicht angelegt wurde.

2.9.8.1.1.1.2.2 Anlegen von Konfigurationsdatensätzen für dynamische Objekte

Die konfigurierenden Daten eines dynamischen Objekts müssen in der Regel bereits beim Anlegen dieses Objekts an die Konfiguration übergeben werden. Dazu wird ein DataAndAtgUsageInformationObjekt verwendet, welches das entsprechende Datenverteiler-Applikationsfunktionen-Objekt kapselt. Ein solches Objekt vereint einen Datensatz mit der Attributgruppe und dem Aspekt, dem die Daten zugeordnet sind. Zum Anlegen eines solchen Objekts stellt das Objekt „konfiguration“ eine entsprechende Funktion zur Verfügung.

konfiguration.datenUndAttributgruppenVerwendungsInformation(<atg>, <asp>, <daten>)

Die Parameter der Funktion haben folgende Bedeutung:

Parameter	Beschreibung
atg	beschreibt die Attributgruppe, für die Daten definiert werden sollen. Die Attributgruppe kann als Zeichenkette, die für die PID der Attributgruppe steht, oder als AttributgruppenObjekt, welches zuvor aus der Konfiguration ermittelt wurde, übergeben werden.
asp	beschreibt den Aspekt, für den Daten gelten sollen. Der Aspekt kann als Zeichenkette, die für die PID des Aspekts steht, oder als AspektObjekt, welches zuvor aus der Konfiguration ermittelt wurde, übergeben werden.
daten	ein Datensatz, der die su setzenden Daten enthält. Der Datensatz wird mittles eines Attributgruppenobjekts erstellt (siehe Abschnitt „Senden und Empfangen von Onlinedaten)

2.9.8.1.1.1.2.3 Zuordnen von Objekten zu dynamischen Mengen

Die Zuordnung von Objekten zu dynamischen Mengen erfolgt innerhalb der Umfassenden Datenanlyse mit folgenden Funktionen eines MengenObjekts:

<menge>.einfügen(<objekt>)
<menge>.entfernen(<objekt>)

Die Menge ist ein zuvor aus der Konfiguration ermitteltes MengenObjekt. Das übergebene Objekt ist ein aus der Konfiguration ermitteltes Systemobjekt oder ein neu angelegtes dynamisches Objekt.

Das Ergebnis des Funktionsaufrufs ist das eingefügte bzw. entfernte Objekt. Wenn die Operation nicht ausgeführt werden konnte liefert die Funktion einen KonfigurationsFehler.

2.9.8.1.1.1.2.4 Löschen von dynamischen Objekten

Dynamsiche Objekte können mittels Umfassender datenanalyse mittels der Funktion

<dynamisches Objekt>.löschen()

entfernt werden.
Das angegebene Objekt kann ein aus der Konfiguration ermitteltes oder ein während des Ablaufs eines UDA-Skripts erzeugtes Objekt sein. Das Objekt wird innerhalb der Konfiguration „invalidiert“.

2.9.8.2 Senden und Empfangen von Onlinedaten

Die Umfassende Datenanalyse bietet die Mittel, um Daten an den Datenverteiler zu versenden und Datensätze von Datenverteiler zu lesen bzw. zu empfangen.

2.9.8.2.1 Repräsentation von Onlinedaten in der Umfassenden Datenanalyse

2.9.8.2.1.1 DavDatenObjekt

Das Objekt repräsentiert ein allgemeines Datenverteilerdatum. Es kann den Wert eines einzelnen nicht weiter untergliederbaren Attributs als auch die Attribute einer Attributliste, d.h. einer Datenstruktur enthalten.

Auf die Elemente und die Werte eines strukturierten Datenobjekts wird über deren Namen und den Elementoperator zugegriffen.
Beispiel:

x = datum.qKfz.plausibel

Für den Zugriff auf die Metadaten eines Datenobjekts stehen Funktionen zur Verfügung, um den Zugriff von der Abfrage der Daten selbst zu unterscheiden.

Beispiel:

x = datum.qKfz.info()

Folgende Funktionen stehen zur Verfügung: typ(), name(), einheit(), wertebereich(), statusliste() und info(). Die Funktionen liefern die gleichen Informationen wie die entsprechenden Attribute einer Attributdefinition (siehe dort).

Ein DavDatenObjekt kann innerhalb der Umfassenden Datenanalyse erzeugt werden, um es mit Daten zu füllen und zu versenden. Ein solches Objekt erhält man außerdem, wenn Daten vom Datenverteiler ausgelesen wurden oder per Benachrichtigung vom Datenverteiler empfangen wurden (siehe Abschnitt Anmeldeanweisung)

Zum Anlegen eines Datensatzes wird die Funktion:

<attributgruppe>.datensatz()

verwendet.
Die verwendete Attributgruppe ist ein AttributgruppenObjekt, das zuvor aus der Konfiguration ermittelt werden muss. Der erzeugte Datensatz entspricht der Struktur, die durch die Attributgruppe definiert wird. Die Attribute selbst sind mit dem Wert „undefiniert“ vorbelegt.

2.9.8.2.1.2 DavDatenFeld

Ein Objekt dieser Klasse repräsentiert ein Feld von Attributwerten innerhalb eines Datenverteiler-Datensatzes. Auf das Feld kann mit allen für allgemeine Container innerhalb der Umfassenden Datenanalyse zur Verfügung stehenden Methoden zugegriffen werden. Die Elemente eines DavDatenFeldes sind DavDatenObjekte oder DabDatenFelder.

2.9.8.2.1.3 DavDatenSatz

Ein DavDatenSatz stellt eine Kombination eines SystemObjekts mit einer Attributgruppe, einem Aspekt und den zugeordneten Daten dar. Desweiteren enthält das Objekt den Zeitstempel, dem die Daten zugeordnet werden sollen.
Einen Datensatz erhält der Uda-Anwender nach dem Auslesen von Daten oder nach dem asynchronen Empfang von Daten innerhalb einer Anmeldeanweisung (siehe dort).

Auf den Inhalt des Datensatzes kann über seine Attribute zugegriffen werden:

Attribute

Name	Beschreibung
`objekt`	das Systemobjekt, zu dem die Daten des Datensatzes gehören. Das Attribut liefert ein SystemObjekt, wie man es auch aus der Konfiguration erhalten würde.
`attributgruppe`	die Attributgruppe, für die Daten innerhalb des Datensatzes enthalten sein können. Das Attribut ist durch ain AttributgruppenObjekt repräsentiert, wie man es auch aus der Konfiguration ermitteln könnte.
`aspekt`	der Aspekt unter dem die Daten verwendet werden. Das Attribut ist durch ein AspektObjekt repräsentiert, wie man es auch aus der Konfiguration ermitteln könnte.
`zeit`	der Zeitpunkt, dem die Daten zuzuordnen sind. Das Attribut ist ein Zeitstempel. Der Zeitstempel hat immer einen Wert, auch wenn der datensatz keine daten enthält.

`daten`	die Daten des Datensatzes selbst. Wenn der Datensatz keine Daten enthält wird der Wert „undefiniert“ geliefert, ansonsten enthält das Attribut ein DavDatenObjekt.
`timeout`	definiert, ob der Datensatz auf Grund eines Timeouts versendet wurde. Innerhalb einer Anmeldeanweisung können optional leere Datensätze für eine Empfängeranmeldung empfangen werden, wenn für eine definierte Zeit kein „echter“ Datensatz empfangen wurde. Das Attribut definiert als Boolscher Wert, ob ein solcher Datensatz vorliegt.

2.9.8.2.2 Versenden von Daten

Das Versenden von Datenverteilerdaten erfolgt mit der Methode `datenSchreiben`, die als Methode des Systemobekts realisiert ist, für das Daten versendet werden sollen.

```
<objekt>.datenSchreiben(zeit, attributgruppe, aspekt, datensatz)
```

Das Objekt ist ein SystemObjekt, das aus der Konfiguration ermittelt oder als dynamisches Objekt innerhalb des Skripts angelegt wurde.

Die Parameter der Funktion sind:

Parameter	Beschreibung
zeit	der Zeitpunkt, der den Daten zugeordnet werden soll. Der Parameter ist als Zeitstempel entsprechend der Definitionen der Umfassenden Datenanalyse zu übergeben
attributgruppe	beschreibt die Attributgruppe, für die Daten versendet werden sollen. Die Attributgruppe kann als Zeichenkette, die für die PID der Attributgruppe steht, oder als AttributgruppenObjekt, welches zuvor aus der Konfiguration ermittelt wurde, übergeben werden.
aspekt	beschreibt den Aspekt unter dem die Daten versendet werden sollen. Der Aspekt kann als Zeichenkette, die für die PID des Aspekts steht, oder als AspektObjekt, welches zuvor aus der Konfiguration ermittelt wurde, übergeben werden.
datensatz	ein DavDatenObjekt, das zuvor innerhalb des Skripts angelegt und mit Daten gefüllt wurde. Es muss in jedem Fall ein vollständig ausgefülltes Datenobjekt übergeben werden, das Versenden eines leern Datensatzes ist mit der Umfassenden Datananalyse nicht möglich.

Das folgende Beispiel verdeutlicht den Gebrauch der Methode `datenSchreiben`. Zunächst wird ein neuer Datensatz der Attributgruppe `verkehrsWerte` erzeugt und mit den ermittelten Daten besetzt. Dieser Datensatz wird der Methode `datenSchreiben` gemeinsam mit den anderen Parametern übergeben.

Beispiel:

```
# Konfigurationsobjekt MQ42
MQ42 = konfiguration.objekte[pid:"eindeutigePidDesMQ42"]

# Konstruktion eines neuen Datensatzes der Attributgruppe verkehrsWerte
atg = konfiguration.attributgruppen[name:"verkehrsWerte"]
neuerDatensatz= atg.datensatz()

# Attribute setzen
neuerDatensatz.qKfz=39
neuerDatensatz.qLkw=12
neuerDatensatz.vPkw=114
neuerDatensatz.vLkw=96

mit fehlerausgabe  MQ42.datenSchreiben(aktuell, atg, „aufbereitetT“, neuerDatensatz)
```

2.9.8.2.3 Lesen von Daten

Zum Lesen von Daten vom Datenverteiler wird die einem Systemobjekt zugeordnete Methode `datenLesen` verwendet.

```
<objekt>.datenLesen(Attributgruppe, Aspekt)
```

In diesem Fall wird zu dem entsprechenden Konfigurationsobjekt der aktuelle Datensatz gelesen und als Rückgabewert zur Verfügung gestellt. Optional kann bei der Methode `datenLesen` mit dem dritten Argument ein Zeitbereich übergeben werden, wenn auf historische Daten zugegriffen werden soll:

```
<objekt>.datenLesen(Attributgruppe, Aspekt, Zeitbereich, [NurGeÄnderte,
[Nachgelieferte]])
```

Der Rückgabewert ist dabei ein Container, der alle Datensätze dieses Zeitbereichs enthält. Der Zeitbereich wird als Zeitbereich, wie er für die Umfassende Datenanalyse“ definiert ist angegeben (siehe dort).
Wenn ein Zeitbereich als Argument übergeben wurde, kann über zwei weitere Argumente zusätzlich spezifiziert werden, ob jeder Datensatz oder nur geänderte Datensätze gelesen werden sollen und ob auch als "nachgeliefert" gekennzeichnete Daten gelesen werden sollen.

Das Ergebnis des Funktionsaufrufes ist ein Feld, welches Objekte vom Typ DavDatenSatz enthält.

2.10 Anweisungen

Anweisungen beschreiben die Schritte, die innerhalb eines Skripts der Umfassenden Datenanalyse ausgeführt werden sollen. Einfache Anweisungen entsprechen einer Zeile innerhalb eines Skripts, komplexere Anweisungen werden aus einzelnen Anweisungen und entsprechenden Schlüsselausdrücken zusammengesetzt.

2.10.1 Einfache Anweisungen

2.10.1.1 Ausgabeanweisung

Eine Ausgabeanweisung besteht aus dem Schlüsselwort **ausgabe** gefolgt von einem Ausdruck. Sie bewirkt die Ausgabe des Ausdrucks in das Protokoll zur entsprechenden Skriptabarbeitung. Dieses Protokoll kann zur Analyse der Skriptabarbeitung herangezogen werden.

```
ausgabe datum.qKfz.info
```

2.10.1.2 Benutzeanweisung

Über das spezielle Konstrukt der "Benutze-Anweisung" kann ein Skript beliebig hierarchisch in mehrere Teilskripte unterteilt werden. Damit können beispielweise Funktionsdefinitionen in eine eigene Skriptdatei ausgegliedert werden und an entsprechender Stelle des Skriptes hinzugeladen werden.

```
benutze "Funktionen" # Einbindung des Skrips „Funktionen“ in das Skript
```

Die „Benutzeanweisung“ kann nur verwendet werden, wenn Skripte auf dem Server der Umfassenden Datenanalyse ausgeführt werden, da nur hier die einzelnen Skripte unter einem Namen abgelegt sind und referenziert werden können.

2.10.1.3 Rückgabeanweisung

Eine Rückgabeanweisung besteht aus dem Schlüsselwort **rückgabe** gefolgt von einem Ausdruck, dessen Wert zurückgegeben werden soll:

```
rückgabe x
```

Rückgabeanweisung werden normalerweise in Funktionen benutzt, um das Ergebnis der Funktion an die aufrufende Stelle zurückzugeben und die Funktion zu verlassen.

2.10.1.4 Zuweisungsanweisung

Eine Zuweisungsanweisung besteht aus einem Ausdruck dem etwas zugewiesen werden kann (z.B. einer Variablen), dem Gleichzeichen und einem Ausdruck dessen Ergebnis zugewiesen werden soll.

```
k = 1
s = "Messquerschnitt"
a = attributgruppe(verkehrswerte:)
a = konfiguration.attributgruppen[name:"ag:verkehrswerte"]
```

Wenn der Zielausdruck eine Variable ist, die noch nicht existiert, wird diese angelegt und erhält den Typ des Objekts, das ihr zugewiesen wurde.
In allen anderen Fällen ist eine Zuweisung nur möglich, wenn eine Kompatibilität zwischen Zieltyp und Quelltyp besteht.

2.10.1.5 Fehlerprüfungsanweisungen

Mit den Fehlerprüfungsanweisungen können Werte eines Funktions- oder Methodenaufruf auf Fehler geprüft werden oder die Prüfung kann explizit unterbunden werden. Die Anweisung muss z.B. zum Aufruf einer Funktion benutzt werden, wenn der Rückgabewert der Funktion nicht benötigt wird (und deshalb keiner Variablen zugewiesen wird).
Die Fehlerprüfungsanweisung beginnt mit einem der Schlüsselwörter

- **ohne fehlerprüfung**
  Mit dieser Anweisung wird die Fehlerprüfung explizit unterbunden (evt. Fehler werden ignoriert) und das Skript wird mit der nächsten Anweisung fortgesetzt.
- **mit fehlerausgabe**
  Wenn der zu prüfende Aufruf ein Fehlerobjekt zurückgibt, wird mit dieser Anweisung veranlasst, dass die Fehlerbeschreibung ausgegeben wird. Der Fehler wird ansonsten ignoriert und das Skript wird mit der nächsten Anweisung fortgesetzt.
- **mit fehlerrückgabe**
  Bei dieser Anweisung wird im Fehlerfall das Fehlerobjekt zurückgegeben und die Funktion beendet. Tritt dieser Fall innerhalb des Hauptskriptes auf, wird der entsprechende Skriptlaif beendet

Beispiele:

```
ohne fehlerprüfung mq4711.datenSchreiben(neuerDatensatz,"aufbereitet")
mit fehlerausgabe mq4711.datenSchreiben(neuerDatensatz,"aufbereitet")
mit fehlerrückgabe mq4711.datenSchreiben(neuerDatensatz,"aufbereitet")
```

2.10.2 Strukturierte Anweisungen

Strukurierte Anweisungen bestehen aus einer definierten Struktur, die einen oder mehrere Anweisungsblöcke enthalten können, deren Anweisungen dann wiederum unter bestimmten Bedingungen ausgeführt werden

2.10.2.1 Schleifen

Bei der Schleifenanweisung sind folgende Varianten möglich:

- Steuerung der Schleife über eine Menge bzw. einen Container mit einer Laufvariable, die bei jedem Durchlauf das jeweilige Element enthält. Die Schleife wird beendet, wenn alle Elemente des Containers verarbeitet wurden, wenn nicht vorher ein expliziter Abbruch der Schleife über eine Rückgabeanweisung erfolgt ist.
- Steuerung der Schleife mit einem Laufindex, der von einem Anfangswert bis zu einem Endwert hochgezählt wird. Dabei ist die Schrittweite bei Bedarf einstellbar. Die Schleife wird beendet, wenn der Laufindex den angegebenen Endwert überschreitet, wenn nicht vorher ein expliziter Abbruch der Schleife über eine Rückgabeanweisung erfolgt ist.
- Steuerung der Schleife über eine angegebene Bedingung, die vor jedem Schleifendurchlauf geprüft wird. Die Schleife wird beendet, wenn die angegebene Bedingung nicht mehr erfüllt ist.

Jede Iteration wird durch das Schlüsselwort **iteration** eingeleitet und durch das Schlüsselwörter **ende iteration** beendet. Die weiteren Schlüsselwörter sind im folgenden Beispiel, das eine Verschachtelung von drei Iterationsanweisungen, die jeweils eine der möglichen Varianten darstellen zeigt, fett gedruckt:

```
iteration über querschnitt in mengeDerQuerschnitte   # Variante 1
  iteration über i von 1 bis anzahlSpuren schrittweite 1   # Variante 2
    iteration solange x<y                              # Variante 3
      ...
    ende iteration
  ende iteration
ende iteration
```

In Variante 1 wird über die Menge `mengeDerQuerschnitte` mit der Laufvariablen `querschnitt` iteriert. Dabei wird diese Iteration vom ersten bis zum letzten Element der Menge durchgeführt, wobei das aktuelle Objekt jeweils über die Laufvariable zugreifbar ist. Variante 2 läuft über den Index `i`, der von `1` bis `anzahlSpuren` hochgezählt wird. Die Angabe der Schrittweite ist optional und hier nicht nötig, da der Standardwert der Schrittweite 1 ist. Die letzte Variante wird durch die Bedingung `x<y` gesteuert. Solange diese Bedingung den Wert wahr zurückgibt wird die innere Schleife durchlaufen.

2.10.2.2 Verzweigungen

Eine Verzweigung enthält einen oder mehrere Anweisungsblöcke, die unter bestimmten Bedingungen ausgeführt werden.

```
wenn WvzFunktionsbyte = 0 dann
       ausgabe "ausschalten"
sonst wenn WvzFunktionsbyte = 1 dann
       ausgabe "einschalten"
sonst wenn WvzFunktionsbyte = 2 dann
       ausgabe "blinken"
sonst
       ausgabe "reserviert"
ende wenn
```

Es wird jeweils die angegebene Bedingung geprüft, wenn die Bedingung den Boolschen Wert „wahr“ liefert, wird der entsprechende Anweisungsblock ausgeführt, liefert die Auswertung des Bedingungsausdrucks den Wert „faslch“ wird der alternative Anweisungsblock ausgeführt.
Der alternative Block kann selbst auch eine Bedingung enthalten, so dass mehrere alternative Anweisungsblöcke innerhalb einer WENN-Anweisung verschachtelt werden können.

2.10.2.3 Nebenläufige Anweisungen

Nebenläufige Anweisungen werden innerhalb eines Skript angelegt und automatsich gestartet. Sie laufen neben den Anweisungen des aufrufenden Hauptskripts in einem eigenen Thread.
Eine solche Anweisung kann in einer Variable abgelegt, referenziert und damit gesteuert werden.
Alle auf diese Weise erzeugten Anweisungen müssen explizit beendet werden, d.h. das Ende des Hauptskripts führt nicht dazu, dass die im Weiteren beschriebenen Aktivitäten beendet werden.

Variablen, die innerhalb einer nebenläufigen Anweisung angelegt werden, gelten nur in deren Kontext, der Kontext des aufrufenden Skripts wird nicht an die Anweisung vererbt. Gemeinsamer Zugriff auf Variablen kann nur über das „global"-Objekt erfolgen.

2.10.2.3.1 Zyklische Anweisung

Über die Zyklisch-Anweisung können Anweisungen in periodischen Zeitabständen wiederholt werden. Die Periodendauer wird durch eine relative Zeitangabe angegeben:

```
zyklisch z alle 5 Minuten
       ausgabe "wie die Zeit vergeht"
ende zyklisch
```

z ist eine Variable über die ein Zugriff auf den Block (Thread) möglich ist. Über die `beenden()`-Methode kann damit ein solcher Anweisungsblock zu einem beliebigen Zeitpunkt wieder gestoppt werden.

```
global.i = 1
zyklisch z alle 5 Minuten
       global.letzterZeitpunkt = aktuell
       global.i = global.i + 1
ende zyklisch

iteration solange global.i < 5
       ausgabe global.letzterZeitpunkt
ende iteration

mit fehlerausgabe z.beenden()
```

2.10.2.3.2 Anmelde-Anweisung

Mit der Anmeldeanweisung kann in der Umfassenden Datenanalyse ein nebenläufiger Thread eingerichtet werden, in dem auf Datensätze von Konfigurationsobjekten zugegriffen werden kann. Der Anmeldungsblock lässt sich in sechs optionale Bereiche unterteilen:

- Definitionsbereich
- Anmeldungsbereich (als Quelle, Sender, Senke bzw. Empfänger)
- Fehlerbehandlungsbereich
- Initialbereich
- Aktualisierungsbereich
- Mengenänderungsbereich

```
anmeldung t
    definition
      :::::::
    quelle                              # (Schreib-/Lese-) Anmeldungsbereich
      :::::::
    sender                              # (Schreib-/Lese-) Anmeldungsbereich
      :::::::
    senke
      :::::::
    empfänger
      :::::::
    fehlerBehandlung f                  # Fehlerbehandlungsbereich
      :::::::
    initial                       # Initialbereich
      :::::::
    jeweils bei aktualisierung a # Aktualisierungsbereich
      :::::::
    jeweils bei mengenÄnderung a # Mengenänderungsbereich
      :::::::
ende anmeldung
```

t ist eine Variable über die ein Zugriff auf den gesamten Anmeldungsblock (Thread) möglich ist.

2.10.2.3.2.1 Definitionsbereich

Der im Definitionsbereich enthaltenen Anweisungen werden zuerst ausgeführt, d.h. auch vor den folgenden Anmeldungen und können daher verwendet werden, um Konfigurationsobjekte für die Anmeldungen erfolgen sollen zu ermitteln oder für die Anmeldung notwendige Variable zu initialisieren.

2.10.2.3.2.2 Anmeldungsbereich

Im Anmeldungsbereich kann spezifiziert werden, zu welchen Objekten Datensätze welcher Attributgruppe unter welchem Aspekt geschrieben (Schlüsselwort **quelle** bzw. **sender**) oder gelesen (Schlüsselwort **senke** bzw. **empfänger**) werden sollen. Dazu müssen jeweils als Argumente eine Objektmenge, die Attributgruppe und der gewünschte Aspekt zur Attributgruppe angegeben werden.

```
anmeldung t
    quelle                        # (Schreib, Lese) Anmeldungsbereich
      alleSpuren, "ag:verkehrswerte", "logischPLgeprueft"
      :::::::
    empfänger
      alleSpuren, "ag:verkehrswerte", "formalPLgeprueft"
      :::::::
    fehlerBehandlung f
```

Neben den obligatorischen Argumenten Objektmenge, Attributgruppe und Aspekt kann optional ein Timer durch eine relative Zeitangabe vorgegeben werden. Diese Angabe bewirkt, dass nach Ablauf dieser Zeitspanne der Aktualisierungsblock durch ein Timer-Objekt angestoßen wird. Diese Zeitspanne wird pro Objekt der Menge jeweils bei einem aktuellen Datensatz zu diesem Objekt zurückgesetzt. Mit diesem Mechanismus kann kontrolliert werden, ob Daten, die zyklisch eintreffen sollten, nicht innerhalb der vorgegebenen Zeitspanne erhalten wurden. Die Angabe eines Timers ist nur im Zusammenhang einer Leseanmeldung möglich.

Die Angabe von Objekten, Attributgruppen und Aspekten kann optional über die PID oder die zuvor aus der Konfiguration ermittelten Objekte erfolgen.

```
    empfänger
      alleSpuren, "ag:verkehrswerte", "formalPLgeprueft", timer 5 Minuten
```

Bei den einzelnen Anmeldungen auf Objektmengen müssen im Fall einer Änderung der Menge implizit Anmeldungen (bei Neuzugängen von Objekten in die Menge) bzw. Abmeldungen (bei Abgängen) durchgeführt werden. Einen möglichen Anwendungsfall stellt dabei die Anmeldung auf die Menge von Staus dar. Wenn ein neuer Stau entsteht bzw. ein Stau nicht mehr vorliegt, muss die Menge entsprechend angepasst werden und im Aktualisierungsbereich muss auf die Situation eingegangen werden können.
Die im Anmeldungsbereich angegebenen Objektmengen können auch konfigurierte Mengen von Konfigurationsobjekten sein. Diese Mengen selbst sind auch Objekte der Konfiguration und können damit auch zugeordnete Attributgruppen haben auf die man sich anmelden kann. Zur Unterscheidung

ob man sich auf alle Objekte einer angegebenen Menge oder auf die Menge selbst anmelden will, muss man im letzten Fall die Schlüsselworte "`als menge`" vor der Objektmenge angeben. Insbesondere ist mit diesem Konstrukt eine Anmeldung auf Veränderungen der Menge selbst möglich.

```
staus = konfiguration.objekte[pid:"VRZ Leverkusen"][name:"staus"]
::::
anmeldung t
    empfänger
       als menge staus
    :::::
ende anmeldung
```

2.10.2.3.2.3 Fehlerbehandlungsbereich

Im Fehlerbehandlungsbereich werden mögliche Fehler, die bei den Schreib- und Leseanmeldungen vorkommen können, behandelt. Wenn eine Anmeldung auf Daten abgewiesen wurde wird ein entsprechendes Fehler-Objekt vom Typ DatenZugriffsFehler erzeugt und dieses Objekt dem Fehlerbehandlungsbereich als *f* übergeben. Damit kann der Fehler direkt analysiert werden und es können notwendige Schritte eingeleitet werden.

```
    fehlerBehandlung f                  # Fehlerbehandlungsbereich
      ausgabe f
      wenn f.typ <> "DatenZugriffsFehler" dann
        ausgabe "unerwarteter Fehlertyp!"
      ende wenn
      :::::::
    initial                         # Initialbereich
      :::::::
```

2.10.2.3.2.4 Initialbereich

Die Anweisungen im Initialbereich der Anmeldungsanweisung werden einmalig nach den Anmeldungen ausgeführt.

```
    initial                         # Initialbereich
      ausgabe "warten auf daten"
      :::::::
    jeweils bei aktualisierung a # Aktualisierungsbereich
      :::::::
```

2.10.2.3.2.5 Aktualisierungsbereich

Die Anweisungen im Aktualisierungsbereich werden mit jedem Empfang eines Datensatzes und bei Ablauf eines Timers abgearbeitet. Dabei wird der Bezug zu den empfangenen Daten, zum zugeordneten Objekt etc. über den Parameter im Aktualisierungsbereichs-Kopf hergestellt.

```
    jeweils bei aktualisierung a        # Aktualisierungsbereich
      obj   = a.objekt
      daten = a.daten
      wenn a.timeout dann
         ausgabe "Timeout für " & obj.name  # betroffenes Objekt
         :::::::
      sonst
         qPkw = daten.qKfz - daten.qLkw
         :::::::
      ende wenn
      :::::::
ende anmeldung
```

2.10.2.3.2.6 Mengenänderungsbereich

Bei Änderung einer im Anmeldungsbereich angegebenen Objektmenge wird der Mengenänderungsbereich abgearbeitet. Dabei wird der Bezug zur veränderten Menge, zum betroffenen Objekt und der Art der Ändererung über den Parameter im Kopf des Mengenänderungsbereichs hergestellt.

```
    jeweils bei mengenÄnderung a        # bereich
      menge = a.menge
      obj = a.objekt
      wenn a.erweitert dann
         ausgabe "Neuer Stau" & obj.name
      sonst wenn a.entfernt
         ausgabe "Stau aufgelöst" & obj.name
      ende wenn
      :::::::
ende anmeldung
```

2.10.2.3.2.7 Zugriff auf den Anmeldungsblock

Nach der Abarbeitung der Anmeldungen, der Fehlerbehandlung und des Initialblocks wird die Skriptausführung hinter dem Anmeldungsblock fortgesetzt. Für den Aktualisierungsblock wird ein eigener Thread gestartet, der parallel zur normalen Skriptausführung ausgeführt wird.
Der Zugriff auf den gesamten Anmeldungsblock (Thread) ist über den Parameter im Anmeldungs-Kopf möglich.

```
anmeldung t
    :::::::
ende anmeldung
```

Der angegebenen Variablen `t` wird ein Objekt vom Typ Anmeldungsthread zugewiesen. Folgende Operationen sind auf einem Anmeldungsthread möglich.

2.10.2.3.2.7.1.1 Abfragen der aktuellen Leseanmeldungen

```
anmeldungen= t.senkeAnmeldungen()
anmeldungen= t.empfängerAnmeldungen()
```

Diese Methoden stellen alle aktuellen Leseanmeldungen für Senken bzw. Empfänger in einem Container zur Verfügung. Die einzelnen Elemente des Containers besitzen die Attribute objekt, attributgruppe, aspekt und timeout um auf die entsprechenden Werte zuzugreifen.

2.10.2.3.2.7.1.2 Abfragen der aktuellen Schreibanmeldungen

```
anmeldungen= t.quelleAnmeldungen()
anmeldungen= t.senderAnmeldungen()
```

Diese Methoden stellen alle aktuellen Schreibanmeldungen für Quellen bzw. Sender in einem Container zur Verfügung. Die einzelnen Elemente des Containers besitzen die Attribute objekt, attributgruppe und aspekt um auf die entsprechenden Werte zuzugreifen.

2.10.2.3.2.7.1.3 Nachträgliche Leseanmeldung

```
t.lesen(senke, objekt, attributgruppe, aspekt, timeout)
```

Mit der Methode `lesen()` kann dem Anmeldungsblock eine weitere Leseanmeldung hinzugefügt werden. Über den Parameter senke bestimmt man den Charakter der Anmeldung: Ist senke `wahr`, so handelt es sich bei der Anmeldung um eine Senkenanmeldung, sonst um eine Empfängeranmeldung. Über die anderen Parameter kann spezifiziert werden, für welches Objekt, Attributgruppe und Aspekt die Anmeldung ausgeführt werden soll. Über den optionalen Parameter `timeout` kann ein Zeitintervall für den Empfangsüberwachungstimer vorgegeben werden. Der Timer wird mit jedem Aufruf des Aktualisierungsblocks für diese Anmeldung zurückgesetzt. Wenn der Timer abläuft, wird der Aktualisierungsblock aufgerufen.

2.10.2.3.2.7.1.4 Nachträgliche Schreibanmeldung

```
t.schreiben(quelle, objekt, attributgruppe, aspekt)
```

Mit der Methode `schreiben()` kann dem Anmeldungsblock eine weitere Schreibanmeldung hinzugefügt werden. Über den Parameter quelle bestimmt man den Charakter der Anmeldung: Ist quelle `wahr`, so handelt es sich bei der Anmeldung um eine Quellenanmeldung, sonst um eine Senderanmeldung. Über die anderen Parameter kann spezifiziert werden, für welches Objekt, Attributgruppe und Aspekt die Anmeldung ausgeführt werden soll.

2.10.2.3.2.7.1.5 Nachträgliche Abmeldung einer Leseanmeldung

```
t.nichtLesen(objekt, attributgruppe, aspekt)
```

Mit der Methode `nichtLesen()` kann eine bisher bestehende Anmeldung zum Lesen wieder aufgehoben werden. Über Parameter kann spezifiziert werden, für welches Objekt, Attributgruppe und Aspekt die Abmeldung ausgeführt werden soll.

2.10.2.3.2.7.1.6 Nachträgliche Abmeldung einer Schreibanmeldung

```
t.nichtSchreiben(objekt, attributgruppe, aspekt)
```

Mit der Methode `nichtSchreiben()` kann eine bisher bestehende Anmeldung zum Schreiben wieder aufgehoben werden. Über Parameter kann spezifiziert werden, für welches Objekt, Attributgruppe und Aspekt die Abmeldung ausgeführt werden soll.

2.10.2.3.2.7.1.7 Abmelden und Beenden

```
t.beenden()
```

Mit der Methode `beenden()` eines Aktualisierungsthreads werden alle Anmeldungen des zugeordneten Anmeldungsblocks abgemeldet und die parallele Ausführung des Aktualisierungsblocks beendet.

2.11 Erweiterungen der Skriptsprache

Die Umfassende Datenanalyse bietet die Möglichkeit, nutzerdefinierte Funktionen und Quantoren zu erstellen und zu verwenden.
Dabei können zum einen mit Hilfe der Skriptsprache neue Funktionen und Quantoren in einer eigenen Datei definiert werden und in beliebigen Skripten über die "benutze"-Anweisung eingebunden werden. Zum anderen muss es möglich sein, Funktionsbibliotheken die beispielsweise in Java geschrieben wurden, einzubinden und die zur Verfügung gestellte Funktionalität zu nutzen.

2.11.1 Benutzerdefinierte Funktionen

Eine Funktionsdefinition wird durch das Schlüsselwort **funktion** eingeleitet und durch die Schlüsselwörter **ende funktion** beendet. Wenn eine Funktion aufgerufen wird, werden die formalen Parameter auf die entsprechenden Ausdrücke des Funktionsaufrufs gesetzt und die Anweisungen der Funktion ausgeführt. Mit der Rückgabeanweisung wird die Funktion verlassen und das Ergebnis der Funktion an die aufrufende Stelle übergeben.

```
funktion f(a,b)
  rückgabe a+b
ende funktion
```

Funktionen, die keine Rückgabeanweisung enthalten oder die abgearbeitet werden können ohne eine Rückgabeanweisung auszuführen, geben `undefiniert` oder im Fehlerfall unter Verwendung der "mit fehlerrückgabe"-Anweisung ein Fehlerobjekt zurück.

2.11.2 Benutzerdefinierte Quantoren

Zur Definition von Quantoren steht in der Umfassenden Datenanalyse eine spezielle Syntax zur Verfügung:

```
quantor überDieHälfte der elemente in M gilt bedingung B
       rückgabe für mindestens 50 % der elemente x in M gilt B(x)
ende quantor
```

2.11.3 Einbinden von JAVA-Bibliotheken

Durch die Möglichkeit zur Laufzeit eines Skriptes externe Funktionen, die in Java implementiert wurden, aufzurufen, können bestimmte Funktionen effizienter implementiert werden. Außerdem kann

dadurch der Zugriff auf Funktionalitäten, die in der Umfassenden Datenanalyse von Haus aus nicht zur Verfügung stehen, ermöglicht werden.
Zum Zugriff auf Methoden einer Javaklasse muss diese im ersten Schritt geladen werden.

```
javaKlasse= Java(klassenname)
```

Die Funktion Java() liest die kompilierte Version der im Parameter `klassenname` spezifizierten Java-Klasse und liefert ein Objekt zurück, mit dem der Zugriff auf bestimmte Methoden der Klasse möglich ist.
Danach können bestimmte Methoden der geladenen Java-Klasse mit dem Konstrukt

```
ergebnis= javaKlasse.funktionsName(...)
```

aufgerufen werden.

Die zu ladendenen Javaklassen müssen sich im Klassenpfad des Uda-Servers (im Serverbetrieb) bzw. des Interpreters befinden.
Alle Parameter werden weitestgehend an die in der Umfassenden Datenanalyse verwendeten Datentypen angepasst.