aqua_plan AZUR
Benutzerhandbuch

aqua_plan GmbH

Aachen, 20. März 2008

Inhaltsverzeichnis

Kapitel 1

Einleitung

Der AZUR-Compiler-Interpreter ist eine Programmiersprache zum Verarbeiten von Zeitreihen, Erstellen von Reports mit und ohne Graphik und zum Bearbeiten von relationalen Datenbanken. In der Syntax ist es angelehnt an traditionelle Programmiersprachen wie C oder Pascal. Azur bietet zwar nicht die Möglichkeit, eigene Typen zu definieren, enthält jedoch eine Vielzahl vordefinierter Typen, die das Arbeiten mit Zeitreihen, Relationen, Achsenkreuzen oder Reportseiten unterstützen.

Eine große Anzahl fest eingebauter Funktionen (built-in-functions) erleichert die Programmierung. Die meisten Programme können so aus nur wenigen Zeilen bestehen. Selbstdefinierte Funktionen können in Libraries abgelegt werden, die bei Bedarf gelinkt werden können.

AZUR wird ständig weiterentwickelt. Benutzerwünsche, Anregungen oder Verbesserungsvorschläge sind immer willkommen.

Kapitel 2

Semantik

2.1 Allgemeines

Groß- und Kleinschreibung wird nicht beachtet. So ist z.B. `REAL` identisch mit `Real`. Ein `#` am Anfang einer Zeile macht die gesamte Zeile zur Kommentarzeile. Andere Kommentare werden nicht unterstützt, insbesondere bezeichnet ein `#` innerhalb einer Zeile den Test auf Ungleichheit. Spaces und Tabs können an beliebiger Stelle zwischen Symbole eingestreut werden. Leerzeilen sind in der Regel an jeder Stelle möglich.

Um die Syntax möglichst einfach und intuitiv zu halten, ist ein Zeilenumbruch gleichbedeutend mit einem Semikolon, welches als Befehlstrenner dient. Die Angabe eines Semikolons am Zeilenende ist demnach überflüssig. Andererseits ersetzt das Zeilenende auch das Semikolon. Das Konstrukt

```
IF (a<b)
   c := 5
ELSE
   c := 6
ENDIF
```

ist äquivalent zu

```
IF (a<b); c := 5; ELSE; c := 6; ENDIF
```

2.2 Typen

Azur ist streng typisiert. Jede Variable hat einen eindeutigen Typ, der sich, nachdem er einmal festgelegt ist, nicht mehr ändert. Jeder Ausdruck hat einen festen Typ, was der Grund für das Fehlen von Deklarationen außerhalb von Funktionsköpfen ist. Jede Variable wird implizit deklariert, wenn ihr das erste Mal etwas zugewiesen wird, da der Typ der rechten Seite fest liegt. Als Seiteneffekt wird dadurch auch gewährleistet, dass alle Variablen initialisiert sind, bevor sie benutzt werden.

Wird eine Prozedur oder eine Funktion aufgerufen, findet eine strenge Typüberprüfung statt. Die Anzahl der Parameter und deren Typ muss genau übereinstimmen. Eine Ausnahme davon bildet die Hauptprozedur `AZUR`, deren Parameter die Schnittstelle nach außen bilden.

Die Typüberprüfung, sowie das Anlegen aller Variablen findet vor dem Programmablauf beim Aufbau des Syntaxbaumes statt. Laufzeitfehler werden so auf ein Minimum reduziert.

Folgende Typen sind in Azur definiert:

Typ	Abk.	Beschreibung
Real	R	alle reellen Zahlen (umfasst damit REAL und INTEGER)
String	S	beliebige Texte
Bool	B	nimmt die logischen Werte `TRUE` oder `FALSE` an
Zeitpunkt	ZP	ein beliebiger Zeitpunkt
Distanz	ZD	eine Zeitdistanz, z.B. 4 Tage oder 1 Monat
Intervall	ZI	ein Zeitintervall oder ein Realintervall
ZR	ZR	eine Zeitreihe
ZRList	ZL	eine Liste von Zeitreihen
Tupel	TU	ein Tupel (Element einer Relation oder auch Menge von Komponenten)
Relation	RL	Relation im Sinne einer relationalen Datenbank
Datenbank	DB	Relationale Datenbank
TupelList	TL	Eine Liste mit Tupeln
AxBox	AX	zur Anbindung von Graphik
GeoPoint	GP	Koordinate X,Y oder X,Y,Z
Page	PG	Eine Reportseite
Quant	Q	ein Stück Zeitreihe
QuantList	QL	eine Liste mit Quanten
Polygon	PO	eine Liste von Punkten und einige Attribute
Layer	L	eine Liste von Polygonen und einige Attribute
Karte	K	eine Liste von Layern und einige Attribute
Array	A	ein Feld, das Strings enthält und mit Strings indiziert wird (assoziatives Array)
Report	RP	eine Liste von Pages

2.3 Arrays

Der Typ **Array** (Feld) stellt gegenüber klassischen Programmiersprachen eine Besonderheit dar. In Azur ist, wie z.B. in *perl*, jedes Feld ein **assoziatives** Array. Es hat den Basistyp String und wird auch mit Strings indiziert. Weil der Indextyp also nicht angeordnet ist, ist es auch das Array nicht, es gibt keine festgelegte Reihenfolge der Elemente.

Der Zugriff erfolgt mittels des `[]`-Operators, auch auf der linken Seite einer Zuweisung. Beispiel:

```
Std["Montag"] := "8"
Std["Dienstag"] := Std["Montag"]
```

Für Index und Wert können neben String alle weiteren Typen benutzt werden. Diese werden automatisch in Strings umgewandelt. Der Ausdruck

```
A["10"] := "34"
```

ist identisch mit

```
A[10] := 34
```

2.3.1 Addieren von Arrays

Arrays können addiert werden. Beispiel (A und B seien Arrays):

```
C := A + B
```

Die Arrays A und B bleiben unverändert. Das Ergebnis C enthält alle Elemente aus A und B.

Gibt es einen Index, der in A **und** in B vorkommt, so ist die eindeutige Zuordnung von Index zu Wert nicht gewährleistet. In diesem Falle werden alle Indexe verworfen und den Werten aus A und B neue Indexe zugeordnet.

Beispiel:

```
A := Array()
A[1] := 10
A["Haus"] := "Dach"
A["Bett"] := "Pfanne"

B := StrToArray("Hund Katze Maus")
# B[0]="Hund", B[1]="Katze", B[2]="Maus"

C := A + B
# B[1]="Katze" und A[1]=10 haben den gleichen Index
# deshalb wird neu nummeriert. Es gilt dann
# C[1]=10, C[2]="Pfanne", C[3]="Dach", C[4]="Hund", C[5]="Katze", C[6]="Maus"
```

2.3.2 FORALL und Arrays

Ein Array kann mit FORALL durchlaufen werden. Der Schleifenvariable werden normalerweise die **Werte** zugeordnet. Um die **Indexe** zu durchlaufen, nennt man die Schleifenvariable *_key...* (die ersten vier Buchstaben sind entscheidend). Beispiel:

```
FORALL s IN A
   print (s)
ENDFOR
```

ergibt

```
Dach
Pfanne
```

aber

```
FORALL _key1 IN A
   print (_key1)
ENDFOR
```

ergibt

```
Haus
Bett
```

Wichtig: beide Schleifen durchlaufen das Array sortiert nach **Wert**! Großbuchstaben werden vor Kleinbuchstaben gereiht.

Wenn die Groß-Klein-Schreibung außer Acht gelassen werden soll, fügt man hinter dem Array das Schlüsselwort `(ÏCASE)` an. Beispiel:

```
FORALL s IN A ("ICASE")
   print (s)
ENDFOR
```

ergibt

```
Dach
doch # angenommen, doch ist auch enthalten
Pfanne
```

Möchte man die Elemente in der Sortierreihenfolge des Index“ durchlaufen, so gibt man hinter dem Array das Schlüsselwort (K̈EŸ) an. Beispiel:

```
FORALL s IN A ("KEY")
   print (s)
ENDFOR
```

ergibt

```
Pfanne
Dach
```

oder

```
FORALL _key IN A ("KEY")
   print (s)
ENDFOR
```

ergibt

```
Bett
Haus
```

2.4 Statische Variable

Alle Variable, die in einer Funktion deklariert werden, können auch nur in dieser Funktion benutzt werden. Sie verlieren ihre Gültigkeit und ihren Inhalt, wenn die Funktion verlassen wird. Auf Variable, die in anderen Funktionen deklariert sind, kann nicht zugegriffen werden.

Es ist jedoch manchmal sinnvoll, den Wert einer Variablen zu erhalten, um dann später in der gleichen oder einer anderen Funktion wieder darauf zuzugreifen. So ist es beispielsweise sinvoll, nicht bei jedem Zugriff auf eine Relation den Index neu zu berechnen. Es genügt, zu Beginn den Index zu berechnen, der dann später immer wieder benutzt werden kann.

Dies wird in AZUR mit **statischen Variablen** erreicht. Diese Variable sind für alle Funktionen eines Moduls sichtbar und verlieren ihren Wert nicht, können jedoch einen neuen (auch ungültigen) Wert zugewiesen bekommen.

Statischen Variablen wird im Namen ein $ vorangestellt. So bedeutet

```
$Stamm := Relation ("stamm")
```

dass die Variable `$Stamm` nach Beendigung der Funktion nicht gelöscht wird, und auf sie von anderen Funktionen zugegriffen werden kann.

Ein Zugriff auf statische Variable ist von allen Funktionen einer .azr-Datei (Modul) aus möglich. Module, die hinzugelinkt werden, können nicht auf globale Variable anderer Module zugreifen.

2.5 Operatoren

2.5.1 Unäre Operatoren

Unäre Operatoren sind solche, die nur ein Argument verarbeiten. Typischerweise unterscheidet sich der Ergebnistyp vom Typ des Arguments

Operator	Argumenttyp	Ergebnistyp	Beispiel
`@`	String	Zeitpunkt	`@"15.9.1992"` oder `@Anfang`
`~`	String	Zeitdistanz	`~"2 Wochen"`
`-`	Real	Real	`-10`
`++`	Real	Real	`i++;`
`--`	Real	Real	`i--;`

2.5.2 Binäre Operatoren

Punktrechnung vor Strichrechnung wird beachtet, d.h. multiplikative Operatoren binden stärker als additive, ansonsten wird von links nach rechts ausgewertet.

Operator	Links	Rechts	Ergebnistyp	Beispiel
+	Real	Real	Real	a+10
-	Real	Real	Real	a-10
*	Real	Real	Real	a*10
/	Real	Real	Real	a/10
^	Real	Real	Real	a^10 , Potenzoperator
+=	Real	Real	Real	a+=5;
-=	Real	Real	Real	a-=9;
+=	String	String	String	s+="Hallo"
-=	String	String	String	"s-="Dr."
+	String	String	String	"Hallo " + Name
-	String	String	String	"Hallo" - "all" = "Ho"
*	Real	String	String	50*"-"
+	Zeitpunkt	Distanz	Zeitpunkt	von + zweitage
-	Zeitpunkt	Distanz	Zeitpunkt	von - zweitage
-	Zeitpunkt	Zeitpunkt	Zeitdistanz	bis - von
+	Distanz	Distanz	Distanz	eintag + dreiminuten
-	Distanz	Distanz	Distanz	eintag - dreiminuten
*	Distanz	Real	Distanz	eintag * -1
[]	Zeitpunkt	Zeitpunkt	Intervall	[von,bis]
[]	Real	Real	Intervall	[23,67]
[]	Array	String	String	A["0"]
{,}	Real	Real	GeoPoint	{10,34.5}
{,,}	Real	Real Real	GeoPoint	{10,34.5,8}
+	Polygon	GeoPoint	Polygon	poly + p
+	Layer	Polygon	Layer	L + poly
-	Layer	Polygon	Layer	L - poly
+	Karte	Layer	Karte	Map + L
-	Karte	Layer	Karte	Map - L
+	Array	Array	Array	feld1 + feld2
+	ZRList	ZR	ZRLIST	zrl := zrl + zr
-	ZRList	ZR	ZRLIST	zrl := zrl - zr
+	TupelList	Tupel	TupelList	TL + tup
+=	TupelList	Tupel	TupelList	TL += tup

2.5.3 Boolsche Operatoren

Boolsche Operatoren finden Verwendung bei logischen Vergleichen, also beim Aufbau von *Bedingungen* (z.B. ist in IF (a=b) das = ein boolscher Operator).

Das Ergebnis eines boolschen Operators ist True oder False. Dies kann jedoch nicht einer Variablen vom Typ Bool zugewiesen werden. Die Benutzung von boolschen Operatoren ist beschränkt auf den Aufbau von *Bedingungen*.

Folgende boolsche Operatoren sind definiert

Operator	linker Typ	rechter Typ	Beispiel
<	Real	Real	IF (a<b)
>	Real	Real	WHILE (a>10)
<=	Real	Real	IF (a<=99)
>=	Real	Real	IF (ret>=0)
=	beliebig	beliebig	IF (dername=\Popernen\)
~=	String	String	IF (dername =\PoPERnen\)
#	beliebig	beliebig	IF (ret1#ret2)
OR	Bool	Bool	UNTIL (fertig OR (zeile>1000)
AND	Bool	Bool	WHILE (weiter AND a<b
NOT	Bool	unär	IF (NOT ganzamanfang

Der Operator ~= vergleicht zwei Strings, ohne die Groß-Klein-Schreibung zu beachten.

2.6 Kontrollstrukturen

2.6.1 IF

IF (*Bedingung*)
Anweisungen1
ENDIF
oder
IF (*Bedingung*)
Anweisungen1
ELSE
Anweisungen2
ENDIF
oder

IF (*Bedingung*)
   Anweisungen1
ELSEIF (*Bedingung2*)
   Anweisungen2
ELSEIF (*Bedingung3*)
   Anweisungen3
ELSE
   Anweisungen4
ENDIF

Die *Bedingung* wird ausgewertet. Ist das Ergebnis TRUE, so werden die *Anweisungen1* ausgeführt. Ist das Ergebnis FALSE, werden bei der zweiten Form die *Anweisungen2* ausgeführt. In jedem Fall wird das Programm nach dem ENDIF fortgesetzt.

Die dritte Variante findet Verwendung, wenn mehr als zwei Fälle möglich sind. Die Bedingungen werden in der Reihenfolge ihres Auftretens abgearbeitet. Ist eine Bedingung wahr, werden die entsprechenden Anweisungen ausgeführt und das Programm nach dem ENDIF fortgesetzt. Ist keine Bedingung wahr, wird, so vorhanden, der ELSE-Fall ausgeführt.

Beispiele:

```
IF (a > max_wert)
   max_wert := a
ENDIF
```

```
IF (a < 0)
   Print ("Die Zahl ist negativ.")
ELSE
   Print ("Die Zahl ist positiv.")
ENDIF
```

```
IF (s="Stefan")
   Print ("Hol schon mal den Wagen")
ELSEIF (s="Harry")
   Print ("Ja")
ELSEIF (s="Haus" AND tuer="Offen")
   Print ("Hinein")
```

```
ELSE
   Print ("Egal")
ENDIF
```

2.6.2 WHILE

WHILE (*Bedingung*)
   Anweisungen
ENDWHILE

Zunächst wird der *Bedingung* ausgewertet. Ist das Ergebnis gleich TRUE, werden die *Anweisungen* durchgeführt und die Bedingung erneut ausgewertet. Dies geschieht solange, bis das Ergebnis der Bedingung gleich FALSE ist.

Beispiel:

```
zaehler := 0
WHILE (zaehler < 10)
   zaehler := zaehler + 1
ENDWHILE
```

2.6.3 FORALL

FORALL *Element* IN *Liste*
   Anweisungen
ENDFOR

oder

FORALL *Element* IN *Liste* WHILE (*Bedingung*)
   Anweisungen
ENDFOR

Dem *Element* werden nacheinander alle Elemente der *Liste* zugewiesen. In AZUR können Listen vom Typ ZRList, QuantList, Polygon, Layer, Karte, Array, Relation und Datenbank durchlaufen werden. Die Elemente von Polygonen sind GeoPoints, die Elemente von Layer Polygone und die Elemente von Karte Layer.

Arrays werden sortiert durchlaufen. Siehe dazu Abschnitt 2.3.

Wird, wie oben in der zweiten Form, ein WHILE benutzt, dann bricht die Schleife ab, wenn die Bedingung nicht erfüllt ist. Die Bedingung wird jeweils nach der Zuweisung an *Element* getestet, sodass *Element* auch in der Bedingung benutzt werden kann.

Falls die Liste eine Relation ist (aber keine dBase-Relation), kann diese auch nach einem bestimmten Schlüssel sortiert durchlaufen werden. Die Syntax hierfür ist

FORALL *Element* IN *Relation* (sortindex)
   Anweisungen
ENDFOR

sortindex ist ein String oder ein Ausdruck vom Typ String. Er enthält den Feldnamen (oder die mit + verketteten Feldnamen) für den Index. Wenn *sortindex* mit einem -Zeichen beginnt, wird die Relation in umgekehrter Reihenfolge (Große zuerst) durchlaufen. Soll der Hauptindex rückwärts durchlaufen werden, wird nur ein -Zeichen übergeben. Beispiel:

FORALL *Element* IN *Relation* ("-")
   Anweisungen
ENDFOR

WHILE und Sortierung können auch kombiniert werden.

Beispiele:

```
zr_liste := OrtQuery ("12345")
FORALL zr IN zr_liste
   Print (Parameter (zr))
ENDFOR
```

```
bereich := [11.11.77, 11.11.88]
quant_liste := QuantenFolge (zr, bereich)
FORALL q IN quant_liste
   Print (YLinks (q))
ENDFOR
```

```
lay := ReadLayer ("orte.geo")
FORALL poly IN lay WHILE (Name(poly)#"Aix")
ENDFOR
IF (Name(poly)="Aix")
 print ("gefunden")
ENDIF
```

2.6.4 REPEAT

REPEAT
   Anweisungen
UNTIL (*Ausdruck*)

Zunächst werden die *Anweisungen* ausgeführt und dann der *Ausdruck* ausgewertet. Dies wird solange wiederholt, bis die Auswertung TRUE ergibt. Das bedeutet, dass die *Anweisungen* auf jeden Fall einmal ausgeführt werden.

Beispiel:

```
zaehler := 0
REPEAT
   zaehler := zaehler + 1
UNTIL (zaehler >= 10)
```

2.7 Selbstdefinierte AZUR-Funktionen

AZUR bietet dem Benutzer die Möglichkeit, eigene Funktionen zu definieren. Ihre Definition erfolgt im AZUR-Programm vor der Funktion, von der sie aufgerufen werden. Rekursionen sind möglich, wobei der rekursive Aufruf am Ende der Funktion stehen muss. Den Funktionen können beliebig viele Parameter übergeben werden, zulässige Datentypen sind dabei alle AZUR-Typen.

Man unterscheidet Funktionen mit Rückgabewert

```
My_Function (Parameterliste) : Typ
   RETURN Rueckgabewert
END
```

und Funktionen ohne Rückgabewert (Prozeduren)

```
   My_Procedure (Parameterliste)
   END
```

Durch das RETURN wird der Rückgabewert an die aufrufende Funktionen zurückgeliefert. Der Typ dieses Rückgabewertes muss mit dem bei der Definition angegebenen Typ übereinstimmen.

2.7.1 Beispiele

```
# AZUR-Programm mit einer selbstdefinierten rekursiven Funktion
# zur Berechnung der Fakultaet einer Zahl
fakultaet (Real r) : Real
   IF (r<2)
      RETURN 1;
   ELSE
      RETURN r*fakultaet(r-1);
   ENDIF
END

AZUR (Real zahl)
   ergebnis := fakultaet (zahl)
   print (zahl, "! = ", ergebnis)
END
```

```
# AZUR-Programm mit selbstdefinierten Funktionen
addition (Real a, Real b) : Real
   RETURN a+b
END

multiplikation (Real a, Real b) : Real
   RETURN a*b
END

AZUR (Real r1, Real r2)
```

```
   Print (r1, " + ", r2, " = ", addition (r1, r2))
   ergebnis := multiplikation (r1, r2)
   Print (r1, " * ", r2, " = ", ergebnis)
END


# AZUR-Porgramm mit einer selbstdefinierten Prozedur
write (Zeitpunkt z, Distanz d)
   zpmode ("#d.#m.#Y")
   null_dist := ~"0 Tage"
   IF (d=null_dist)
      Print ("Der ", z, " ist heute.")
   ELSE
      IF (d<null_dist)
         text   := " war vor "
         faktor := -1
      ELSE
         text   := " ist in "
         faktor := 1
      ENDIF
      Print ("Der ", z, text, d*faktor, ".")
   ENDIF
END

AZUR (Zeitpunkt zp)
   dist  := ~"1 Tag"
   dist  := zp - @"Heute"
   write (zp, dist)
END
```

Kapitel 3

Aufruf des Compilers und des Linkers

3.1 Aufruf des Compilers-Interpreters AZUR

Der Name des Compiler-Interpreters ist azur. Wenn das Azur-Programm `prog.azr` heißt und keine Parameter übergeben werden , dann erfolgt der Aufruf mit:

```
azur prog.azr
```

Alle Ausgaben mittels`print`, erfolgen auf die Standardausgabe (Bildschirm). Dies ist die einfachste Form. Bietet das Betriebssystem keine Möglichkeit der Kommunikation über eine Shell, wie z.B. Microsoft Windows, dann muss eine Shell zusätzlich erworben werden. Ein Aufruf in einer DOS-Shell ist natürlich nicht möglich.

Parameter werden über die Kommandozeile übergeben. Jede Belegung eines Parameters hat die Form Parametername=Wert. Es können beliebig viele Parameter übergeben werden. Wenn `prog.azr` mit der Zeile

```
AZUR (String wo, Zeitpunkt wann)
```

beginnt, dann wäre ein typischer Aufruf:

```
azur prog.azr wo=Ahausen wann=15.9.1992
```

Die Zeichenkette `15.9.1992` wird automatisch in einen Zeitpunkt umgesetzt.

Enthält prog.azr mehrere Azur-Funktionen, die aus Gründen der Übersichtlichkeit zusammengefasst sind, dann wird über die Option `-f` Name die Azur-Funktion Name aufgerufen. Beispiel:

```
azur -f Ausgabe prog.azr
```

Das gesamte Programm wird compiliert, die Interpretation beginnt mit Ausgabe, statt mit AZUR.

Um ein Azur-Programm auf syntaktische Korrektheit zu testen, kann es sinnvoll sein, nur zu compilieren, jedoch nicht auszuführen. Dies geschieht mit

```
azur -c prog.azr
```

Das Compilieren und Interpretieren kann auch getrennt werden. Dies kann sinnvoll sein, um die Performance zu erhöhen oder um fertige Azur-Programme oder -Libraries dem Anwender zur Verfügung zu stellen (z.B. im Zusammenhang mit aqua_gramm). Mit

```
azur -c -ao prog.azr
```

wird das Produkt des Compilierens (Syntax-Baum und Scopes) in der Datei `prog.ao` abgelegt. Diese Datei kann dann im folgenden interpretiert werden mit

```
azur -i prog.ao
```

3.2 Aufruf des Linkers

Mit dem Aufruf `azur -o` werden mehrere ao-Dateien zu einer ao-Datei zusammengelinkt. Dies ist sinnvoll, wenn man fertige Azur-Funktionen immer wieder in anderen Azur-Programmen verwenden will.

Um auf Funktionen zuzugreifen, die aus einer anderen ao-Datei stammen, müssen diese bekannt gemacht werden. Dies geschieht entweder mittels EXTERN *Funktionskopf* pro benutzter Funktion oder mittels USES *Modulname* für alle Funktionen aus einem anderen Modul. Die zweite Variante ist vorzuziehen, da ein Parameter-Check automatisch stattfindet.

Mit der Option `-o name` wird der Name der Ausgabedatei festgelegt. Beispiel:

```
azur -o analyse starkregen.ao verbandslib.ao
```

Wenn in der Datei `starkregen.ao` die Funktion AZUR vorhanden ist, dann startet man das Programm `analyse` mit

```
azur -i analyse
```

3.3 Beispiel Makefile

Azur-Programme kann man auch mittels eines Makefiles compilieren und linken, so wie man das von anderen Programmen gewohnt ist. Ein Beispiel-Makefile (Unix) ist unten angegeben. In `Main.azr` ist die Hauptprozedur AZUR definiert. Das Programm startet man mit `azur -i Main`.

```
# Makefile for Main azur library
SUFFIXES= .ao .azr
.SUFFIXES:
.SUFFIXES: $(SUFFIXES)

.azr.ao:
azur -c -ao $<

SRCS=\
Pfocus.azr\
Prep_er.azr\
Prep_a.azr\
Pgebiet.azr\
Pdtpquit.azr\
Phandinp.azr

OBJS=$(SRCS:.azr=.ao)

all: Main.ao

Main.ao: $(OBJS)
azur -o Main.ao $(OBJS)

go: Main.ao
azur -i Main.ao
```

Kapitel 4

Vordefinierte Funktionen nach Kategorie

4.1 Konstanten

Folgende Konstanten sind vordefiniert:

Name	Typ	Wert
True	Bool	true
False	Bool	false
Luecke	Real	Real-Luecke
TextLuecke	Real	String-Luecke
Pi	Real	3.1415926...
EULERSCHEZAHL	Real	e
MAXQUAL	Real	maximale Qualität einer Zeitreihe
ORIGINAL	Real	0
KORRIGIERT	Real	1
GEFUELLT	Real	2
ANGEPASST	Real	3
MAXFOCUS	Intervall	Maximaler Focus einer Zeitreihe
MINUSINFTY	Real	-Infty für Realreihen
PLUSINFTY	Real	+Infty für Realreihen
PLUSMINUSINFTY	Real	+-Infty für Realreihen
CURRKEY	String	für Arrays (aktueller Key)
CURRVAL	String	für Arrays (aktueller Wert)

4.2 Allgemeine Funktionen

Herauszuheben ist hier die Funktion **System**. Mit ihr ist es auch möglich, weitere aqua_gramme oder Azurprogramme aufzurufen, Simulationsmodelle direkt anzustoßen, auch wenn sie nicht integraler Bestandteil der aqua_plan Software-Welt sind (aqua_flux ist sowohl von der Modellierung als auch von der graphischen Benutzerschnittstelle völlig integriert).

- isUnix
- Random
- Name
- Interprete
- System
- SystemIO
- GetPID
- GetEnv
- isValid
- SetName
- RealFormat
- zpmode
- GetGlobal
- SetInvalid
- SetEnv
- Str
- GStr
- RStr

- Beep
- SetGlobal
- TempName
- OnOff
- CompileStr
- Sleep
- SetLingua
- Lingua
- LinguaSort
- Username
- SendSignal
- CatchSignal
- Hostname
- CreateSem

4.2.1 Beispiel

```
# Beispielprogramm zu den allgemeinen Funktionen

AZUR (Zeitpunkt zp=@"11.11.88", Real zahl=1)
IF (IsValid (zp))
Print ("Der Zeitpunkt : ", zp)
zpmode ("#A, der #x. #i #Y, #w.#M Uhr")
zeitstring := Str (zp)
Print (zeitstring)
Print ()
Print ("Die Zahl : ", zahl)
RealFormat (".10f")
```

```
zahlstring := Str (zahl)
Print (zahlstring)
ELSE
System ("azur info.azr")
ENDIF
Print ()
Print ("Und die Zufallszahl lautet: ", Random ())
END
```

Ausgabe:

```
Der Zeitpunkt : 11.11.1988 07:30
Freitag, der 11. November 1988, 7.30 Uhr

Die Zahl : 1
1.0000000000

Und die Zufallszahl lautet: 0.1036071777
```

4.3 AxBox, Achsenkreuze

AxBoxen sind Achsenkreuze, die Zeitreihen (oder Realreihen) enthalten. Jede AxBox hat Y- und X-Achse. Diverse Einstellungen steuern das weitere Aussehen. AxBoxen können auf eine Page ausgegeben werden oder Plot an das aufrufende Auqagramm zurückgegeben werden (siehe Abschnitt 4.4).

4.3.1 Steuerung der Darstellung des Achsenkreuzes

- NewAxBox
- YKonstInAxBox
- ZRInAxBox
- AxBoxOnPage
- Plot
- SelYAx
- SetAxLage
- SetAxVoll
- SetAxLegPos
- SetAxClipping
- SetAxLueckeModus
- SetAxTextsize
- SetName
- SetAxHideBS
- SetAxGedreht
- SetAxScalDist
- SetAxGitter

- SetAxXBereich
- SetAxXInvers
- SetAxXUnit
- SetAxXLog
- SetAxXTexte
- SetAxXOben
- AxSetXGitter
- AxSetDist
- SetAxYBereich
- SetAxYStart0
- SetAxYInvers
- SetAxYUnit
- SetAxYLog
- SetAxYRechts
- AxSetYGitter
- AxReplaceYTexts
- SetAxFont
- AxClearKonsts
- AxDelZR
- ClearAxBox
- AxZRList
- Name
- AxLageRO

- AxLageLU
- AxLegPos
- AxXBereich
- AxYBereich
- AxInfo
- StrToAxBox
- AxSetMarker
- AxMarkerBereich

4.3.2 Steuerung der Darstellung von Zeitreihen

- SetLinienArt
- AxSetLineWidth
- AxSetTextsize
- AxSetSymSize
- SetIntervallArt
- SetLineStyle
- SetPunktArt
- SetTextVertical
- SetSymbolTyp
- SetZahlZeichnen
- AxZRColor
- AxZRLineWidth
- AxZRLineStyle

4.3.3 Zeichnen in Achsenkreuze

- AxDrawLine
- AxDrawSymbol
- AxDrawText
- AxDrawKreis
- AxSetColor
- AxSetAlign
- AxDelDrawing
- AxSetSymSize
- AxSetLineWidth
- AxSetTextsize

4.4 Kommunikation mit aqua_gramm

Wird das Azurprogramm aus einem aqua_gramm heraus aufgerufen, dann kann es mit diesem kommunizieren. aqua_gramm übergibt die Inhalte aller Steuerelemente über die Parameterliste der AZUR-Funktion. Texte (Text, Eingabe) kann man als String, Zeitpunkt, Zeitdistanz oder Real empfangen, eine Umwandlung findet automatisch statt. CheckBoxen werden als Bool und AxBoxen als AxBox übergeben.

Der Standard-Parameter Intervall Focus enthält den aktuell gewählten Zeitausschnitt. String Command den Namen (nicht die Aufschrift!) des gedrückten Buttons und ZRList zrsel die Liste aller dargestellten Zeitreihen.

Die Kommunikation in die andere Richtung, also von Azur nach aqua_gramm, erfolgt über die unten angeführten Funktionen. Die Arbeitsweise kann dem folgenden Beispiel entnommen werden.

- NewAGWindow
- DelAGWindow

- SetAGWindow
- GetAGWindow
- AGParent
- AGShow
- AGToTop
- AGSetActive
- ExistsAGWindow
- OkBox
- SelectBox
- InputBox
- ElementBox
- MultiBox
- ZIBox
- DruckerWahl
- ScreenSize
- PrintStatus
- AGSetShading
- NewButton
- NewText
- NewEingabe
- NewCheckBox
- NewSlider
- NewAuswahl

- NewTrigger
- NewMenu
- NewMenuBar
- NewMenuButton
- NewRahmen
- NewEditFeld
- NewListe
- NewDBListe
- NewDBGrid
- NewCombo
- NewLabel
- NewGeoCanvas
- NewCanvas
- DelCanvas
- DelAGElement
- SetHandle
- AddHandle
- RemoveHandle
- ThrowEvent
- SetHelpText
- AGElemPos
- AGElemSize
- AGSetElemPos

- AGSetElemSize
- GridPosTup
- SetActive
- SetChangeable
- AGSetFocus
- AGListe
- AGElemente
- AGRelabel
- AGSetTitel
- AGSetPos
- AGPos
- AGSize
- AGAxboxList
- CanvasOnPage
- GeoCanvasOnPage
- AGElemInfo
- SetAGElemInfo
- ImportVar
- ExportVar
- SetLegende
- GetLegende
- Zeilen
- Spalten

- ZRInAxBox
- Plot
- PlotKarte
- YKonstInAxBox
- ClearCanvas
- SetAutoScroll
- RedrawCanvas
- UpdateCanvas
- SetPointerType
- DoAt
- SetEditMode

Weitere Funktionen finden sich in Abschnitt 4.3.

4.4.1 Beispiel

```
# Beispielprogramm zu allen Funktionen dieser Kategorie

wird "uberarbeitet.
```

4.5 Ein- und Ausgabe (auf Datei)

- ExistsFile
- FileLesbar
- FileSchreibbar
- Prompt
- ReadLine
- ChangeDir
- Rewind
- FileForward
- PrintPage
- EndOfFile
- Print
- PrintFile
- Remove
- RemoveDir
- CloseFile
- Rename
- DirList
- FileList
- MakeDir
- GetDir
- FileDate

- ReadFile
- WriteFile
- FileSize
- FileCopy
- IsDirectory
- CreateTar
- ExtractTar

4.5.1 Beispiel

```
# Beispielprogramm zu allen Funktionen dieser Kategorie

AZUR ()
Print ("Hallo Welt.")
Print ()
Print ("Datei beschreiben ...")
i := 1
PrintFile ("printtest.dat", FALSE, "Zeile ", i, ": Hallo Welt.")
i := i + 1
text := "Dieser Text steht in einer Datei."
PrintFile ("printtest.dat", TRUE, "Zeile ", i, ": ", text)

Print ("Inhalt der Datei ausgeben ...")
WHILE (NOT EndOfFile ("printtest.dat"))
zeile := ReadLine ("printtest.dat")
Print (zeile)
ENDWHILE

Print ("Datei zur\ucksetzen ...")
Rewind ("printtest.dat")
Print ("Zeile lesen und ausgeben ...")
zeile := ReadLine ("printtest.dat")
Print (zeile)
```

```
page := NewPage (80, DINA4, PORTRAIT)
TextCenterOnPage (page, 4, "Die Seite", BOLD+BIG)
PrintPage (page, "printtest.ps", "PS")
END
```

Ausgabe:

```
Hallo Welt.

Datei beschreiben ...
Inhalt der Datei ausgeben ...
Zeile 1: Hallo Welt.
Zeile 2: Dieser Text steht in einer Datei.

Datei zuruecksetzen ...
Zeile lesen und ausgeben ...
Zeile 1: Hallo Welt.
```

4.6 Reelle Funktionen und Konstanten

In AZUR sind Integer- und Realzahlen zum AZUR-Typ Real zusammengefasst. Überall, wo als Parameter Integerzahlen erwartet werden, wird der AZUR-Typ Real verwendet.

- Sin
- Cos
- Tan
- Log
- Abs
- Mod
- IngRunden
- ArcSin
- ArcCos
- ArcTan
- LnGamma
- RMax
- RMin
- RealToRaw
- RawToReal
- BitMask
- HexToReal
- PI
- EULERSCHEZAHL
- LUECKE
- Operatoren

4.6.1 Beispiel

```
# Beispielprogramm zu reellen Funktionen und Konstanten

anpassen (Real zahl) : Real
# zahl gleich null setzen, wenn |zahl| < genauigkeit
   genauigkeit := 0.000000001
   IF (zahl>-genauigkeit AND zahl<genauigkeit)
      RETURN 0
   ELSE
      RETURN zahl
   ENDIF
END

print_trigonom (Real teiler_pi)
   IF (teiler_pi=0)
      text := "   "
      arg  := 0
   ELSE
      text := "PI/"
      arg  := PI/teiler_pi
   ENDIF
   sinus   := anpassen (Sin(arg))
   cosinus := anpassen (Cos(arg))
   l1 := Length (Str(sinus))
   d1 := 10-l1
   print (text, teiler_pi, "  |  ", sinus, d1*" ", cosinus)
END

AZUR ()
   Print ("PI=", PI)
   Print ()
   # Tabelle wichtiger Funktionswerte von Sinus und Cosinus
   Print (9*" ", "Sinus", 5*" ", "Cosinus")
   print_trigonom (0)
   print_trigonom (6)
   print_trigonom (4)
   print_trigonom (3)
```

```
   print_trigonom (2)
   Print ()
   zahl := PI/2
   Print ("ArcSin (Sin (", zahl, ")) = ", anpassen (ArcSin(Sin(zahl))))
   Print ("ArcCos (Cos (", zahl, ")) = ", anpassen (ArcCos(Cos(zahl))))
   Print ()
   ergebnis1 := anpassen (Sin(zahl))
   ergebnis2 := anpassen (Cos(zahl))
   Print ("Sin (", zahl, ") = ", ergebnis1)
   Print ("Cos (", zahl, ") = ", ergebnis2)
   Print ("ArcSin (", ergebnis1, ") = ", anpassen (ArcSin(ergebnis1)))
   Print ("ArcCos (", ergebnis2, ") = ", anpassen (ArcCos(ergebnis2)))
   Print ()
   zahl := PI/4
   erg  := anpassen (Tan(zahl))
   Print ("Tan (", zahl, ") = ", erg)
   Print ("ArcTan (", erg, ") = ", anpassen (ArcTan(erg)))
   Print ()
   zahl := 2.71828
   Print ("Log (", zahl, ") = ", Log(zahl))
END
```

Ausgabe:

```
PI=3.14159

         Sinus     Cosinus
   0  |  0         1
PI/6  |  0.5       0.866025
PI/4  |  0.707107  0.707107
PI/3  |  0.866025  0.5
PI/2  |  1         0

ArcSin (Sin (1.5708)) = 1.5708
ArcCos (Cos (1.5708)) = 1.5708

Sin (1.5708) = 1
Cos (1.5708) = 0
```

```
ArcSin (1) = 1.5708
ArcCos (0) = 1.5708

Tan (0.785398) = 1
ArcTan (1) = 0.785398

Log (2.71828) = 0.999999
```

4.6.2 Beispiel

```
# Beispielprogramm zu den reellen Operatoren + - * / ^

AZUR (Real r1=10, Real r2=4)
   Print ("Zahl1: ", r1)
   Print ("Zahl2: ", r2)
   Print ()
   Print ("Summe     :  ", r1+r2)
   Print ("Differenz :  ", r1-r2)
   Print ("Produkt   :  ", r1*r2)
   IF (r2=0)
      text := "-- (Division durch Null)"
   ELSE
      text := Str (r1/r2)
   ENDIF
   Print ("Quotient  :  ", text)
   Print ()
   Print ("Quadrat von ", r1, "  : ", r1^2)
END
```

Ausgabe:

```
Zahl1: 10
Zahl2: 4

Summe     :  14
Differenz :  6
Produkt   :  40
```

```
Quotient  :  2.5

Quadrat von 10  : 100
```

4.7 Funktionen auf dem Azur-Typ String

Der AZUR-Typ String ist eine Zeichenkette beliebiger Länge. Ein String darf kein % enthalten, stattdessen muss die Escape-Sequenz `\p` verwendet werden.

- IsReal
- StrToReal
- StrSort
- Token
- UpCase
- Length
- SubStr
- StrHead
- StrTail
- StrSplit
- StrAnz
- LowCase
- Pos
- RevPos
- SIGroesse
- WildcardMatch
- RegExpMatch
- Str
- StrToArray
- Replace

- Trim
- Char
- CharVal
- StrEncrypt
- StrDecrypt
- StrHunz
- StrToBase64
- Base64ToStr
- MD5Sum
- StrHex
- StrFromHex
- HexToReal
- Operatoren

4.7.1 Beispiel

```
# Beispielprogramm zu den String-Funktionen Length, UpCase, LowCase und
# den Operatoren * +

AZUR (String text="Nur einen winzigen Schluck!")
# Laenge des Textes bestimmen
laenge := Length (text)
# Text ausgeben und unterstreichen
Print (text)
Print (laenge*"-")
grosstext := UpCase (text)
kleintext := LowCase (text)
Print (grosstext)
Print (kleintext)
```

```
langtext := grosstext + " " + kleintext
Print (langtext)
END
```

Ausgabe:

```
Nur einen winzigen Schluck!
---------------------------
NUR EINEN WINZIGEN SCHLUCK!
nur einen winzigen schluck!
NUR EINEN WINZIGEN SCHLUCK! nur einen winzigen schluck!
```

4.7.2 Beispiel

```
# Beispielprogramm zu den String-Funktionen Str, StrToReal
# und zum Operator +

AZUR (String text="3.14159)
Print ("Am Anfang war der Text     : ", text)
Print ()
# Umwandlung einer Zahl in einen String und umgekehrt
zahl := StrToReal(text)
Print ("Aus dem Text wurde die Zahl: ", zahl)
Print ("Reelle Addition            : ", zahl, " + 111 = ", zahl+111)
Print ()
ziffs := Str (zahl)
Print ("Aus der Zahl wurde der Text: ", ziffs)
Print ("String-Addition            : ", ziffs, " + 111 = ", ziffs+"111")
END
```

Ausgabe:

```
Am Anfang war der Text     : 3.14159

Aus dem Text wurde die Zahl: 3.14159
Reelle Addition            : 3.14159 + 111 = 114.142
```

```
Aus der Zahl wurde der Text: 3.14159
String-Addition            : 3.14159 + 111 = 3.14159111
```

4.7.3 Beispiel

```
# Beispielprogramm zu den String-Funktionen Pos, SubStr und
# den Operatoren + -

AZUR ()
text := "Es gr\o\unt so gr\o\un, wenn Spaniens Bl\o\uten bl\o\uhen."
Print (text)
Print ()
umlaut := "\o"
# Die "\o"s entfernen
text1    := " "
teiltext := text
position := Pos (teiltext, umlaut)
WHILE (position>-1)
# Den Teilstring vom Anfang bis zum ersten "\o" ausschliesslich an
# den String text1 anhaengen
text1 := text1 + SubStr(teiltext, 0, position-1)
Print (text1)
# Den Teilstring verkuerzen
teiltext := SubStr(teiltext,position+Length(umlaut),Length(teiltext)-1)
position := Pos (teiltext, umlaut)
ENDWHILE
text1 := text1 + teiltext
text1 := text1 - " "
Print ()
Print (text1)
END
```

Ausgabe:

```
Es gröünt so gröün, wenn Spaniens Blöüten blöühen.

  Es gr
```

```
  Es grünt so gr
  Es grünt so grün, wenn Spaniens Bl
  Es grünt so grün, wenn Spaniens Blüten bl

Es grünt so grün, wenn Spaniens Blüten blühen.
```

4.7.4 Beispiel

```
# Beispielprogramm zu der String-Funktion SIGroesse

AZUR ()
groesse_meter := "1000 m"
Print (groesse_meter, " <==> ", SIGroesse (groesse_meter))
groesse_newton := "1 N"
Print (groesse_newton, " <==> ", SIGroesse (groesse_newton))
END
```

Ausgabe:

```
1000 m <==> km
1 N <==> km/s^2*g
```

4.8 Funktionen auf Zeitpunkt, Distanz und Intervall

- IsZP
- IsZD
- Sekunde
- Minute
- Stunde
- Tag
- Monat
- Jahr
- WWJ
- ZPAbrunden
- Breite
- Links
- Rechts
- LinksReal
- RechtsReal
- DistStunden
- DistStr
- ZPStr
- ZIStr
- MaxFocusZR
- MaxTextFocusZR

- GetFocus
- MAXFOCUS
- SetZPRaster

4.8.1 Beispiel

```
# Beispielprogramm zu den Funktionen WWJ, Breite, Links und Rechts

AZUR (Real jahr=1994)
IF (IsValid (jahr))
wwjahr := WWJ (jahr)
Print ("WWJahr\n\t", wwjahr)
br := Breite (wwjahr)
Print ("\tBreite des Intervalls : ", br)
Print ()
# Intervall erweitern
neuer_bereich := [Links(wwjahr)-~"1 Monat", Rechts(wwjahr)+~"1 Monat"]
Print ("Intervall an beiden Seiten um 1 Monat erweitert")
Print ("\t", neuer_bereich)
br := Breite (neuer_bereich)
Print ("\tBreite des Intervalls : ", br)
ELSE
Print ("Ung\ultiges Jahr!")
ENDIF
END
```

Ausgabe:

```
WWJahr
[01.11.1993 07:30,01.11.1994 07:30]
Breite des Intervalls : 365d

Intervall an beiden Seiten um 1 Monat erweitert
[01.10.1993 07:30,01.12.1994 07:30]
Breite des Intervalls : 426d
```

4.8.2 Beispiel

```
# Beispielprogramm zu der Funktion ZPAbrunden

AZUR (Zeitpunkt zp=@"11.11.1988 11:11")
Print ("Der Zeitpunkt  ", zp, "  abgerundet auf")
Print ()
zp_stunde:= ZPAbrunden (zp, ~"1 Stunde")
zp_tag   := ZPAbrunden (zp, ~"1 Tag")
zp_woche := ZPAbrunden (zp, ~"1 Woche")
zp_monat := ZPAbrunden (zp, ~"1 Monat")
zp_jahr  := ZPAbrunden (zp, ~"1 Jahr")
Print ("die Stunde   : ", zp_stunde)
Print ("den Tag      : ", zp_tag)
Print ("die Woche    : ", zp_woche)
Print ("den Monat    : ", zp_monat)
Print ("das Jahr     : ", zp_jahr)
END
```

Ausgabe:

```
Der Zeitpunkt  11.11.1988 11:11  abgerundet auf

die Stunde   : 11.11.1988 11:00
den Tag      : 11.11.1988 00:00
die Woche    : 07.11.1988 00:00
den Monat    : 01.11.1988 00:00
das Jahr     : 01.01.1988 00:00
```

4.8.3 Beispiel

```
# Beispielprogramm zu der Funktion DistStunden

AZUR ()
Print ("1 Tag   : ", DistStunden (~"1 Tag"), "   Stunden")
Print ("1 Woche : ", DistStunden (~"1 Woche"), "  Stunden")
Print ("1 Monat : ", DistStunden (~"1 Monat"), "  Stunden")
```

```
Print ("1 Jahr  : ", DistStunden (~"1 Jahr"), " Stunden")
Print ()
Print ("Zu beachten: Monat und Jahr habe keine feste L\ange!")
zp1 := @"1.1.1992"
zp2 := zp1 + ~"1 Monat"
dist := zp2 - zp1
zpmode ("#d.#m.#y")
Print (zp1, " + 1 Monat = ",zp2," ===> ", DistStunden(dist)," Stunden")
zp2 := zp1 + ~"1 Jahr"
dist := zp2 - zp1
Print (zp1, " + 1 Jahr  = ",zp2," ===> ", DistStunden(dist)," Stunden")
END
```

Ausgabe:

```
1 Tag   : 24   Stunden
1 Woche : 168  Stunden
1 Monat : 720  Stunden
1 Jahr  : 8760 Stunden

Zu beachten: Monat und Jahr habe keine feste Länge!
01.01.92 + 1 Monat = 01.02.92 ===> 744 Stunden
01.01.92 + 1 Jahr  = 01.01.93 ===> 8784 Stunden
```

4.8.4 Beispiel

```
# Beispielprogramm zu den Funktionen GetFocus, MaxFocusZR und
# der Konstanten MAXFOCUS

AZUR (ZRList zrl)
Print ("Die ZRListe :")
Print (zrl)
Print ()
bereich := GetFocus (zrl)
Print ("Focus der ZRListe : ", bereich)
Print ()
i := 1
```

```
FORALL zr In zrl
Print ("ZReihe ", i, " : ", zr)
i := i + 1
zr_bereich := MaxFocusZR (zr)
Print ("\t\tMaxFocus: ", zr_bereich)
br := Breite (zr_bereich)
dist := DistStunden (br)
Print ("\t\t", dist, " Stunden")
maximum := Max (zr, MAXFOCUS)
Print ("\t\tMaximum : ",maximum)
ENDFOR
END
```

Ausgabe:

```
Die ZRListe :
05.04.1992 12:59:00
01.11.1992 07:00:00
lucky1.lk0
lucky2.lk0
lucky3.lk0
lucky4.lk0

Focus der ZRListe : [05.04.1992 12:59:00,01.11.1992 07:00:00]

ZReihe 1 : (Lucky1,Luke,,,0,,Z)
MaxFocus: [05.04.1992 12:59:55,01.11.1992 07:30:00]
5034.5 Stunden
Maximum : 5
ZReihe 2 : (Lucky2,Luke,,,0,,Z)
MaxFocus: [05.04.1992 12:59:55,01.11.1992 07:00:00]
5034 Stunden
Maximum : 4
ZReihe 3 : (Lucky3,Luke,,,0,,Z)
MaxFocus: [05.04.1992 12:59:55,01.11.1992 07:00:00]
5034 Stunden
Maximum : 2
ZReihe 4 : (Lucky4,Luke,,,0,,Z)
```

```
MaxFocus: [05.04.1992 12:59:55,01.11.1992 07:00:00]
5034 Stunden
Maximum : 1
```

4.9 Funktionen für den ADB-Manager

- ADBInit
- ADBUpdate
- ADBFlush
- ADBChangeFields
- Stammdaten
- ADBQuery
- StammdatenByFeld
- ADBStammRel
- ADBBenutzerRel
- ADBZROrt
- ADBName
- ADBBezeichnung
- ADBTupZROrt
- Benutzer
- UpdateBenutzer
- ADBLockTupel
- ADBUnlockTupel
- ADBListe
- ADBKeyfeld
- ADBNameFeld
- ADBOpenRel

- ADBExists
- ADBCreateRel
- ADBDeCache
- ADBEmptyCache
- ADBFlushCache
- ADBUpdateCache
- ADBSetResolveInfo
- ADBResolveField
- ADBUnlockAll

Beispiel

```
# Beispielprogramm

AZUR ()
ADBInit ("stamnied.dbf", "ORT", "benutzer.dbf","ORT","ORT","NAME","BESON_ORT|LAND_ORT

tup1 := Stammdaten ("Monschau")
tup2 := StammdatenByKey ("Monschau", "ORT") # gueltig
tup2 := StammdatenByKey ("Monschau", "NAME") # ungueltig

s := ADBZROrt ("Monschau") # -> "Monschau"
s := ADBName ("Monschau") # -> "Station Monschau Bruecke"
s := ADBName ("Station Monschau Bruecke") # -> "Monschau"
```

4.10 Funktionen auf ZR

4.10.1 Fremdformate

- Export
- Import
- ImportListe
- ExportListe
- ZRFormat
- ImpCSV
- PackRead
- PackHeader
- PackInfo
- PackWrite

Beispiel

```
# Beispielprogramm zu den Funktionen Import, Export

AZUR ()
zr := GetZR ("1119", "Niederschlag", "", "K")
Print (zr)
Print ("Max. Focus der Zeitreihe : ", MaxFocusZR (zr))
bereich := [@"1.1.80", @"1.1.81"]
Print ("Zu exportierender Bereich: ", bereich)
Print ()
outfile := "1119exp.asc"
kommando := "ls 1119exp.asc"
Print (kommando)
System (kommando)
Print ()
```

```
Print ("Export ...")
Export (zr, bereich, 0, "ASCII", outfile)
Print ()
Print (kommando)
System (kommando)
Print ()
Print ("Import ...")
zr2 := Import (outfile)
Print ()
Print ("Importierte Zeitreihe: ", zr2)
END
```

Ausgabe:

```
(1119,Niederschlag,Sum,M,,G,Z)
Max. Focus der Zeitreihe : [01.01.1952 10:05,01.11.1987 13:00]
Zu exportierender Bereich: [01.01.1980 07:30,01.01.1981 07:30]

ls 1119exp.asc
1119exp.asc not found

Export ...

ls 1119exp.asc
1119exp.asc

Import ...
aquaplan Ascii-Format 1119exp.asc

Importierte Zeitreihe: (1119,Niederschlag,Sum,M,,G,Z)
```

4.10.2 Öffnen von Zeitreihen

- SetDatenpool
- OpenZRId
- Query
- OrtQuery
- ParamQuery
- GetZR
- GetDatenpool
- OpenZR
- ZRSetBurst
- ZRSchreibbar

Beispiel

```
# Beispielprogramm zu allen Funktionen dieser Kategorie

AZUR ()
rt := ReiheTupel ()
zr1 := OpenZR (rt)
Print ("Eine neue ZR: ", zr1)
zr2 := GetZR ("1119", "Niederschlag", "", "K")
Print ("Eine bestimmte ZR: ", zr2)
Print ()
zr3liste := OrtQuery ("99999")
Print ("Eine Liste von ZR mit dem Ort 99999:")
FORALL zr3 IN zr3liste
Print (zr3)
ENDFOR
Print ()
zr4liste := ParamQuery ("Temperatur")
```

```
Print ("Eine Liste von ZR mit dem Parameter Temperatur:")
FORALL zr4 IN zr4liste
Print (zr4)
ENDFOR
END
```

Ausgabe:

```
Eine neue ZR: (,,,,0,,)
Eine bestimmte ZR: (1119,Niederschlag,Sum,M,,G,Z)

Eine Liste von ZR mit dem Ort 99999:
(99999,Niederschlag,Lck,,0,A,Z)
(99999,Niederschlag,Abl,,0,A,Z)

Eine Liste von ZR mit dem Parameter Temperatur:
(1190,TEMPERATUR,,T,,G,Z)
(2205,TEMPERATUR,,T,,G,Z)
(2211,TEMPERATUR,,T,,G,Z)
```

4.10.3 Veränderung einer Zeitreihe

- AddWP
- WriteQuantenfolge
- WriteTextQuantenfolge
- StoreQF
- StoreTextQF
- Transpose

Beispiel

```
# Beispielprogramm zu den Funktionen AddWP, Transpose

AZUR ()
t := ReiheTupel ()
setText (t, "Parameter", "Niederschlag")
setText (t, "DefArt", "I")
setText (t, "Einheit", "mm/h")
zp := @"11.11.88"
zr := OpenZR (t)
i := 0
WHILE (i < 5)
AddWP (zr, zp, i*11)
zp := zp + ~"1 Tag"
i := i + 1
ENDWHILE
Print (zr)
Print ("Einheit : ", Einheit(zr))
Print ()
qf := Quantenfolge (zr, MAXFOCUS)
Print (qf)
Print ()
Transpose (zr, "l/(s*ha)")
Print (zr)
```

```
Print ("Einheit : ", Einheit(zr))
Print ()
qf := Quantenfolge (zr, MAXFOCUS)
Print (qf)
END
```

Ausgabe:

```
(,Niederschlag,,,0,,)
Einheit : mm/h

IntervallQuant 0 auf (-Infty , 11.11.1988 07:30]
IntervallQuant 11 auf (11.11.1988 07:30 , 12.11.1988 07:30]
IntervallQuant 22 auf (12.11.1988 07:30 , 13.11.1988 07:30]
IntervallQuant 33 auf (13.11.1988 07:30 , 14.11.1988 07:30]
IntervallQuant 44 auf (14.11.1988 07:30 , 15.11.1988 07:30]


(,Niederschlag,,,0,,)
Einheit : l/(s*ha)

IntervallQuant 0 auf (-Infty , 11.11.1988 07:30]
IntervallQuant 30.555553 auf (11.11.1988 07:30 , 12.11.1988 07:30]
IntervallQuant 61.111107 auf (12.11.1988 07:30 , 13.11.1988 07:30]
IntervallQuant 91.666664 auf (13.11.1988 07:30 , 14.11.1988 07:30]
IntervallQuant 122.22221 auf (14.11.1988 07:30 , 15.11.1988 07:30]
```

4.10.4 Abfragen von Zeitreihenwerten

- Quantenfolge
- TextQuantenfolge
- MaxFocusZR
- MaxTextFocusZR
- YWert
- YTextWert
- ZPQuant
- AnzahlWerte
- SetQualitaet
- LueckenProz
- Qualitaetsfolge
- MaxQualitaet
- WerteQF
- ModifiedSince
- ModificationTable
- ModTabCompact

Beispiel

```
# Beispielprogramm zu allen Funktionen dieser Kategorie

AZUR ()
zrl := ParamQuery ("Temperatur")
zr := FirstZR (zrl)
einh := Einheit (zr)
```

```
bereich := [@"1.1.80", @"8.1.81"]
FORALL zr IN zrl
Print (zr)
Print ("Anzahl der Werte: ", AnzahlWerte (zr, MAXFOCUS))
Print ()
Print ("Minimum:")
Print (Min (zr, MAXFOCUS), einh, "   ", MinZP (zr, MAXFOCUS))
Print ()
Print ("Maximum:")
maxzeitp := MaxZP (zr, MAXFOCUS)
Print (Max (zr, MAXFOCUS), einh, "   ", maxzeitp)
Print ("YTextWert: ", YTextWert (zr, maxzeitp))
Print ("YWert eine Woche sp\ater: ", YWert (zr, maxzeitp+~"1 Woche"))
Print ()
Print ("Arith. Mittel:  ", Mittel (zr, MAXFOCUS))
Print ()
Print (bereich)
Print ("Summe:          ", Summe (zr, bereich))
Print ("L\uckenanteil:  ", LueckenProz (zr, bereich))
Print ()
Print ()
ENDFOR
END
```

Ausgabe:

```
(1190,TEMPERATUR,,T,,G,Z)
Anzahl der Werte: 12421

Minimum:
-9.9°C   01.02.1954 00:00

Maximum:
28.4°C   02.07.1976 00:00
YTextWert: LU
YWert eine Woche später: 21.4

Arith. Mittel:   9.57766
```

```
[01.01.1980 07:30,08.01.1981 07:30]
Summe:         0
Lückenanteil:  0


(2205,TEMPERATUR,,T,,G,Z)
Anzahl der Werte: 14247

Minimum:
-9.9°C   30.01.1956 00:00

Maximum:
28.6°C   08.07.1959 00:00
YTextWert: LU
YWert eine Woche später: 16.6

Arith. Mittel:   9.75488

[01.01.1980 07:30,08.01.1981 07:30]
Summe:         82756.5
Lückenanteil:  0


(2211,TEMPERATUR,,T,,G,Z)
Anzahl der Werte: 14247

Minimum:
-9.9°C   04.01.1971 00:00

Maximum:
27°C   30.07.1983 00:00
YTextWert: LU
YWert eine Woche später: 11.4

Arith. Mittel:   7.86748

[01.01.1980 07:30,08.01.1981 07:30]
Summe:         64278.9
Lückenanteil:  0
```

4.10.5 Verknüpfen von Zeitreihen

- AddGroesse
- SubGroesse
- AddZR
- SubZR
- MulGroesse
- MulZR
- DivGroesse
- DivZR
- ExpWert
- ZRFolge
- FolgenRel
- FolgenIDs

Beispiel

```
# Beispielprogramm zu den Funktionen AddGroesse, SubGroesse,
# MulGroesse, DivGroesse

AZUR ()
zr := GetZR ("4311001", "Niederschlag", "", "I")
Print (zr)
RealFormat ("10.2f")
zpmode ("#d.#m.#Y")
bereich := [@"15.1.75", @"25.1.75"]
Print ("Bereich : ", bereich)
Print ()
Print ("Original")
qf := Quantenfolge (zr, bereich)
```

```
FORALL q IN qf
Print (XBereich(q), "  ", YRechts(q))
ENDFOR
Print ()

Print ("AddGroesse")
sum := AddGroesse (zr, "100 mm", bereich, "nieder1", true)
Print (sum)
Print ("Einheit: ", Einheit(sum))
qf := Quantenfolge (sum, MAXFOCUS)
FORALL q IN qf
Print (XBereich(q), "  ", YRechts(q))
ENDFOR
Print ()

Print ("SubGroesse")
diff := SubGroesse (zr, "100 mm", bereich, "nieder2", true)
Print (diff)
Print ("Einheit: ", Einheit(diff))
qf := Quantenfolge (diff, bereich)
FORALL q IN qf
Print (XBereich(q), "  ", YRechts(q))
ENDFOR
Print ()

Print ("MulGroesse")
RealFormat ("14.6f")
prod := MulGroesse (zr, "100 mm", bereich, "nieder3", true)
Print (prod)
Print ("Einheit: ", Einheit(prod))
qf := Quantenfolge (prod, bereich)
FORALL q IN qf
Print (XBereich(q), "  ", YRechts(q))
ENDFOR
Print ()

Print ("DivGroesse")
RealFormat ("10.2f")
```

```
quot := DivGroesse (zr, "100 mm", bereich, "nieder4", true)
Print (quot)
Print ("Einheit: ", Einheit(quot))
qf := Quantenfolge (quot, bereich)
FORALL q IN qf
Print (XBereich(q), "  ", YRechts(q))
ENDFOR
END
```

Ausgabe:

```
(4311001,Niederschlag,Sum,T,,A,Z)
Bereich : [15.01.1975,25.01.1975]

Original
[15.01.1975,16.01.1975]        0.00
[16.01.1975,17.01.1975]       10.93
[17.01.1975,18.01.1975]        2.07
[18.01.1975,19.01.1975]        6.00
[19.01.1975,20.01.1975]        2.00
[20.01.1975,21.01.1975]        1.00
[21.01.1975,22.01.1975]        9.00
[22.01.1975,23.01.1975]       67.00
[23.01.1975,24.01.1975]       20.00
[24.01.1975,25.01.1975]        0.00

AddGroesse
(4311001,nieder1,Sum,T,,M,Z)
Einheit: mm
[15.01.1975,16.01.1975]      100.00
[16.01.1975,17.01.1975]      110.93
[17.01.1975,18.01.1975]      102.07
[18.01.1975,19.01.1975]      106.00
[19.01.1975,20.01.1975]      102.00
[20.01.1975,21.01.1975]      101.00
[21.01.1975,22.01.1975]      109.00
[22.01.1975,23.01.1975]      167.00
[23.01.1975,24.01.1975]      120.00
```

```
[24.01.1975,25.01.1975]      100.00

SubGroesse
(4311001,nieder2,Sum,T,,M,Z)
Einheit: mm
[15.01.1975,16.01.1975]     -100.00
[16.01.1975,17.01.1975]      -89.07
[17.01.1975,18.01.1975]      -97.93
[18.01.1975,19.01.1975]      -94.00
[19.01.1975,20.01.1975]      -98.00
[20.01.1975,21.01.1975]      -99.00
[21.01.1975,22.01.1975]      -91.00
[22.01.1975,23.01.1975]      -33.00
[23.01.1975,24.01.1975]      -80.00
[24.01.1975,25.01.1975]     -100.00

MulGroesse
(4311001,nieder3,Sum,T,,M,Z)
Einheit: m^2
[15.01.1975,16.01.1975]        0.000000
[16.01.1975,17.01.1975]        0.001093
[17.01.1975,18.01.1975]        0.000207
[18.01.1975,19.01.1975]        0.000600
[19.01.1975,20.01.1975]        0.000200
[20.01.1975,21.01.1975]        0.000100
[21.01.1975,22.01.1975]        0.000900
[22.01.1975,23.01.1975]        0.006700
[23.01.1975,24.01.1975]        0.002000
[24.01.1975,25.01.1975]        0.000000

DivGroesse
(4311001,nieder4,Sum,T,,M,Z)
Einheit:
[15.01.1975,16.01.1975]        0.00
[16.01.1975,17.01.1975]        0.11
[17.01.1975,18.01.1975]        0.02
[18.01.1975,19.01.1975]        0.06
[19.01.1975,20.01.1975]        0.02
```

```
[20.01.1975,21.01.1975]        0.01
[21.01.1975,22.01.1975]        0.09
[22.01.1975,23.01.1975]        0.67
[23.01.1975,24.01.1975]        0.20
[24.01.1975,25.01.1975]        0.00
```

Beispiel

```
# Beispielprogramm zu den Funktionen AddZR, SubZR, MulZR, DivZR

AZUR ()
zr1 := GetZR ("4311001", "Niederschlag", "", "I")
Print (zr1)
zr2 := GetZR ("4604003", "Niederschlag", "", "I")
Print (zr2)
RealFormat ("10.2f")
zpmode ("#d.#m.#Y")
bereich := [@"15.1.75", @"25.1.75"]
Print ("Bereich : ", bereich)
Print ()

sumzr := AddZR (zr1, zr2, bereich, "niederadd", true)
text := "Summe"
laenge := Length (text)
Print (text, "\n", laenge*"-")
Print ("Einheit: ", Einheit(sumzr))
qf := Quantenfolge (sumzr, bereich)
FORALL q IN qf
Print (XBereich(q), "  ", YRechts(q))
ENDFOR
Print ()

diffzr := SubZR (zr1, zr2, bereich, "niedersub", true)
text := "Differenz"
laenge := Length (text)
Print (text, "\n", laenge*"-")
Print ("Einheit: ", Einheit(diffzr))
qf := Quantenfolge (diffzr, bereich)
```

```
FORALL q IN qf
Print (XBereich(q), "  ", YRechts(q))
ENDFOR
Print ()

prodzr := MulZR (zr1, zr2, bereich, "niedermul", true)
text := "Produkt"
laenge := Length (text)
Print (text, "\n", laenge*"-")
Print ("Einheit: ", Einheit(prodzr))
qf := Quantenfolge (prodzr, bereich)
FORALL q IN qf
Print (XBereich(q), "  ", YRechts(q))
ENDFOR
Print ()

quotzr := DivZR (zr1, zr2, bereich, "niederdiv", true)
text := "Quotient"
laenge := Length (text)
Print (text, "\n", laenge*"-")
Print ("Einheit: ", Einheit(quotzr))
qf := Quantenfolge (quotzr, bereich)
FORALL q IN qf
Print (XBereich(q), "  ", YRechts(q))
ENDFOR
END
```

Ausgabe:

```
(4311001,Niederschlag,Sum,T,,A,Z)
(4604003,Niederschlag,Sum,T,,A,Z)
Bereich : [15.01.1975,25.01.1975]

Summe
-----
Einheit: mm
[15.01.1975,16.01.1975]        0.00
[16.01.1975,17.01.1975]       13.23
```

```
[17.01.1975,18.01.1975]       17.95
[18.01.1975,19.01.1975]       18.19
[19.01.1975,20.01.1975]        2.00
[20.01.1975,21.01.1975]        7.13
[21.01.1975,22.01.1975]       22.10
[22.01.1975,23.01.1975]      108.10
[23.01.1975,24.01.1975]       24.20
[24.01.1975,25.01.1975]        0.00

Differenz
---------
Einheit: mm
[15.01.1975,16.01.1975]        0.00
[16.01.1975,17.01.1975]        8.63
[17.01.1975,18.01.1975]      -13.81
[18.01.1975,19.01.1975]       -6.19
[19.01.1975,20.01.1975]        2.00
[20.01.1975,21.01.1975]       -5.13
[21.01.1975,22.01.1975]       -4.10
[22.01.1975,23.01.1975]       25.90
[23.01.1975,24.01.1975]       15.80
[24.01.1975,25.01.1975]        0.00

Produkt
-------
Einheit: mm^2
[15.01.1975,16.01.1975]        0.00
[16.01.1975,17.01.1975]       25.14
[17.01.1975,18.01.1975]       32.87
[18.01.1975,19.01.1975]       73.14
[19.01.1975,20.01.1975]        0.00
[20.01.1975,21.01.1975]        6.13
[21.01.1975,22.01.1975]      117.90
[22.01.1975,23.01.1975]     2753.70
[23.01.1975,24.01.1975]       84.00
[24.01.1975,25.01.1975]        0.00

Quotient
```

```
--------
Einheit:
[15.01.1975,16.01.1975]  Lücke
[16.01.1975,17.01.1975]        4.75
[17.01.1975,18.01.1975]        0.13
[18.01.1975,19.01.1975]        0.49
[19.01.1975,20.01.1975]  Lücke
[20.01.1975,21.01.1975]        0.16
[21.01.1975,22.01.1975]        0.69
[22.01.1975,23.01.1975]        1.63
[23.01.1975,24.01.1975]        4.76
[24.01.1975,25.01.1975]  Lücke
```

4.10.6 Starkregenanalyse

- Regenhoehenlinie
- Verteilungsparameter
- PartielleSerie
- JaehrlicheSerie
- Owunda

Beispiel

```
# Beispielprogramm zur Starkregenanalyse

AZUR ()
bereich := [@"15.1.75",@"25.1.85"]

# kontinuierliche Zeitreihe
zr := GetZR ("801987","Niederschlag","","K")
# Tagessummen-Zeitreihe oeffnen
zrt := GetZR ("801987","Niederschlag","","I", "", "Z", ~"1 Tag")

# Partielle Serie bilden (temporaer)
pserie := PartielleSerie (zr, zrt, bereich, true)

# u. U. die partielle Serie editieren (Ausreisser eliminieren)

# Parameter u und w der Verteilungsfunktionen bestimmen (temporaer)
# u doppelt und w einfach logarithmisch
vp := Verteilungsparameter (pserie, bereich, true, false, true)

# Regenhoehenlinie fuer die Jaerhlichkeit 30a berechnen
rhl := Regenhoehenlinie (vp, bereich, 30, false)
END
```

4.10.7 Statistik auf W- und Q-Reihen

- WnachQ
- HJahrSerie
- HEreignisse
- WnachQHZB
- HPartSerie
- NJahrSerie
- NEreignisse
- NPartSerie
- QvonW

Beispiel

```
# Beispielprogramm zu der Funktion WnachQ

AZUR ()
t := ReiheTupel ()
setText (t, "Ort", "24005107")
setText (t, "Parameter", "Wasserstand")
zrl := Query (t)
zpmode ("#d.#m.#y")

FORALL zr IN zrl
Print ("Wasserstands-Zeitreihe:")
Print (zr)
Print ("Bereich: ", MaxFocusZR (zr))
Print ("Einheit: ", Einheit (zr))
Print ()
qf  := Quantenfolge (zr, MAXFOCUS)
Print ("Abflu\s-Zeitreihe:")
wq := WnachQ (zr, MAXFOCUS, "STAU", false)
```

```
Print (wq)
Print ("Bereich: ", MaxFocusZR (wq))
Print ("Einheit: ", Einheit (wq))
Print ()
ENDFOR
END
```

4.10.8 Lücken füllen von Intensitätsreihen

- FindeEreignis
- ExtrahiereEreignis
- SynthetisiereEreignis
- NextLuecke

Beispiel

```
# Beispielprogramm, siehe auch Modul AquaTrop
AZUR ()
   nzr := GetZR ("Ahausen", "Niederschlag", "", "K")
   auffuellzr := GetZR ("Behausen", "Niederschlag", "", "K")
   bereich := WWJ(1998)
   lueckber := NextLuecke (auffuellzr, Links(bereich))
   IF (lueckber.IsValid())
      zi := FindeEreignis (nzr, lueckber, 0.3, ~"1h", 0.05, ~"1h", 0.01)
      IF (zi.IsValid())
         quantile := ExtrahiereEreignis (nzr, zi, 20)
         IF (qf.AnzQuanten()>0)
            SynthetisiereEreignis (auffuellzr, quantile)
         ELSE
            Print ("Fehler beim Bestimmen der Quantile.")
         ENDIF
      ELSE
         Print ("Ereignis nicht bestimmbar.")
      ENDIF
   ELSE
      Print ("keine L\ucke mehr vorhanden.")
   ENDIF
END
```

4.10.9 Intervall-Zeitreihen erzeugen

- IntervallMittel
- LueckenAnteile
- Tagessummen
- SummenWerte
- IntervallMin
- IntervallMax

Beispiel

```
# Beispielprogramm zu allen Funktionen dieser Kategorie

AZUR ()
bereich := [@"15.1.75",@"25.1.75"]
zr1 := GetZR ("4311001","Niederschlag","","K")
zr1i := Tagessummen (zr1, bereich, false);
zr2 := GetZR ("4604003","Niederschlag","","K")
zr2i := Tagessummen (zr2, bereich, true);
zr3 := AddZR (zr1i, zr2i, bereich, "Muemmelmann",true)
zr4 := SummenWerte (zr3, bereich, Breite(bereich), true)
Print ("Zeitreihe 1:\n", zr1, "\nDefArt: ", DefArt (zr1))
Print ("Tagessummen der ZR 1: ", zr1i, "\nDefArt: ", DefArt (zr1i))
Print ()
Print ("Zeitreihe 2:\n", zr2, "\nDefArt: ", DefArt (zr2))
Print ("Tagessummen der ZR 2: ", zr2i, "\nDefArt: ", DefArt (zr2i))
Print ()
Print ("Summe der Tagessummen:")
Print (zr3, "\nDefArt: ", DefArt (zr3))
Print ()
Print ("Summenwerte der Summe der Tagessummen:")
Print (zr4, "\nDefArt: ", DefArt (zr4))
Print ()
Print ("Zu Zeitreihe 1:")
```

```
Print ("IntervallMittel:")
zr1mitt := IntervallMittel (zr1, bereich, ~"1 Tag", true)
Print ("\t", zr1mitt, "\n\tDefArt: ", DefArt (zr1mitt))
Print ("IntervallMin:")
zr1min := IntervallMin (zr1, bereich, ~"1 Tag", true)
Print ("\t", zr1min, "\n\tDefArt: ", DefArt (zr1min))
Print ("IntervallMax:")
zr1max := IntervallMax (zr1, bereich, ~"1 Tag", true, 10.0)
# 10.0 bedeutet, dass Intervalle mit einem Lueckenanteil
# groesser 10 Prozent zu Luecke werden
Print ("\t", zr1max, "\n\tDefArt: ", DefArt (zr1max))
Print ("LueckenAnteile:")
zr1lueck := LueckenAnteile (zr1, bereich, ~"1 Tag", true)
Print ("\t", zr1lueck, "\n\tDefArt: ", DefArt (zr1lueck))
END
```

Ausgabe:

```
Zeitreihe 1:
(4311001,Niederschlag,,M,,G,Z)
DefArt: K
Tagessummen der ZR 1: (4311001,Niederschlag,Sum,T,,A,Z)
DefArt: I

Zeitreihe 2:
(4604003,Niederschlag,,M,,G,Z)
DefArt: K
Tagessummen der ZR 2: (4604003,Niederschlag,Sum,T,,M,Z)
DefArt: I

Summe der Tagessummen:
(Muemmelmann,Niederschlag,Sum,T,,M,Z)
DefArt: I

Summenwerte der Summe der Tagessummen:
(Muemmelmann,Niederschlag,Sum,T,,M,Z)
DefArt: I
```

```
Zu Zeitreihe 1:
IntervallMittel:
(4311001,Niederschlag,Mit,T,,M,Z)
DefArt: I
IntervallMin:
(4311001,Niederschlag,Min,T,,M,Z)
DefArt: I
IntervallMax:
(4311001,Niederschlag,Max,T,,M,Z)
DefArt: I
LueckenAnteile:
(4311001,Lueckenanteil,Ant,T,,M,Z)
DefArt: I
```

4.10.10 weitere Funktionen auf ZR

- SchwellenZR
- TransZR
- LueckenReihe
- NextLuecke
- SummenLinien
- Reorganize
- PrioZR
- ImportDBF
- ScanRel
- ImportRel
- Stempeln
- SetZRBearbStand
- ZRBearbStand
- BearbStaende
- HideZR
- SummenHeberLinien
- RemoveZR
- Transform
- FuellenZR
- SchwellenMax
- ExportDBF
- ZRToRel

- DelQuality
- DelTextQuality
- Glaetten
- Terminwerte
- LogUpdate
- LogTupImp
- AkimaSpline
- SyncZR
- KalibZR
- GleitWerte
- ZRToCSV

Beispiel

```
# Beispielprogramm zu den Funktionen SummenLinien, SummenHeberLinien,
# Dauerlinie

AZUR ()
RealFormat ("10.2f")
zpmode ("#d.#m.#y #H:#M")
von := @"1.1.75"
bis := @"2.1.75"
bereich := [von, bis]
Print ("Bereich : ", bereich)
Print ()

zr := GetZR ("4311001", "Niederschlag", "", "K")
Print ("A --> OriginalZR:\n", zr)
Print ("Einheit: ", Einheit(zr))
qf := Quantenfolge (zr, bereich)
Print ()
```

```
summl := SummenLinien (zr, bereich, ~"1 Monat", true)
Print ("B --> SummenLinien:\n", summl)
Print ("Einheit: ", Einheit(summl))
qfsumml := Quantenfolge (summl, bereich)
Print ()

param := "heber"
heberl := SummenHeberLinien (zr, bereich, param, 0,10, true, ~"5 min")
Print ("C --> SummenHeberLinien:\n", heberl)
Print ("Einheit: ", Einheit(heberl))
qfheberl := Quantenfolge (heberl, bereich)
Print ()

dauerl := Dauerlinie (zr, bereich, true)
Print ("D --> Dauerlinie:\n", dauerl)
Print ("Einheit: ", Einheit(dauerl))
qfdauerl := Quantenfolge (dauerl, bereich)
Print ()

zpmode ("#d.#m.#y")
Print ("Datum: ", von)
zpmode ("#H:#M")
anzq := AnzQuanten (qf)
maxanz := 10
IF (anzq < maxanz)
maxanz := anzq
ENDIF
i := 0
Print (21*" ", "A", 11*" ", "B", 11*" ", "C", 11*" ", "D")
WHILE (i < maxanz)
qzr := YRechts(Quantnr(qf, i))
qsumml := YRechts(Quantnr(qfsumml, i))
qheberl := YRechts(Quantnr(qfheberl, i))
qdauerl := YRechts(Quantnr(qfdauerl, i))
derbereich := XBereich(Quantnr(qf, i))
Print (derbereich, " |", qzr, " |", qsumml, " |", qheberl, " |", qdauerl)
i := i+1
```

```
ENDWHILE
END
```

Ausgabe:

```
Bereich : [01.01.75 07:30,02.01.75 07:30]

A --> OriginalZR:
(4311001,Niederschlag,,M,1075052544,G,Z)
Einheit: mm/5min

B --> SummenLinien:
(4311001,Niederschlag,Sum,M,1075052544,M,Z)
Einheit: mm

C --> SummenHeberLinien:
(4311001,heber,Sum,M,1075052544,M,Z)
Einheit: mm

D --> Dauerlinie:
(4311001,Niederschlag,Dau,M,1075052544,M,Z)
Einheit: mm/5min

Datum: 01.01.75
A           B           C           D
[07:30,07:40] |      0.00 |      0.00 |      0.00 |      0.00
[07:40,07:45] |      0.11 |      0.05 |      0.11 |      0.01
[07:45,07:50] |      0.13 |      0.17 |      0.24 |      0.03
[07:50,07:55] |      0.14 |      0.31 |      0.38 |      0.04
[07:55,08:00] |      0.13 |      0.44 |      0.51 |      0.06
[08:00,08:05] |      0.14 |      0.58 |      0.65 |      0.07
[08:05,08:10] |      0.13 |      0.71 |      0.78 |      0.07
[08:10,08:15] |      0.14 |      0.85 |      0.92 |      0.08
[08:15,08:20] |      0.33 |      1.08 |      1.25 |      0.09
[08:20,08:25] |      0.62 |      1.56 |      1.87 |      0.10
```

Beispiel

```
# Beispielprogramm zu den Funktionen OrtQuery, LueckenReihe

AZUR ()
zrl := OrtQuery ("Lemming")
zpmode ("#d.#m.")
RealFormat ("5.0f")

FORALL zr IN zrl
Print (zr)
Print ()
zqf  := Quantenfolge (zr, MAXFOCUS)
lzr := LueckenReihe (zr, MAXFOCUS, 1.0, true)
lqf := Quantenfolge (lzr, MAXFOCUS)
Print ("Zeitreihe:")
FORALL zq IN zqf
Print (XBereich(zq), " ", YRechts(zq))
ENDFOR
Print ()
Print ("L\uckenreihe:")
FORALL lq IN lqf
Print (XBereich(lq), " ", YRechts(lq))
ENDFOR
Print ()
ENDFOR
END
```

Ausgabe:

```
(Lemming,Buddler,,,0,,Z)

Zeitreihe:
[01.01.,02.01.]    10
[02.01.,03.01.]    12
[03.01.,04.01.]     8
[04.01.,05.01.] Lücke
[05.01.,06.01.] Lücke
[06.01.,07.01.]     7

Lückenreihe:
[01.01.,04.01.]     1
[04.01.,06.01.] Lücke
[06.01.,07.01.]     1

(Lemming,Graber,,,0,,Z)

Zeitreihe:
[03.01.,04.01.]     8
[04.01.,05.01.]    12
[05.01.,06.01.] Lücke
[06.01.,07.01.]     7
[07.01.,08.01.]     5

Lückenreihe:
[03.01.,05.01.]     1
[05.01.,06.01.] Lücke
[06.01.,08.01.]     1

(Lemming,Baggerer,,,0,A,Z)

Zeitreihe:
[01.01.,02.01.]    10
[02.01.,03.01.]    12
[03.01.,04.01.]     8
[04.01.,05.01.]    12
[05.01.,06.01.] Lücke
[06.01.,07.01.]     7
```

```
[07.01.,08.01.]     5

Lückenreihe:
[01.01.,05.01.]     1
[05.01.,06.01.] Lücke
[06.01.,08.01.]     1
```

Beispiel

```
# Beispielprogramm zu den Funktionen Transform

AZUR ()
zpmode ("#H:#M")
RealFormat ("10.2f")
beckenzr := GetZR ("Hier","Beckenstand","","K");
Print (beckenzr)
bereich := MaxFocusZR (beckenzr)
Print ("Bereich: ", bereich)
Print ()
qf := Quantenfolge (beckenzr, bereich)
FORALL q IN qf
Print (XBereich (q), "  ", YLinks(q), " | ", YRechts(q))
ENDFOR
Print ()
kurve := GetZR ("Hier","Beckenkurve","","K");
Print (kurve)
qf := Quantenfolge (kurve, MAXFOCUS)
Print (qf)
Print ()
zr2 := Transform (beckenzr, kurve, MAXFOCUS, "Beckenf\ulle", false);
Print (zr2)
qf := Quantenfolge (zr2, MAXFOCUS)
FORALL q IN qf
Print (XBereich (q), "  ", YLinks(q), " | ", YRechts(q))
ENDFOR
END
```

Ausgabe:

```
(Hier,Beckenstand,,,0,,Z)
Bereich: [10:00,13:00]

[10:00,12:00]        0.00 |       0.00
[12:00,12:20]        0.00 |       1.00
[12:20,12:30]        1.00 |       6.00
```

```
[12:30,12:50]        6.00 |      12.00
[12:50,12:55]       12.00 |      18.00
[12:55,12:55]       18.00 |      20.00
[12:55,13:00]       20.00 |      15.00

(Hier,Beckenkurve,,,0,,R)
Linienquant 0  100   auf (0 , 10]
Linienquant 100  500   auf (10 , 20]


(Hier,Beckenfülle,,,0,A,Z)
[12:20,12:30]       10.00 |      60.00
[12:30,12:50]       60.00 |     180.00
[12:50,12:55]      180.00 |     420.00
[12:55,12:55]      420.00 |     500.00
[12:55,13:00]      500.00 |     300.00
```

Ausgabe:

```
Wasserstands-Zeitreihe:
(24005107,Wasserstand,,E,0,O,Z)
Bereich: [01.11.87,31.10.88]
Einheit: mNN

Abfluss-Zeitreihe:
(24005107,Abfluss,,E,0,A,Z)
Bereich: [01.11.87,31.10.88]
Einheit: m^3/s
```

Beispiel

```
# Beispielprogramm zu der Funktion PrioZR

AZUR ()
zpmode ("#d.#m.")
RealFormat ("5.0f")
zrl := OrtQuery ("Lemming")
FORALL zr IN zrl
```

```
Print (zr)
qf := Quantenfolge (zr, MAXFOCUS)
FORALL q IN qf
Print (XBereich(q), " ", YRechts(q))
ENDFOR
Print ()
ENDFOR
Print ()

zrneu := PrioZR (zrl, GetFocus(zrl), "Baggerer", "Lemming", false)
Print (zrneu)
qf := Quantenfolge (zrneu, MAXFOCUS)
FORALL q IN qf
Print (XBereich(q), " ", YRechts(q))
ENDFOR
END
```

Ausgabe:

```
(Lemming,Buddler,,,0,,Z)
[01.01.,02.01.]    10
[02.01.,03.01.]    12
[03.01.,04.01.]     8
[04.01.,05.01.] Lücke
[05.01.,06.01.] Lücke
[06.01.,07.01.]     7

(Lemming,Graber,,,0,,Z)
[03.01.,04.01.]     8
[04.01.,05.01.]    12
[05.01.,06.01.] Lücke
[06.01.,07.01.]     7
[07.01.,08.01.]     5


(Lemming,Baggerer,,,0,A,Z)
[01.01.,02.01.]    10
[02.01.,03.01.]    12
```

```
[03.01.,04.01.]     8
[04.01.,05.01.]    12
[05.01.,06.01.] Lücke
[06.01.,07.01.]     7
[07.01.,08.01.]     5
```

4.10.11 Attribute abfragen

- Lebenslauf
- Parameter
- Reihenart
- Einheit
- Version
- Herkunft
- Quelle
- Messgenau
- Hoehe
- Aussage
- Kommentar
- NWGrenze
- Attribute
- GueltVon
- Ort
- SubOrt
- FToleranz
- YTyp
- DefArt
- Zeitschritt
- XFaktor
- XDistanz

- XEinheit
- Koord
- GueltBis
- Publiziert
- ZRAttr
- ZRAnyTup
- ZRStr
- ZRSchreibbar
- ZRModifyInfo
- ZRMD5Sum
- ZRInfo

Beispiel

```
# Beispielprogramm zu allen Funktionen dieser Kategorie

AZUR ()
zr := GetZR ("4001", "Niederschlag", "", "K")
Print (zr)
Print (Attribute(zr))
Print ()
Print ("Parameter    :  ", Parameter(zr))
Print ("Ort          :  ", Ort(zr))
Print ("DefArt       :  ", DefArt(zr))
Print ("Aussage      :  ", Aussage(zr))
Print ("Zeitschritt  :  ", Zeitschritt(zr))
Print ("Herkunft     :  ", Herkunft(zr))
Print ("Reihenart    :  ", Reihenart(zr))
Print ("Koord        :  ", Koord(zr))
Print ("GueltVon     :  ", GueltVon(zr))
Print ("GueltBis     :  ", GueltBis(zr))
```

```
Print ("Einheit      :  ", Einheit(zr))
Print ("Messgenau    :  ", Messgenau(zr))
Print ("FToleranz    :  ", FToleranz(zr))
Print ("NWGrenze     :  ", NWGrenze(zr))
Print ("relStart     :  ", relStart(zr))
Print ("Kommentar    :  ", Kommentar(zr))
Print ("YTyp         :  ", YTyp(zr))
Print ("Hoehe        :  ", Hoehe(zr))
Print ("XEinheit     :  ", XEinheit(zr))
Print ()
Print ("Lebenslauf   : ")
Print (Lebenslauf (zr))
END
```

Ausgabe:

```
(4001,Niederschlag,Sum,M,,G,Z)


PARAMETER       "Niederschlag"
ORT             "4001"
DEFART          "K"
AUSSAGE         "Sum"
XDISTANZ        "M"
XFAKTOR             5
HERKUNFT        "G"
REIHENART       "Z"
X                     0
Y                     0
GUELTVON        ""
GUELTBIS        ""
EINHEIT         "mm/5min"
MESSGENAU         0.0000
FTOLERANZ         0.0000
NWGRENZE          0.0000
RELSTART        ""
WERTEDATEI      "4001.nk0"
KOMMENTAR       ""
```

```
HOEHE                0
YTYP            ""
XEINHEIT        ""


Parameter    :  Niederschlag
Ort          :  4001
DefArt       :  K
Aussage      :  Sum
Zeitschritt  :  5*30*d
Herkunft     :  G
Reihenart    :  Z
Koord        :  (0,2.05568e-314)
GueltVon     :  01.01.1900 00:00
GueltBis     :  01.01.1900 00:00
Einheit      :  mm/5min
Messgenau    :  0
FToleranz    :  0
NWGrenze     :  0
relStart     :  01.01.1900 00:00
Kommentar    :
YTyp         :
Hoehe        :  0
XEinheit     :

Lebenslauf   :
Importiert aus File acgemund.ntp (LWAFlut) am 15.06.1993 von ralf
```

4.10.12 Attribute setzen

- SetGueltVon
- SetFToleranz
- SetOrt
- SetSubOrt
- SetDefArt
- SetZeitschritt
- SetXDistanz
- SetXFaktor
- SetXEinheit
- SetGueltBis
- SetKoord
- SetLebenslauf
- SetParameter
- SetReihenart
- SetVersion
- SetEinheit
- SetHerkunft
- SetQuelle
- SetMessgenau
- SetHoehe
- SetAussage
- SetKommentar

- SetNWGrenze
- SetYTyp
- SetPubliziert
- ZRSetAttr
- ZRSetAnyTup
- ZRSetInfo
- DoModRecording
- DeleteModRecords

Beispiel

```
# Beispielprogramm zu den Funktionen setKoord, setGueltVon,
# setGueltBis, setMessgenau, setFToleranz, setNWGrenze,
# setKommentar, setHoehe, setXEinheit

AZUR ()
t := ReiheTupel ()
setText (t, "Ort", "Aachen")
setText (t, "Parameter", "Sonnenschein")
setText (t, "DefArt", "K")
zr := OpenZR (t)
Print (Attribute(zr))
setKoord (zr, GeoPoint (2505000, 5525000))
setGueltVon (zr, @"11.11.1977")
setGueltBis (zr, @"11.11.1988")
setMessgenau (zr, 0.1)
setFToleranz (zr, 0.01)
setNWGrenze (zr, 0.125)
setKommentar (zr, "Ein Beispiel")
setHoehe (zr, 150.0)
setXEinheit (zr, "cm")
zrneu := GetZR ("Aachen", "Sonnenschein", "", "K")
SetLebenslauf (zrneu, "Frei erfunden")
```

```
Print ()
Print (zr)
Print (Attribute(zr))
Print ("Lebenslauf:      ", Lebenslauf (zr))
END
```

Ausgabe:

```
PARAMETER       "Sonnenschein"
ORT             "Aachen"
DEFART          "K"
AUSSAGE         ""
XDISTANZ        ""
XFAKTOR             0
HERKUNFT        ""
REIHENART       ""
X               2505000
Y               5525000
GUELTVON        "19771111073000"
GUELTBIS        "19881111073000"
EINHEIT         ""
MESSGENAU         0.1000
FTOLERANZ         0.0100
NWGRENZE          0.1250
RELSTART        ""
WERTEDATEI      "aachen.sk0"
KOMMENTAR       "Ein Beispiel"
HOEHE              150
YTYP            ""
XEINHEIT        "cm"


(Aachen,Sonnenschein,,,0,,)


PARAMETER       "Sonnenschein"
ORT             "Aachen"
DEFART          "K"
```

```
AUSSAGE         ""
XDISTANZ        ""
XFAKTOR             0
HERKUNFT        ""
REIHENART       ""
X               2505000
Y               5525000
GUELTVON        "19771111073000"
GUELTBIS        "19881111073000"
EINHEIT         ""
MESSGENAU         0.1000
FTOLERANZ         0.0100
NWGRENZE          0.1250
RELSTART        ""
WERTEDATEI      "aachen.sk0"
KOMMENTAR       "Ein Beispiel"
HOEHE               150
YTYP            ""
XEINHEIT        "cm"

Lebenslauf:      Frei erfunden
```

4.10.13 Statistik

- Korrelogramm
- Min
- Max
- MinZP
- MaxZP
- Summe
- Haeufigkeiten
- Dauertabelle
- Dauerlinie
- IntDauerlinien
- Korrelation
- Varianz
- TrendBereinigung
- Trend
- Covarianz
- Korrellogramm
- Median
- Mittel
- Verteilung
- HWVerteilung
- KSTest
- STUDTest
- LnGamma

4.10.14 Analysis

- DeltaZR
- SignumZR
- AblZR
- IntZR
- LokaleMinima
- LokaleMaxima
- DistMinima
- DistMaxima

Beispiel

```
# Beispielprogramm zu den Funktionen IntZR, AblZR, DeltaZR

AZUR ()
zr := GetZR ("4311001", "Niederschlag", "", "I")
Print (zr)
RealFormat ("10.2f")
zpmode ("#d.#m.#Y")
bereich := [@"15.1.75", @"25.1.75"]
Print ("Bereich : ", bereich)
Print ()

integrzr := IntZR (zr, bereich, "niederint", true, ~"1 Tag")
text := "Integral"
laenge := Length (text)
Print (text, "\n", laenge*"-")
Print ("Einheit: ", Einheit(integrzr))
qf := Quantenfolge (integrzr, bereich)
FORALL q IN qf
Print (XBereich(q), "  ", YRechts(q))
ENDFOR
```

```
Print ()

ableitzr := AblZR (zr, bereich, "niederabl", true, ~"1 Tag")
text := "Ableitung"
laenge := Length (text)
Print (text, "\n", laenge*"-")
Print ("Einheit: ", Einheit(ableitzr))
qf := Quantenfolge (ableitzr, bereich)
FORALL q IN qf
Print (XBereich(q), "  ", YRechts(q))
ENDFOR
Print ()

dltzr := DeltaZR (zr, bereich, ~"1 Tag", "niederdelta", true)
text := "Delta"
laenge := Length (text)
Print (text, "\n", laenge*"-")
Print ("Einheit: ", Einheit(dltzr))
qf := Quantenfolge (dltzr, bereich)
FORALL q IN qf
Print (XBereich(q), "  ", YRechts(q))
ENDFOR
END
```

Ausgabe:

```
(4311001,Niederschlag,Sum,T,,A,Z)
Bereich : [15.01.1975,25.01.1975]

Integral
--------
Einheit: m*min
[15.01.1975,16.01.1975]        0.00
[16.01.1975,17.01.1975]       15.74
[17.01.1975,18.01.1975]       18.72
[18.01.1975,19.01.1975]       27.36
[19.01.1975,20.01.1975]       30.24
[20.01.1975,21.01.1975]       31.68
[21.01.1975,22.01.1975]       44.64
[22.01.1975,23.01.1975]      141.12
[23.01.1975,24.01.1975]      169.92
[24.01.1975,25.01.1975]      169.92

Ableitung
---------
Einheit: mm/d
[15.01.1975,16.01.1975]        0.00
[16.01.1975,17.01.1975]       10.93
[17.01.1975,18.01.1975]       -8.86
[18.01.1975,19.01.1975]        3.93
[19.01.1975,20.01.1975]       -4.00
[20.01.1975,21.01.1975]       -1.00
[21.01.1975,22.01.1975]        8.00
[22.01.1975,23.01.1975]       58.00
[23.01.1975,24.01.1975]      -47.00
[24.01.1975,25.01.1975]      -20.00

Delta
-----
Einheit: mm
[15.01.1975,16.01.1975]        0.00
[16.01.1975,17.01.1975]       10.93
```

```
[17.01.1975,18.01.1975]       -8.86
[18.01.1975,19.01.1975]        3.93
[19.01.1975,20.01.1975]       -4.00
[20.01.1975,21.01.1975]       -1.00
[21.01.1975,22.01.1975]        8.00
[22.01.1975,23.01.1975]       58.00
[23.01.1975,24.01.1975]      -47.00
[24.01.1975,25.01.1975]      -20.00
```

4.11 Funktionen auf ZRList

- GetFocus
- FirstZR
- AnzZR
- Query
- PrioZR
- ParamQuery
- ZRList
- OrtQuery
- Operator
- SetFocus
- ZRNr
- ZRLZip
- ZRLSetInfo
- ZRLInfo

Beispiel

```
# Beispielprogramm zu den Funktionen GetFocus, SetFocus, FirstZR, PrioZR,
# OrtQuery

AZUR ()
zrl := OrtQuery ("Lemming")
Print (zrl)
Print ()
bereich := GetFocus (zrl)
Print (bereich)
```

```
Print ()
zr1 := FirstZR (zrl)
maxf := MaxFocusZR (zr1)
l := Links (maxf)
r := Rechts (maxf)
zpmode ("#d.#m.#y")
FORALL zr IN zrl
Print (zr)
li := Links (MaxFocusZR(zr))
IF (l > li)
l := li
ENDIF
re := Rechts (MaxFocusZR(zr))
IF (r < re)
r := re
ENDIF
ENDFOR
Print ()
SetFocus (zrl, [l,r])
Print (GetFocus (zrl))
Print ()
zrneu := PrioZR (zrl, GetFocus(zrl), "Baggerer", "Lemming", true)
Print (zrneu)
qf := Quantenfolge (zrneu, MAXFOCUS)
FORALL q IN qf
Print (XBereich(q), " ", YRechts(q))
ENDFOR
END
```

Ausgabe:

```
+-Infty
+-Infty
lemming.bi0
lemming.gi0

[+-Infty,+-Infty]
```

```
(Lemming,Buddler,,,0,,Z)
(Lemming,Graber,,,0,,Z)

[01.01.90,08.01.90]

(Lemming,Baggerer,,,0,M,Z)
[01.01.90,02.01.90] 10
[02.01.90,03.01.90] 12
[03.01.90,04.01.90] 8
[04.01.90,05.01.90] 12
[05.01.90,06.01.90] Lücke
[06.01.90,07.01.90] 7
[07.01.90,08.01.90] 5
```

Beispiel

```
# Beispielprogramm zu den Funktionen ZRList, FirstZR, Query, OrtQuery,
# ParamQuery und zum Operator +

AZUR ()
zrl := ZRList ()
Print ("Eine leere ZRList :")
Print (zrl)
rt := ReiheTupel ()
setText (rt, "ORT", "1190")
setText (rt, "PARAMETER", "Temperatur")
liste1 := Query (rt)
liste2 := OrtQuery ("1119")
liste3 := ParamQuery ("Temperatur")
Print ()
Print ("Liste 1:")
FORALL zr1 IN liste1
Print (zr1)
ENDFOR
Print ()
Print ("Liste 2:")
FORALL zr2 IN liste2
Print (zr2)
```

```
ENDFOR
Print ()
Print ("Liste 3:")
FORALL zr3 IN liste3
Print (zr3)
ENDFOR
Print ()
liste4 := liste1 + FirstZR (liste2)
Print ("Liste 4 (Liste 1 + erste ZR von Liste 2):")
FORALL zr4 IN liste4
Print (zr4)
ENDFOR
END
```

Ausgabe:

```
Eine leere ZRList :
+-Infty
+-Infty

Liste 1:
(1190,TEMPERATUR,,T,,G,Z)

Liste 2:
(1119,Niederschlag,Sum,M,,G,Z)
(1119,Niederschlag,Lck,M,,A,Z)

Liste 3:
(1190,TEMPERATUR,,T,,G,Z)
(2205,TEMPERATUR,,T,,G,Z)
(2211,TEMPERATUR,,T,,G,Z)

Liste 4 (Liste 1 + erste ZR von Liste 2):
(1190,TEMPERATUR,,T,,G,Z)
(1119,Niederschlag,Sum,M,,G,Z)
```

4.12 Funktionen auf Tupel und TupelList

- Tupel
- GetText
- SetText
- SetZahl
- GetZahl
- SetDatum
- GetDatum
- SetZeit
- GetZeit
- SetBool
- GetBool
- SetZP
- GetZP
- ReiheTupel
- Attribute
- Match
- TupEqual
- IsModified
- ClearModify
- SetModify
- KeyComplete

- CopyTupel
- TupAdd
- TupPrjct
- TupJoin
- TupJoinTo
- TupelStr
- TupToCSV
- CSVToTup
- TupQFAdd
- TupRecNum
- TupClearRecNum
- SetWildcard
- IsWildcard
- TupIndex
- SetUntrusted
- IsUntrusted
- TupSelect
- TupIsSelected
- TupSetReadOnly
- TupSetStyle
- TupRestruct
- TupelList
- TupelKey

- TupCodesResolve
- Operator +
- Operator +=

4.12.1 Beispiel

```
# Beispielprogramm zu den Funktionen Tupel, setZahl, setText,
# getZahl, getText

AZUR ()
   s := "Name#10s,Strasse#20s,Hausnr#3.0n"
   t := Tupel (s)
   setText (t, "Name", "M\uller")
   setText (t, "Strasse", "Hauptstr.")
   setZahl (t, "Hausnr", 111)
   Print (t)
   name := getText(t, "Name")
   strasse := getText(t, "Strasse")
   hausnr := getZahl(t, "Hausnr")
   Print (name, ", ", strasse, hausnr)
END
```

Ausgabe:

```
NAME            "Müller"
STRASSE         "Hauptstr."
HAUSNR          111

Müller, Hauptstr.111
```

4.12.2 Beispiel

```
# Beispielprogramm zu den Funktionen ReiheTupel, Attribute, setText

AZUR ()
```

```
   rt := ReiheTupel()
   setText (rt, "PARAMETER", "Temperatur")
   Print (rt)
   liste := Query (rt)
   zr1 := FirstZR (liste)
   Print ("Attribute der ersten Zeitreihe in der Liste : ", Attribute (zr1))
   Print ("Maxima der Zeitreihen:")
   FORALL zr IN liste
      einh := Einheit (zr)
      maxwert := Max (zr, MAXFOCUS)
      Print (maxwert, einh)
   ENDFOR
END
```

Ausgabe:

```
PARAMETER       "Temperatur"
ORT             ""
DEFART          ""
AUSSAGE         ""
XDISTANZ        ""
XFAKTOR             0
HERKUNFT        ""
REIHENART       ""
X                     0
Y                     0
GUELTVON        ""
GUELTBIS        ""
EINHEIT         ""
MESSGENAU         0.0000
FTOLERANZ         0.0000
NWGRENZE          0.0000
RELSTART        ""
WERTEDATEI      ""
KOMMENTAR       ""
HOEHE                 0
YTYP            ""
XEINHEIT        ""
```

```
Attribute der ersten Zeitreihe in der Liste :

PARAMETER       "TEMPERATUR"
ORT             "1190"
DEFART          "I"
AUSSAGE         ""
XDISTANZ        "T"
XFAKTOR             1
HERKUNFT        "G"
REIHENART       "Z"
X                   658
Y                   154
GUELTVON        ""
GUELTBIS        ""
EINHEIT         "0C"
MESSGENAU         0.0000
FTOLERANZ         0.0000
NWGRENZE          0.0000
RELSTART        "19841101000000"
WERTEDATEI      "1190.ti0"
KOMMENTAR       ""
HOEHE                0
YTYP            ""
XEINHEIT        ""


Maxima der Zeitreihen:
28.4°C
28.6°C
27°C
```

4.12.3 Beispiel

```
# Beispielprogramm zu den Funktionen Tupel, getText, setText, AppTupel

AZUR ()
   R := Relation ("dierelation")
```

```
   Neu := NewRelation ("neuerelation", "Name#20s")
   tneu := Tupel (Struktur(Neu));
   Print ("Struktur des neuen Tupels : ", tneu)
   anz := AnzTupel(R)
   Print ("Anzahl der Tupel : ", anz)
   Print ()
   FORALL tp IN R
      text := getText(tp, "Name")
      setText (tneu, "Name", text)
      AppTupel (Neu, tneu)
      Print (getText(tneu, "Name"))
   ENDWHILE
END
```

Ausgabe:

```
Struktur des neuen Tupels :

NAME            ""

Anzahl der Tupel : 21

Aachen0
Aachen1
Aachen1
Aachen2
Aachen2
Aachen2
Aachen3
Aachen3
Aachen4
Aachen5
Aachen6
Aachen6
Aachen6
Aachen7
Aachen8
Aachen9
```

Aachen10
Aachen11
Aachen12
Aachen12
Aachen13

4.13 Funktionen auf Relation

- Name
- OpenRel
- Relation
- OraRelation
- RelSchreibbar
- MySQLRelation
- NewRelation
- NewMemRelation
- NewOraRelation
- NewMySQLRelation
- AppTupel
- DelTupel
- DelAllTupels
- FirstTupel
- LastTupel
- NextTupel
- PrevTupel
- DBFilter
- DBInvFilter
- CreateSortIndex
- CreateIndex

- AnzTupel
- Struktur
- Rewrite
- Search
- SearchFirst
- SearchAll
- KeyUnique
- AppField
- RelPrjct
- DBFlush
- LockTupel
- UnlockTupel
- SearchNum
- OraSQLDirekt
- RelChanged
- WriteDBF
- ReadPalmDB
- WritePalmDB
- SelectAll
- CollectAll
- DelSelected
- RelDateMatch
- RelJoin

- RelUnion
- RelSchnitt
- RelKey
- RelRewrite
- RelModified
- RelClearModify
- WerteMenge

4.13.1 Beispiel

```
# Beispielprogramm zu den Funktionen NewRelation, Struktur und AnzTupel

AZUR ()
R := NewRelation ("B\ucher", "Titel#50s,Autor#20s,Bestnr#10.0n")
t := Tupel (Struktur(R));
Print ("Struktur der neuen Relation : ", t)

setText (t, "Titel", "The Hitchhiker's Guide To The Galaxy")
setText (t, "Autor", "D. Adams")
setZahl (t, "Bestnr", 111)
AppTupel (R, t)

setText (t, "Titel", "The Restaurant At The End Of The Universe")
setText (t, "Autor", "D. Adams")
setZahl (t, "Bestnr", 222)
AppTupel (R, t)

setText (t, "Titel", "Life, The Universe and Everything")
setText (t, "Autor", "D. Adams")
setZahl (t, "Bestnr", 333)
AppTupel (R, t)

setText (t, "Titel", "So Long, And Thanks For All The Fish")
```

```
setText (t, "Autor", "D. Adams")
setZahl (t, "Bestnr", 444)
AppTupel (R, t)

anz := AnzTupel (R)
Print ("Anzahl der Tupel : ", anz)
END
```

Ausgabe:

```
Struktur der neuen Relation :

TITEL           ""
AUTOR           ""
BESTNR                  0

Anzahl der Tupel : 4
```

4.13.2 Beispiel

```
# Beispielprogramm zu den Funktionen Relation, NewRelation, AppTupel

AZUR ()
R := Relation ("B\ucher")
Neu := NewRelation ("B\uchertitel", "Titel#50s")
tneu := Tupel (Struktur(Neu));
Print ("Struktur des neuen Tupels : ", tneu)
anz := AnzTupel(R)
Print ("Anzahl der Tupel : ", anz)
Print ()
i := 0
FORALL tp IN R
text := getText (tp, "Titel")
setText (tneu, "Titel", text)
AppTupel (Neu, tneu)
Print (getText (tneu, "Titel"))
ENDWHILE
```

```
END
```

Ausgabe:

```
Struktur des neuen Tupels :

TITEL           ""

Anzahl der Tupel : 4

The Hitchhiker's Guide To The Galaxy
The Restaurant At The End Of The Universe
Life, The Universe and Everything
So Long, And Thanks For All The Fish
```

4.14 Funktionen auf Datenbank

- Name
- NewDatenbank
- DBRead
- DBWrite
- DBInfo
- DBAddRel
- DBGetRel
- DBDelRel
- XLSToDB
- FORALL

4.14.1 Beispiel

```
# Alle dbf-Relationen in die Datenbank alles.rdb schreiben

AZUR ()
#
fl := FileList ("*.dbf")
db := DBCreate ("alles")
FORALL s IN fl
s2 := s - ".dbf"
R := s2.Relation()
db.AddRel (R)
ENDFOR
END
```

4.14.2 Beispiel

```
# Alle Relationen der Datenbank alles.rdb als dbf-Relation extrahieren

AZUR ()
db := DBOpen ("alles")
FORALL R IN db
R.WriteDBF (R.Name())
ENDFOR
END
```

4.15 Geometrie

4.15.1 Funktionen auf GeoPoint

- XKoo
- GKTrans
- YKoo
- ZKoo
- Abstand
- GeoPoint
- GeoDist
- Koord
- GKToGrad
- GradToGK
- UTMToGrad
- GradToUTM
- LambertToGrad
- GradToLambert

Beispiel

```
# Beispielprogramm zu allen Funktionen dieser Kategorie

AZUR ()
zr1 := FirstZR (ParamQuery ("Temperatur"))
Print (zr1)
point1 := Koord (zr1)
Print ("Position1 : ", point1)
```

```
Print ("            X: ", XKoo(point1), "  Y: ", YKoo(point1))
zr2 := FirstZR (ParamQuery ("Niederschlag"))
Print (zr2)
point2 := Koord (zr2)
Print ("Position2 : ", point2)
Print ("            X: ", XKoo(point2), "  Y: ", YKoo(point2))
abst := Abstand (point1, point2)
Print ("Abstand : ", abst)
END
```

Ausgabe:

```
(1190,TEMPERATUR,,T,,G,Z)
Position1 : (658,154.0001)
X: 658  Y: 154
(99999,Niederschlag,Lck,,0,A,Z)
Position2 : (0,2.055679e-314)
X: 0  Y: 0
Abstand : 675.781
```

4.15.2 Funktionen auf Polygon

- Name
- PolyLen
- PolySelect
- PolyCenter
- PolyEntkolin
- Polygon
- PolyDist
- PolySetAttr
- SetName
- PolyArea
- PolyAttr
- PolyGlue
- PolyAdjust
- PolySplit
- PolySection
- PolyProjection
- PolySchnitt
- PolyInside

4.15.3 Funktionen auf Layer

- Name
- LayerSetAttr
- WriteLayer
- LayerAttr
- ReadLayer
- Isolinien
- Isoflaechen
- Isotachen
- LayerNetz
- Layer
- SetName
- LayerAnzPoly

4.15.4 Funktionen auf Karte

- KarteSetVoll
- Name
- WriteKarte
- FindPoly
- Selection
- KarteBereichRO
- KarteBereichLU
- SetName
- KarteOnPage
- KarteAttr
- StrToKarte
- ReadKarte
- KarteSetBereich
- PlotKarte
- Karte
- KarteSelect
- KarteSetAttr
- SelectPolys
- SetGeoParam
- KarteBlatt
- GeoCanvasOnPage

4.15.5 Beispiel: Berechnung von Isolinien

```
# Wird aus einem Aquagramm aufgerufen
MakeIsos (ZP von, ZP bis, Real delta=1, String map)

   SetDatenPool ("../konti")

   bereich := [von,bis]
   L := Layer("isos")

   i := 5
   name := token(map,i)
   WHILE (name#"")
      zr := GetZR (name, "Niederschlag", "", "K")
      IF (IsValid(zr))
         z := Summe (zr, bereich)
         IF (z#Luecke)
            pk := Koord(zr)
            IF (XKoo(pk)>0)
               p := {XKoo(pk), YKoo(pk), z}
               poly := Polygon (Ort(zr), p)
               L := L + poly
            ENDIF
         ENDIF
      ENDIF
      i := i+1
      name := token(map,i)
   ENDWHILE

   IL := Isolinien (L, 0, 1000, delta)
   WriteLayer (IL, "isos.ai", "AI")
END
```

4.16 Funktionen auf Page und Report

- NewPage
- NewBigPage
- LinesOnPage
- TextOnPage
- TextCenterOnPage
- DrawTextOnPage
- PlotTextOnPage
- DrawLineOnPage
- DrawSymOnPage
- AxBoxOnPage
- KarteOnPage
- PrintPage
- ClearPage
- SetPageColor
- SetPageGrauton
- SetPageFont
- SetPageLineStyle
- PagePos
- PageMaxPos
- TextWidth
- CanvasOnPage

- GeoCanvasOnPage
- DrawBoxOnPage
- BitmapOnPage
- NewReport
- PageOnReport
- PrintReport

Folgende Konstanten sind definiert:

- DINA0
- DINA1
- DINA2
- DINA3
- DINA4
- DINA5
- BESTFIT
- PORTRAIT
- LANDSCAPE
- NORMAL
- BOLD
- UNDERLINE
- ITALIC
- BIG
- SMALL

Weitere Konstanten sind in TextOnPage() beschrieben.

4.16.1 Beispiel

```
# Beispielprogramm zu den Funktionen NewPage, LinesOnPage, TextOnPage,
# TextCenterOnPage, DrawOnPage, PrintPage, ClearPage und den
# Konstanten DINA4, LANDSCAPE, NORMAL, BOLD, UNDERLINE, BIG

AZUR ()
max_spalte := 112
page := NewPage (max_spalte, DINA4, LANDSCAPE)
lines := LinesOnPage (page)
Print ("Anzahl der Zeilen : ", lines)

TextCenterOnPage (page, 4, "Tabelle", BOLD+UNDERLINE+BIG)
breit := max_spalte - 2
hoch  := lines - 2

# wir nehmen an, die Koordinaten sind in Zeile,Spalte
# dann muessen sie umgerechnet werden mit PagePos
DrawOnPage (page, PagePos({8,15}), PagePos({breit,15}), 0.02)
i := 15
zeilennr := 1
WHILE (i < hoch)
DrawOnPage (page, PagePos({8,i}), PagePos({breit,i}), 0.01)
TextOnPage (page, 5,i+1, Str (zeilennr), NORMAL)
zeilennr := zeilennr+1
i := i+3
ENDWHILE
DrawOnPage (page, PagePos({10,14}), PagePos({10,hoch}), 0.02)
j := 10
spaltennr := 1
WHILE (j < breit)
DrawOnPage (page, PagePos({j,14}), PagePos({j,hoch}), 0.01)
TextOnPage (page, j+5,12, Str (spaltennr), NORMAL)
spaltennr := spaltennr+1
j := j+10
ENDWHILE

PrintPage (page, "thepage.ps", "PS")
```

```
ClearPage (page)
PrintPage (page, "emptypage.ps", "PS")
END
```

4.17 Funktionen auf Quant und QuantList

- AppendQuant
- PrependQuant
- DelQuant
- YRechts
- NewQuantenfolge
- AnzQuanten
- SetYLinks
- SetXBereich
- TextIQuant
- LinienQuant
- MomentanQuant
- QuantNr
- Quantenfolge
- Qualitaetsfolge
- IntervallQuant
- XBereich
- YLinks
- SetQuantText
- TextQuant
- YText
- SetYRechts

- TextQuantenfolge
- TupQFAdd
- WriteQuantenfolge
- WriteTextQuantenfolge
- StoreQF
- StoreTextQF
- AppendQF
- SortQuantenfolge
- Rastern

4.17.1 Beispiel

```
# Beispielprogramm zu allen Funktionen dieser Kategorie

AZUR ()
ql := LinienQuant ([@"11.11.1988",@"11.11.1989"], 11, 22)
Print ("Der Linienquant : ", Str(ql))
Print ("Der Bereich     : ", XBereich(ql))
Print ("Die Summe der Randwerte : ", YRechts(ql)+YLinks(ql))
Print ()

qi := IntervallQuant ([@"11.11.1988",@"11.11.1989"], 1111)
Print ("Der Intervallquant : ", Str(qi))
Print ("Der Bereich        : ", XBereich(qi))
Print ()

zr1 := GetZR ("2211", "TEMPERATUR", "", "I")
qf  := Quantenfolge (zr1, [@"1.8.1988",@"8.8.1988"])
Print (qf)
einh := Einheit (zr1);
FORALL q IN qf
Print (Rechts(XBereich(q)), " ", YRechts(q), einh)
```

```
ENDFOR
Print ()
anz := AnzQuanten (qf)
num := 0
WHILE (num < anz)
q := QuantNr (qf, num)
Print (Rechts(XBereich(q)), " ", YRechts(q), einh)
num := num + 1
ENDWHILE
Print ()

zr2 := GetZR ("90908", "Niederschlag", "", "K")
Print (zr2)
bereich := MaxFocusZR (zr2)
Print (bereich)
maxbis := Rechts (bereich) + ~"3 Monate"
von := Links (bereich)
bis := von + ~"3 Monate"
qual := 1
WHILE (bis < maxbis)
stbereich := [von, bis]
Stempeln (zr2, stbereich, qual)
von  := bis
bis  := von + ~"3 Monate"
qual := qual + 1
ENDWHILE
qualifolge := Qualitaetsfolge (zr2, bereich)
anzql := AnzQuanten (qualifolge)
Print ("Anzahl der Qualit\atsquanten : ", anzql)
IF (anzql > 0)
FORALL ql IN qualifolge
Print (XBereich(ql), " ", YRechts(ql))
ENDFOR
ENDIF
END
```

Ausgabe:

```
Der Linienquant : (11.11.1988 07:30,11.11.1989 07:30]: 11 22
Der Bereich     : [11.11.1988 07:30,11.11.1989 07:30]
Die Summe der Randwerte : 33

Der Intervallquant : (11.11.1988 07:30,11.11.1989 07:30]: 1111 1111
Der Bereich        : [11.11.1988 07:30,11.11.1989 07:30]

IntervallQuant 12.8 auf (01.08.1988 07:30 , 02.08.1988 00:00]
IntervallQuant 11.4 auf (02.08.1988 00:00 , 03.08.1988 00:00]
IntervallQuant 14.9 auf (03.08.1988 00:00 , 04.08.1988 00:00]
IntervallQuant 14.4 auf (04.08.1988 00:00 , 05.08.1988 00:00]
IntervallQuant 18.5 auf (05.08.1988 00:00 , 06.08.1988 00:00]
IntervallQuant 20.6 auf (06.08.1988 00:00 , 07.08.1988 00:00]
IntervallQuant 19.6 auf (07.08.1988 00:00 , 08.08.1988 00:00]
IntervallQuant 18.1 auf (08.08.1988 00:00 , 08.08.1988 07:30]

02.08.1988 00:00 12.8°C
03.08.1988 00:00 11.4°C
04.08.1988 00:00 14.9°C
05.08.1988 00:00 14.4°C
06.08.1988 00:00 18.5°C
07.08.1988 00:00 20.6°C
08.08.1988 00:00 19.6°C
08.08.1988 07:30 18.1°C

02.08.1988 00:00 12.8°C
03.08.1988 00:00 11.4°C
04.08.1988 00:00 14.9°C
05.08.1988 00:00 14.4°C
06.08.1988 00:00 18.5°C
07.08.1988 00:00 20.6°C
08.08.1988 00:00 19.6°C
08.08.1988 07:30 18.1°C

(90908,Niederschlag,Abl,,0,0,Z)
[01.11.1991 08:30,01.11.1993 07:00]
Anzahl der Qualitätsquanten : 8
[01.11.1991 08:30,01.02.1992 08:30] 1
[01.02.1992 08:30,01.05.1992 08:30] 2
```

```
[01.05.1992 08:30,01.08.1992 08:30] 3
[01.08.1992 08:30,01.11.1992 08:30] 4
[01.11.1992 08:30,01.02.1993 08:30] 5
[01.02.1993 08:30,01.05.1993 08:30] 6
[01.05.1993 08:30,01.08.1993 08:30] 7
[01.08.1993 08:30,01.11.1993 07:00] 1
```

4.18 Arbeiten mit Arrays

- ArraySize
- SetCurrent
- StrToArray
- +-Operator
- StrSplit
- Array

4.18.1 Beispiel

```
# Beispielprogramm zu allen Funktionen dieser Kategorie

AZUR ()
a := Array();
a[1] := 10
a["Haus"] := "Dach"
a["Bau"] := "Bett"
# Elemente werden sortiert ausgegeben
FORALL s IN a
print (s)
ENDFOR
print (ArraySize(a))
b := StrToArray ("Rot Blau Gelb Schwarz")
print (b);
END
```

Ausgabe:

10 Bett Dach 3 Blau Gelb Rot Schwarz

4.19 Funktionen zum Netzwerkbetrieb (z.B. AquaWeb)

Diese Funktionen sind für den Betrieb im Netzwerk gedacht.

- isUnixClient
- isAQTP
- SendAQTP
- RecvAQTP
- ClientExec
- ClientPathList
- FTPGet
- FTPMultiGet
- FTPPut
- FTPDir
- FTPDel
- FTPRename
- URLParams
- MakeURLParams
- WebGet
- ImportQF

- WebPut
- ExportQF
- TCPOpen
- TCPClose
- TCPGet
- TCPPut
- UniversQuery
- SendMail
- CorbaOpen
- CorbaClose

4.19.1 Beispiel

```
# Beispielprogramm zu den Funktionen

SendeWas ()
IF (IsUnixClient())
SendAQTP ("datei.txt.bz2")
ELSE
SendAQTP ("datei.zip")
ENDIF
END

EmpfangeWas (String datei)
datei2 := RecvAQTP (datei)
s := datei2.ReadLine()
print (s)
END
```

4.20 Funktionen zur Kommunikation mit seriellen Schnittstellen

Diese Funktionen sind für die Kommunikation mit seriellen Schnittstellen gedacht.

- SerOpen
- SerClose
- SerReady
- SerGet
- SerWrite
- SerRead
- SerBinRead
- IEC870Recv
- IEC870Send

4.20.1 Beispiel

```
# Beispielprogramm zu den Funktionen

Folgt ...
```

Kapitel 5

Vordefinierte Funktionen alphabetisch

AblZR

Syntax: AblZR(ZR z, Intervall i, String s, Bool b) : ZR
z: Ausgangszeitreihe
i: Berechnungszeitraum
s: Neuer Parameter der Ergebniszeitreihe
b: [?]

Beispiel:
```
abl := AblZR (zr, zr.MaxFocusZR(), "Niederschlag", False)
```

Beschreibung: Differenziert die Zeitreihe z über dem Intervall i. Die Aussage wird auf Abl gesetzt. Zum Algorithmus siehe [?]. Die Berechnung der Ergebniseinheit findet automatisch statt.

Abs

Syntax: Abs (Real r) : Real
r:

Beispiel:
```
betrag := Abs (wert)
```

Beschreibung: Liefert den Absolutbetrag einer Zahl.

Abstand

Syntax: Abstand (GeoPoint g1, GeoPoint g2) : Real
g1:
g2:

Beispiel:
```
abst := Abstand (g1, g2)
```

Beschreibung: Berechnet den Abstand zwischen den GeoPoints $g1$ und $g2$. Das Ergebnis ist eine Realzahl.

ADBBenutzerRel

Syntax: ADBBenutzerRel () : Relation

Beispiel: `rel := ADBBenutzerRel ()`

Beschreibung: Liefert die Benutzer-Relation zurück.

Siehe auch ADBInit().

ADBBezeichnung

Syntax: ADBBezeichnung (String was) : String
was: spezifiziert die Messstelle/Station

Beispiel: `derort := ADBBezeichnung ("207654")`

Beschreibung: Sucht das Stammdaten-Tupel, das auf *was* passt (siehe Stammdaten()), und liefert den Inhalt des Feldes daraus zurück, das die Rolle Bezeichnung hat.

Wenn keine Stammdaten gefunden wurden oder der ADB-Manager nicht initialisiert wurde, wird *was* zurückgeliefert.

Siehe auch ADBInit(), ADBZROrt() und ADBName().

ADBChangeFields

Syntax: ADBChangeFields (S zrort, S name, S bezeich, S suchfelder)
zrort: Name des Feldes, das den Ort der ZR enthält
name: Name des Feldes, das den Namen enthält
bezeich: Name des Feldes, das die Bzeichnung enthält
suchfelder: Liste mit weiteren Suchfeldern

Beispiel: `ADBChangeFields ("DBMSNR","HDNR","MSTNAM","")`

Beschreibung: Legt neue Felder für die jeweiligen Rollen fest. Dies ist z.B. notwendig, wenn sich anhand des Benutzers das Feld bestimmt, das die Rolle Name spielen soll (im Beispiel `HDNR`).

ADBCreateRel

Syntax: ADBCreateRel(String relname, String aufbau) : Relation
relname: Name der zu öffnenden Relation
aufbau: Struktur der Relation

Beispiel: `grel := ADBCreateRel("a_grenzen", strukt)`

Beschreibung: Erzeugt eine neue Relation *relname* mit der Struktur $aufbau$, tut sie in den Cache und liefert sie zurück.

Wenn schon eine Relation namens *relname* vorhanden ist, wird $aufbau$ ignoriert. Die Funktion arbeitet dann wie ADBOpenRel().

Misslingt das Anlegen, wird eine ungültige Relation geliefert.

DBMS und Zugriffspfade werden automatisch anhand der an ADBInit übergebenen Stammdaten ergänzt.

Siehe auch ADBInit() und ADBOpenRel().

ADBDeCache

Syntax: ADBDeCache (String relname)
relname: Name der zu entfernenden Relation

Beispiel: `ADBDeCache("zeit_geber")`

Beschreibung: Die Relation *relname* wird **aus dem Cache** des ADBManagers gelöscht. Ein erneutes Öffnen mittels ADBOpenRel() nähme sie wieder auf.

Siehe auch ADBInit().

ADBEmptyCache

Syntax: ADBEmptyCache ()

Beispiel: `ADBEmptyCache()`

Beschreibung: Der Cache des ADBManagers wird geleert.

Siehe auch ADBInit(), ADBOpenRel() und ADBDeCache().

ADBExists

Syntax: ADBExists(String relname) : Bool
relname: Name der Relation

Beispiel:
```
IF (ADBExists("zeit_geber"))
```

Beschreibung: Überprüft, ob die Relation *relname* vorhanden ist.

Siehe auch ADBOpenRel().

ADBFlush

Syntax: ADBFlush ()

Beispiel:
```
ADBFlush ()
```

Beschreibung: Prüft, ob sich Tupel der Stammdaten- oder der Benutzer-Relation geändert haben oder gelöscht wurden und schreibt diese zurück in die jeweilige Datei oder löscht sie da.

Diese Änderung auf der Platte wird nicht automatisch von anderen Prozessen erkannt. Da die Kommunikation nur über das Dateisystem läuft, besteht auch nicht die Möglichkeit, den anderen Prozessen ein ADBFlush mitzuteilen. Stattdessen müssen diese an passender Stelle ADBUpdate() aufrufen.

Siehe auch ADBInit().

ADBFlushCache

Syntax: ADBFlushCache ()

Beispiel:
```
ADBFlushCache()
```

Beschreibung: Änderungen der Relationen im Cache werden permanent in die Datenbank übernommen. Die Relationen im Cache werden dabei nicht verändert.

Siehe auch ADBInit(), ADBOpenRel() und ADBUpdateCache().

ADBInit

Syntax: ADBInit (S stamm, S key, S benutzer, S feldzrort, S feldname, S feldbezeich, S suchfelder)
stamm: Name der Stammdatenrelation (bei dBase inkl. .dbf)
key: Key der Stammdaten-Relation
benutzer: Name der Benutzerrelation (bei dBase inkl. .dbf)
feldzrort: Name des Feldes, das den Ort der ZR enthält
feldname: Name des Feldes, das den Namen enthält
feldbezeich: Name des Feldes, das die Bezeichnung enthält
suchfelder: Liste mit weiteren Suchfeldern

Beispiel: `ADBInit ("stamnied.dbf", "ORT", "benutzer.dbf", "ORT","ORT","NAME","BESON`

Beschreibung: Initialisiert den ADB-Manager (Aquaplan-Database-Manager). Dieser verwaltet die Stammdaten (von Stationen, Messstellen) und die Benutzerrechte. Diese Aufgaben werden dem Azurprogramm somit abgenommen. Das Azurprogramm muss über die genaue Struktur der Stammdaten an fast keiner Stelle informiert sein. Es fragt die entscheidenden Felder anhand ihrer Rolle (ZROrt, Name und Bezeichnung) ab, und es holt die Stammdaten-Tupel anhand bei ADBInit übergebener Schlüsselfelder.

key ist **der** Key der relationalen Struktur. Dieser Key kann nicht verändert werden.

$suchfelder$ kann leer sein oder einen oder mehrere Suchfelder enthalten. Die einzelnen Suchfelder sind mit | getrennt. Diese Felder können Multi-Suchfelder sein (also eine mit + verbundene Liste von Feldern enthalten). $feldzrort$, $feldname$ und $feldbezeich$ hingegen müssen einfache Feldnamen sein.

Das Init darf nur einmal ausgeführt werden. Wenn sich die Stammdaten- oder die Benutzer-Datei verändert haben, müssen sie mittels ADBUpdate() aktualisiert werden. Sollen andere Suchfelder benutzt werden, kann dies mit ADBChangeFields() erreicht werden. Nach dem Init ist kein Benutzer gesetzt, das heißt, der Aufruf von Benutzer() liefert ein ungültiges Tupel. Ein Benutzer wird mit UpdateBenutzer() gesetzt.

Die Funktionen Stammdaten(), ADBZROrt(), ADBName() und ADBBezeich() suchen in allen Suchfeldern in der beim Init angegebenen Reihenfolge nach dem übergebenen Suchstring. Das heißt, erst wird der Suchstring im Feld *key* gesucht, dann im Feld $feldzrort$, in $feldname$, in $feldbezeich$ und dann in allen Feldern aus $suchfelder$.

ADBInit unterstützt verschiedenen Datenbank-Manager-Systeme (DBMS). Das DBMS wird anhand der Angaben des Parameters *stamm* festgelegt. Alle weiteren vom ADBManager verwalteten Relationen (siehe ADBOpenRel()) benutzen die selben Angaben zum DBMS. Möglich sind:

- DBF-Dateien: *stamm* endet mit .dbf. Ist ein Pfad angegeben, (z.B: stammdir/kerndaten.dbf), werden alle Relationen in diesem Verzeichnis gesucht.
- Oracle: *stamm* enthält drei Tokens. Das erste Token ist der Name der Kernrelation, das zweite der Instanzname und das dritte user/password (mit / getrennt)
- MySQL: *stamm* enthält vier Tokens. Das erste Token ist der Name der Kernrelation, das zweite der Name der Datenbank, das dritte der Hostname und das vierte user/password (mit / getrennt)
- UniverSQL: *stamm* beginnt mit dem String UVS:. Dann folgen Name der Kernrelation, DSN und URL.

ADBKeyfeld

Syntax: ADBKeyfeld ()

Beispiel: `keyfeld := ADBKeyfeld ()`

Beschreibung: Liefert den Feldnamen des Keys der Stammdaten.

Siehe auch ADBInit().

ADBListe

Syntax: ADBListe (Tupel muster) : Relation
muster: Mustertupel

Beispiel: `rel := ADBListe (tup)`

Beschreibung: Filtert alle Tupel aus den Stammdaten, die auf *muster* passen. Falls alle Tupel umfasst werden sollen, übergibt man `Tupel(ADBStammRel())`. Diese gefilterten Tupel werden dann projiziert auf eine Mem-Relation, die zwei Felder enthalten. Das erste Feld heißt KEY und hat die gleiche Struktur wie das Feld, das bei ADBInit als Key angemeldet wurde. Das zweite Feld heißt Name und hat die Struktur des Name-Felds.

Siehe auch ADBInit() NewDBListe().

ADBLockTupel

Syntax: ADBLockTupel(Tupel tup) : Bool
tup: ein Tupel aus den Stammdaten

Beispiel: `ok := ADBLockTupel(tup)`

Beschreibung: Das Tupel *tup* wird in den permanent gespeicherten Stammdaten (also z.B. in der dbf-Datei) gelockt (siehe LockTupel()).

Wenn das Tupel schon gelockt war, wird `False` zurückgeliefert, sonst `True`.

Siehe auch ADBInit() und ADBUnlockTupel().

ADBName

Syntax: ADBName (String was) : String
was: spezifiziert die Messstelle/Station

Beispiel: `derort := ADBName ("207654")`

Beschreibung: Sucht das Stammdaten-Tupel, das auf *was* passt (siehe Stammdaten()), und liefert den Inhalt des Felds daraus zurück, das die Rolle Name hat.

Wenn keine Stammdaten gefunden wurden oder der ADB-Manager nicht initialisiert wurde, wird *was* zurückgeliefert.

Siehe auch ADBInit(), ADBZROrt(), ADBBezeichnung() und ADBNameFeld().

ADBNameFeld

Syntax: ADBNameFeld () : String

Beispiel: `dispfeld := ADBNameFeld()`

Beschreibung: Liefert den Feldnamen des Feldes, das die Rolle *Name* spielt.

Wenn keine Stammdaten gefunden wurden oder der ADB-Manager nicht initialisiert wurde, wird ein Leerstring zurückgeliefert.

Siehe auch ADBInit(), und ADBName().

ADBOpenRel

Syntax: ADBOpenRel(String relname) : Relation
relname: Name der zu öffnenden Relation

Beispiel: `grel := ADBOpenRel("zeit_geber")`

Beschreibung: Öffnet die Relation *relname* und liefert sie zurück. Über ADBOpenRel geöffnete Relationen werden vom ADBManager in einem Kamellesäckche. (Cache) gespeichert.

Wenn keine Relation namens *relname* vorhanden ist, wird eine ungültige Relation zurückgeliefert.

DBMS und Zugriffspfade werden automatisch anhand der an ADBInit übergebenen Stammdaten ergänzt.

Es ist zu beachten, dass Änderungen, die in der Folge an der Relation vorgenommen werden, nur auf die Kopie im Cache wirken. Um die Änderungen permanent zu machen, muss man die Relation aus dem Cache in die Datenbank speichern. Das geschieht (für alle Relationen im Cache) mit ADBFlushCache().

Siehe auch ADBInit(), ADBCreateRel() und ADBDeCache().

ADBQuery

Syntax: ADBQuery (String muster) : Relation
muster: Suchmuster

Beispiel:
```
fundrel := ADBQuery ("Frankenh*")
```

Beschreibung: Sucht alle Stammdateneinträge, die auf *muster* passen, und liefert diese in einer Relation zurück.

Auf welches Feld der Stammdaten *muster* passt, spielt keine Rolle. Es wird das Keyfeld (siehe ADBKeyfeld()) und alle Such-Felder durchsucht, die bei ADBInit() bzw. ADBChangeFields() übergeben wurden.

In *muster* dürfen Wildcards verwendet werden, das heißt, Teile des Namens dürfen durch ein `*` abgekürzt, oder einzelne Buchstaben durch den Platzhalter `?` ersetzt werden. Siehe dazu WildcardMatch().

Wenn man nicht nach einem Muster suchen möchte, sondern eine genaue Spezifikation des Stammdateneintrags übergeben will, benutzt man die Funktion Stammdaten().

ADBResolveField

Syntax: ADBResolveField (S feld, Bool on)
feld: Name der Komponente
on: Codeauflösung ein- oder ausschalten

Beispiel:
```
ADBResolveField ("BEOBCODE", False)
```

Beschreibung: Standardmäßig werden beim Darstellen eines Tupels die mittels ADBSetResolveCode() festgelegten Codes in Feldern in Klartext gewandelt. Mithilfe von ADBResolveField kann man dynamisch ein- und abschalten, ob Codes gewandelt werden sollen.

Siehe auch GetText() und TupCodesResolve().

ADBSetResolveInfo

Syntax: ADBSetResolveInfo (S feld, S coderel, S keyfeld, S dispfeld)
feld: Name der Komponente
coderel: Name der Code-Relation
keyfeld: Feldname des Keys in der Code-Relation
dispfeld: Feldname der Code-Relation, der angezeigt wird

Beispiel: `ADBSetResolveInfo(rel,"BEOBCODE", "code_benutzer", "BEOBCODE", "NAME")`

Beschreibung: Legt fest, dass das Feld $feld$ in Relationen, die in der Folge mit ABDOpenRel() geöffnet werden, ein Code-Feld ist. Code-Feld bedeutet, dass der Inhalt des Felds ein Schlüssel ist, der über eine weitere Relation (*coderel*) über das dortige Feld $keyfeld$ in den Klartext ($dispfeld$) umgesetzt werden kann.

$dispfeld$ kann auch mehrere Feldnamen enthalten, die mit + verbunden angegeben werden (z.B. `NAME+TELNR`).

Siehe auch ADBResolveField(), GetText() und TupCodesResolve().

ADBStammRel

Syntax: ADBStammRel () : Relation

Beispiel: `rel := ADBStammRel ()`

Beschreibung: Liefert die Stammdaten-Relation zurück.

Siehe auch ADBInit().

ADBTupZROrt

Syntax: ADBTupZROrt (Tupel wovon) : String
wovon: ein Stammdateneintrag

Beispiel: `derort := ADBTupZROrt (tup)`

Beschreibung: Liefert den Inhalt des Felds aus *wovon* zurück, das die Rolle ZROrt hat.

Wenn keine Stammdaten gefunden wurden oder der ADB-Manager nicht initialisiert wurde, wird ein Leerstring zurückgeliefert.

Siehe auch ADBInit() und ADBZROrt().

ADBUnlockAll

Syntax: ADBUnlockAll ()

Beispiel: `ADBUnlockAll ()`

Beschreibung: Entsperrt alle Tupel der vom ADBManager verwalteten Kerndaten.

Zu beachten ist, dass lediglich die Kerndaten (und keine Unterrelationen) gelockt werden dürfen. Daher brauchen auch nur diese entsperrt werden.

Siehe auch ADBInit() und ADBLockTupel().

ADBUnlockTupel

Syntax: ADBUnlockTupel (Tupel tup)
tup: ein Tupel aus den Stammdaten

Beispiel: `ADBUnlockTupel (tup)`

Beschreibung: Ein Lock für das Tupel *tup* in den permanent gespeicherten Stammdaten wird entfernt (siehe UnlockTupel()).

Siehe auch ADBInit() und ADBLockTupel().

ADBUpdate

Syntax: ADBUpdate ()

Beispiel: `ADBUpdate ()`

Beschreibung: Prüft, ob sich die Stammdaten- oder die Benutzer-Datei geändert haben, und aktualisiert sie bei Bedarf.

Siehe auch ADBInit(), ADBChangeFields() und ADBUpdateCache().

ADBUpdateCache

Syntax: ADBUpdateCache ()

Beispiel: `ADBUpdateCache()`

Beschreibung: überprüft zu jeder Relation, die sich im Cache befindet, ob sie noch auf dem neusten Stand ist, und lädt sie gegebenenfalls neu.

Siehe auch ADBInit(), ADBOpenRel() und ADBFlushCache().

ADBZROrt

Syntax: ADBZROrt (String was) : String
was: spezifiziert die Messstelle/Station

Beispiel: `derort := ADBZROrt ("207654")`

Beschreibung: Sucht das Stammdaten-Tupel, das auf *was* passt (siehe Stammdaten()), und liefert den Inhalt des Felds daraus zurück, das die Rolle ZROrt hat.

Wenn keine Stammdaten gefunden wurden oder der ADB-Manager nicht initialisiert wurde, wird *was* zurückgeliefert.

Siehe auch ADBInit(), ADBName(), ADBBezeichnung() und ADBTupZROrt().

AddGroesse

Syntax: AddGroesse (ZR z, String sg, Intervall i, String sp, Bool b) : ZR
z:
sg: Größe
i: Berechnungszeitraum
sp: Parameter der neuen Zeitreihe
b: Temporärflag

Beispiel:
```
sum := AddGroesse (zr1, "30 mm/h", i, "Nieder2", FALSE)
```

Beschreibung: Es wird eine neue Zeitreihe erzeugt, die sich aus der Addition der Zeitreihe *z* und der Konstanten *sg* ergibt. *sg* ist als einheitbehaftete Zahl aufzufassen, deren Einheit mit der der Zeitreihe kompatibel ist. Der Parameter der neuen Zeitreihe wird auf *sp* gesetzt.

AddHandle

Syntax: AddHandle (String elementname, String azurfunktion)
elementname: Name des AGElements
azurfunktion: Name einer Azur-Funktion

Beispiel:
```
AddHandle ("gobut", "NochEinAufruf")
```

Beschreibung: Meldet *azurfunktion* als weiteren Handle für das Element *elementname* an. Die Handle werden in der Reihenfolge ihres Anmeldens ausgeführt.

Sie auch RemoveHandle() und SetHandle(), wo auch die Sonderfunktionen beschrieben sind.

AddWp

Syntax: AddWP (ZR z, XPunkt xp, Real r)
zr:
xp: Zeitpunkt oder Real
r:

Beispiel: `AddWP (z, zp, r)`

Beschreibung: Fügt das Wertepaar (xp, r) in die Reihe z ein, bzw. überschreibt ein dort vorhandenes. Achtung: Diese Prozedur nicht bei Massendaten verwenden, weil WriteQuantenfolge() viel schneller ist.

AddZR

Syntax: AddZR (ZR z1, ZR z2, ZI i, String s, Bool b [, Bool lueckzunull]) : ZR
z1:
z2:
i: Berechnungszeitraum
s: Ort der Ergebniszeitreihe
b: [**?**] lueckezunull: optional: Sollen Lücken als 0 gelten.

Beispiel: `sum := AddZR (z1, z2, MAXFOCUS, s, FALSE)`

Beschreibung: Addiert zwei Zeitreihen. Die Einheiten müssen kompatibel sein. Handelt es sich um Intervall-Zeitreihen, dann müssen die Intervallmengen exakt übereinstimen. Bei kontinuierlichen Zeitreihen findet eine Vereinigung der Stützpunktmengen statt. Zu beachten ist, dass die Ergebniszeitreihe einen neuen Ort und nicht einen neuen Parameter bekommt. Die Einheit bleibt gleich, und damit höchstwahrscheinlich auch der Parameter. Der Ort muss jedoch ein anderer sein, da dieser Parameter an einem der Ausgangszeitreihen ja schon in $zr1$ bzw. $zr2$ gespeichert ist. Typisches Anwendungsbeispiel ist die Addition zweier Abfluss-Zeitreihen in einem Pegeldreieck.

Ist *lueckezunull* True, werden Lücken in $z1$ oder $z2$ als 0 gewertet. Wenn sowohl $z1$ als auch $z2$ eine Lücke aufweisen, ist das Ergebnis **nicht 0, sondern Lücke**. Die Voreinstellung für *lueckezunull* ist False.

AGAxboxList

Syntax: AGAxboxList(String name) : String
name: Name des AGWindows

Beispiel: `s := AGAxboxList( "ag0")`

Beschreibung: Liefert die Liste der Axboxen auf dem Canvas des AGWindows *name*. Diese Liste ist eine mit Leerzeichen getrennte Stringliste. Jedes Token() repräsentiert eine AxBox. Die Funktion StrToAxBox() liefert zu einem Token die jeweilige AxBox. Mit der Funktion StrSplit() zerlegt man die Stringliste in ein Feld von Strings, das man mittels FORALL (siehe Semantik) durchlaufen kann. Die Funktion StrToArray() darf **nicht** benutzt werden, da sie die Information in der Stringliste missinterpretiert.

Beispiel:

```
sl := AGAxboxList("QBilanz")
feld := StrSplit(s, " ")
FORALL s IN feld
axb := StrToAxBox(s)
ENDFOR
```

Existiert das AGWindow nicht, ist kein Canvas definiert oder befinden sich auf diesem keine AxBoxen, wird ein Leerstring zurückgeliefert.

Siehe auch AxZRList() und CanvasOnPage().

AGElemente

Syntax: AGElemente() : Array

Beispiel: `alle := AGElemente()`

Beschreibung: Liefert die Liste aller AquaGramm-Elemente des aktuellen AGWindows. Die Elemente sind von 1 bis anzahl durchnummeriert. Der Inhalt ist jeweils ein String, der das Element beschreibt. Der Name ist das zweite Token() jedes Strings.

Siehe auch AGSetElemPos() und AGListe().

AGElemInfo

Syntax: AGElemInfo (String elename) : String
elename: Name eines Elements

Beispiel:
```
s := AGElemInfo( "eingabe1")
```

Beschreibung: Fragt den Info-String des Elements *elename* ab.

Existiert das Element nicht oder nicht auf dem aktuellen AGWindow, dann wird ein Leerstring geliefert. Wenn ein Element angelegt wird, wird ein Leerstring als Info-String hinterlegt.

Siehe auch SetAGElemInfo().

AGElemPos

Syntax: AGElemPos(String name) : GeoPoint
name: Name eines Elements

Beispiel:
```
p := AGElemPos( "eingabe1")
```

Beschreibung: Liefert die Position des Window-Elements *name* in Pixeln. Existiert das Element nicht oder nicht auf dem aktuellen AGWindow, dann wird der Nullpunkt zurückgeliefert.

Siehe auch AGSetElemPos(), AGSetElemSize() und GridPosTup.

AGElemSize

Syntax: AGElemSize(String name) : GeoPoint
name: Name eines Elements

Beispiel:
```
p := AGElemSize( "eingabe1")
```

Beschreibung: Liefert die Ausdehnung des Window-Elements *name* in Pixeln. Die X-Koordinate enthält die Breite und die Y-Koordinate die Höhe. (Siehe auch XKoo() und YKoo()).

Existiert das Element nicht oder nicht auf dem aktuellen AGWindow, dann wird der Nullpunkt zurückgeliefert.

Siehe auch AGElemPos() und AGSetElemSize().

AGListe

Syntax: AGListe() : Array

Beispiel:
```
alle := AGListe()
```

Beschreibung: Liefert die Liste aller AGWindows.

Siehe auch AGElemente().

AGParent

Syntax: AGParent (String name) : String
name: Name eines AGWindows

Beispiel:
```
 ober := AGParent ("AufPopWindow")
```

Beschreibung: Liefert den Namen des Windows, von dem aus das Window *name* gestartet wurde. Das erste Window der Applikation, das also von System erzeugt wurde, liefert `@System`.

Wenn kein Window namens *name* existiert, wird ein Leerstring geliefert.

Siehe auch SetAGWindow(), GetAGWindow() und NewAGWindow().

AGPos

Syntax: AGPos(String name) : GeoPoint
name: Name des AGWindows

Beispiel:
```
p := AGPos( "ag0")
```

Beschreibung: Liefert die linke obere Ecke des AGWindows *name*. Existiert das AGWindow nicht, dann wird der Nullpunkt zurückgeliefert.

Siehe auch AGSetPos() und AGSize().

AGRelabel

Syntax: AGRelabel()

Beispiel:
```
AGRelabel()
```

Beschreibung: Setzt die Beschriftungen aller Elemente auf allen Windows neu. So kann nach einem Sprachwechsel (siehe SetLingua()) erreichen, dass auch die schon vorhandenen Elemente übersetzt werden.

AGSetActive

Syntax: AGSetActive(String agname, Bool anaus)
agname: Name eines Windows
anaus: Aktiv: TRUE, Inaktiv: FALSE

Beispiel:
```
AGSetActive( "AG0", FALSE)
```

Beschreibung: Schaltet das gesamte Window inaktiv (FALSE) oder aktiv (TRUE). Im inaktiven Zustand kann das Window keine Eingaben empfangen und deutet dies durch einen „Grauschleier“ an.

Siehe auch SetActive().

AGSetElemPos

Syntax: AGSetElemPos(String name, GeoPoint p)
name: Name eines Elements
p: neue Position des Elements

Beispiel:
```
AGSetElemPos( "eingabe1", {100,50})
```

Beschreibung: Setzt die Position des Window-Elements *name* auf *p*. Die Änderung wird sofort sichtbar. Existiert das Element nicht oder nicht auf dem aktuellen AGWindow, dann hat der Aufruf keine Wirkung.

Siehe auch AGElemPos().

AGSetElemSize

Syntax: AGSetElemSize(String name, GeoPoint p)
name: Name eines Elements
p : neue Größe in Pixeln

Beispiel:
```
AGSetElemSize( "eingabe1", {100,200})
```

Beschreibung: Setzt die Ausdehnung des Window-Elements *name* in Pixeln. Die X-Koordinate enthält die Breite und die Y-Koordinate die Höhe.

Existiert das Element nicht oder nicht auf dem aktuellen AGWindow, dann hat der Aufruf keine Wirkung.

Siehe auch AGElemSize().

AGSetFocus

Syntax: AGSetFocus(String name)
name: Name eines Elements

Beispiel:
```
AGSetFocus("eingabe1")
```

Beschreibung: Normalerweise wird ein Element durch Anklicken mit der Maus für die Eingabe über die Tastatur bereit gemacht. Dies wird bei Eingabefeldern durch das Blinken des Cursors und bei Buttons durch eine leichte Umrandung kenntlich gemacht.

Mit AGSetFocus kann man explizit ein Element wählen, das die Eingabe empfangen soll.

Mit Tab und Shift-Tab kann der Benutzer das Element wechseln, das die Eingabe empfangen kann.

AGSetPos

Syntax: AGSetPos(String name, GeoPoint lo, GeoPoint ru)
name: Name des AGWindows
lo: neue linke obere Ecke des AGWindows
ru: neue rechte untere Ecke des AGWindows

Beispiel:
```
AGSetPos("ag0", {100,50}, {600,350})
```

Beschreibung: Setzt die Position des AGWindows *name* auf *lo* und *ru*. Die Angaben sind in Pixeln, relativ zur linken oberen Ecke des Bildschirms. Die Änderung wird sofort sichtbar. Existiert das AGWindow nicht, dann hat der Aufruf keine Wirkung.

Siehe auch AGPos() und AGSize().

AGSetShading

Syntax: AGSetShading (String vonfarbe, String bisfarbe)
vonfarbe: Farbe für die Farbnummer 100
bisfarbe: Farbe für die Farbnummer 200

Beispiel:
```
AGSetShading ("Blau", "Rot")
```

Beschreibung: Setzt die untere und obere Farbe für den Farbverlauf. Der Farbe mit der Nummer 100 wird $vonfarbe$ zugeordnet, der Nummer 200 $bisfarbe$. Die Farbnummern dazwischen nehmen in 100 Schattierungen die Übergangsfarben an. Im Beispiel ist 150 Violett.

Diese Einstellung hat globale Wirkung.

Siehe auch IsoFlaechen().

AGSetTitel

Syntax: AGSetTitel (String name, String titel)
name: Name des AGWindows
titel: neuer Titel

Beispiel:
```
AGSetTitel("AGstamm", "Koselbruch")
```

Beschreibung: Setzt den Text in der Titelzeile des AGWindows *name* auf *titel*.

Siehe auch AGRelabel() und AGSetPos().

AGShow

Syntax: AGShow(String name)
name: Name des AGWindows

Beispiel: `AGShow( "AGPop2")`

Beschreibung: Stellt ein AGWindow dar, das zuvor aufgebaut wurde. Diese Funktion ist nur in Verbindung mit dem *sofort*-Parameter von NewAGWindow() sinnvoll.

Der Aufruf von AGShow auf ein schon dargestelltes AGWindow hat keine Auswirkung.

AGSize

Syntax: AGSize(String name) : GeoPoint
name: Name des AGWindows

Beispiel: `p := AGSize( "ag0")`

Beschreibung: Liefert die Breite und Höhe des AGWindows *name*. Existiert das AGWindow nicht, dann wird der Nullpunkt zurückgeliefert.

Siehe auch AGSetPos() und AGPos().

AGToTop

Syntax: AGToTop(String name)
name: Name des AGWindows

Beispiel: `AGToTop( "AGPop2")`

Beschreibung: Holt das AGWindow *name* in den Vordergrund des Bildschirms.

Der Aufruf von AGToTop auf ein nicht dargestelltes AGWindow hat keine Auswirkung.

Siehe auch SetAGWindow().

AkimaSpline

Syntax: AkimaSpline (QL qf, Real ftoleranz) : QL
qf: Quantenfolge mit Punktquanten
ftoleranz: zum Linearisieren

Beispiel:
```
akima := AkimaSpline(qf, 0.05)
```

Beschreibung: Aus den Datenpunkten in qf wird ein Akimaspline berechnet. Ein Akimaspline ist ein **interpolierender** Spline, der jedoch im Gegensatz zu regulären Splines nur einmal ableitbar ist. Es ist nicht möglich, den Akimaspline an vorgegebene Randsteigungen anzupassen. Der Spline wird zwischen den ersten und den letzten Punkt gelegt. Die in qf vorgegebenen Punkte werden genau getroffen. Siehe dazu [11].

Die $ftoleranz$ wird benötigt, um die kubischen Verläufe des Splines zu linearisieren.

Siehe auch KubischerSpline() (**approximierender** Spline).

AnzahlWerte

Syntax: AnzahlWerte (ZR reihe, Intervall xi) : Real
reihe: eine Reihe
xi: Bezugsintervall

Beispiel:
```
n := AnzahlWerte (zr, MaxFocusZR(zr))
```

Beschreibung: Liefert die Anzahl der Wertepaare der Zeitreihe *reihe* auf dem Intervall xi der höchsten Qualität.

Für große Reihen dauert die Berechnung u.U. sehr lange, da alle Daten eingelesen werden; besonders gilt dies für Reihen mit mehreren Qualitäten. Wenn die genaue Anzahl der Werte nicht interessant ist, sondern die Zahl nur einen groben Überblick verschaffen soll, empfiehlt sich die Konstante $MAXFOCUS$. Wird diese übergeben, dann wird die Anzahl der Werte anhand der Zeitreihengröße geschätzt. Beispiel

```
n := AnzahlWerte (zr, MAXFOCUS)
```

AnzQuanten

Syntax: AnzQuanten (QuantList ql) : Real
ql: eine Quantenliste

Beispiel: `i := AnzQuanten( ql )`

Beschreibung: Liefert die Anzahl der Quanten in der QuantList *ql*.

AnzTupel

Syntax: AnzTupel (Relation R) : Real
R: eine Relation

Beispiel: `i := AnzTupel (stammrel)`

Beschreibung: Liefert die Anzahl Tupel, die in einer Relation enthalten sind. Die Numerierung der Tupel läuft von 0 bis AnzTupel()-1

AnzZR

Syntax: AnzZR (ZRList zrl) : Real
zrl: eine Zeitreihenliste

Beispiel: `i := AnzZR (zrl)`

Beschreibung: Liefert die Anzahl der Zeitreihen in der ZRList *zrl*.

AppendQF

Syntax: AppendQF (QuantList ql1, QuantList ql2)
ql1: eine Quantenliste
ql2: eine andere Quantenliste

Beispiel: `AppendQF(qf, qfklein)`

Beschreibung: Hängt die Quantenliste $ql2$ an die Quantenliste $ql1$ an. Die Quanten werden nicht kopiert, sondern lediglich verschoben, so dass $ql2$ nach dem Aufruf leer ist.

AppendQuant

Syntax: AppendQuant(QuantList ql, Quant q)
ql: eine Quantenliste
q: ein Quant

Beispiel: `AppendQuant(ql, lq)`

Beschreibung: Hängt das Quant lq an die Quantenfolge ql an. Vorne anfügen mit PrependQuant().

Siehe auch DelQuant().

AppField

Syntax: AppField(Relation R, String aufbau)
R: Relation
feld: Formatbeschreibung des neuen Felds

Beispiel: `AppField( mrel, "NeuName#20s" )`

Beschreibung: Fügt allen Tupeln der Relation R ein neues Feld hinzu, das durch $aufbau$ beschrieben ist. Das Feld wird bei allen Tupeln als unbesetzt markiert.

Die Syntax des Formates ist im Anhang beschrieben.

AppTupel

Syntax: AppTupel (Relation R, Tupel t)
R:
t: Tupel

Beispiel: `AppTupel( Stamm, t )`

Beschreibung: Eine Kopie des Tupels t wird in die Relation R aufgenommen. Mögliche Freiräume mit ungültigen Tupeln werden dabei gefüllt (bezieht sich auf dbf-Relationen).

Wichtig: wenn t aus einer dbf-Relation stammt, ist in ihm die Recordnummer aus der Datei vermerkt (siehe TupRecNum()). Wird t in eine Memory-Relation geschrieben, behält es seine Recordnummer. Dies kann hinderlich sein, wenn die Memory-Relation mit einer weiteren dbf-Relation abgeglichen wird, da die Recordnummer auf die neue dbf-Relation nicht passt. Dies ist **insbesondere** beim ADBManager (siehe ADBStammRel()) der Fall. Da **muss** die Recordnummer des Tupels mittels TupClearRecNum() vor dem Append gelöscht werden.

Siehe auch Rewrite() und Search().

ArcCos

Syntax: ArcCos (Real r) : Real
r:

Beispiel: `winkel := ArcCos (arg)`

Beschreibung: Liefert den Arcus Cosinus von arg

ArcSin

Syntax: ArcSin (Real r) : Real
r:

Beispiel: `winkel := ArcSin (arg)`

Beschreibung: Liefert den Arcus Sinus von arg

ArcTan

Syntax: ArcTan (Real r) : Real
r:

Beispiel: `winkel := ArcTan (arg)`

Beschreibung: Liefert den Arcus Tangens von arg

Array

Syntax: Array() : Array

Beispiel: `feld := Array ()`

Beschreibung: Erzeugt ein neues Array.
Siehe auch StrToArray() und ArraySize().

ArraySize

Syntax: ArraySize(Array a) : Real
a : ein Array

Beispiel: `anz := ArraySize (feld)`

Beschreibung: Liefert die Anzahl der Elemente im Array a.
Siehe auch Array() und StrToArray().

Attribute

Syntax: Attribute (ZR z) : Tupel
z:

Beispiel: `atrb := Attribute (z)`

Beschreibung: Liefert die Attribute der Zeitreihe z zurück.

Aussage

Syntax: Aussage (ZR z) : String
z:

Beispiel: `a := Aussage (z)`

Beschreibung: Liefert die statistische Aussage der (typischerweise Intervall-) Zeitreihe. Dies kann z.B. Mit für Mittelwert sein. Das Vorhandensein dieser Funktion beleuchtet einen wichtigen Kernaspekt des aqua_plan-Zeitreihenkonzepts: Werden aus einer Zeitreihe Intervalle gebildet, dann geschieht dies **immer** mittels einer statistischen Funktion (z.B. Mittelwert, Summe, Maximum ...). Intervallbildung ohne Spezifikation dieser Funktion ist nicht möglich. So wird einem „Zeitreihen-Wildwuchs“ zentral vorgebeugt.

AxBoxOnPage

Syntax: AxBoxOnPage (Page seite, AxBox ax)
seite: Eine Reportseite
ax: Eine AxBox

Beispiel: `AxBoxOnPage (seite, ax)`

Beschreibung: Zeichnet die AxBox *ax* und alle in ihr enthaltenen Zeitreihen (siehe ZRInAxBox()) auf die Reportseite *seite*. Die Position der AxBox muss vorher mit SetAxLage() festgelegt werden. Ebenfalls muss der Bereich der X-Achse mit SetAxXBereich() festgelegt werden.

Siehe auch CanvasOnPage().

AxClearKonsts

Syntax: AxClearKonsts(AxBox ax)
ax:

Beispiel: `AxClearKonsts(axoben)`

Beschreibung: Löscht die Liste der Y-Konstanten in der AxBox *ax*.

Siehe auch YKonstInAxBox() und ClearAxBox().

AxDelDrawing

Syntax: AxDelDrawing(AxBox ax)
ax: eine AxBox

Beispiel:
```
AxDelDrawing (ax)
```

Beschreibung: Löscht alle Graphikelemente, die mit AxDrawLine(), AxDrawSymbol(), AxDrawText() oder AxDrawKreis() in die AxBox *ax* gezeichnet wurden. Die Graphik auf einem Canvas wird nicht verändert. Diese muss durch erneutes Plotten oder durch ClearCanvas() angepasst werden.

AxDelZR

Syntax: AxDelZR (AxBox ax, Real num [, Bool mitlegende])
ax: eine AxBox
num: Nummer der zu löschenden Reihe
mitlegende: optional: auch der Legendeneintrag wird gelöscht

Beispiel:
```
AxDelZR (ax, 1)
```

Beschreibung: Löscht eine Reihe aus der AxBox *ax*. *num* bezeichnet die Nummer der zu löschenden Reihe in der AxBox. Die erste Reihe hat die Nummer 1, die letzte die Nummer AnzZR(AxZRList(ax)). Liegt *num* außerhalb dieses Bereichs, wird der Befehl ignoriert.

Der Parameter *mitlegende* legt fest, ob der Legendeneintrag der Reihe auch gelöscht werden soll. Voreinstellung ist False, also nein. Hierbei ist Folgendes zu beachten:

Durch ZRInAxBox() wurde der Reihe ein Legendeneintrag zugewiesen. Durch die Reihenfolge der Reihen ergibt sich eine Verbindung zwischen Reihe und zugehörigem Legendeneintrag. Dann ist es sinnvoll, *mitlegende* auf True zu setzen. Diese Verbindung zwischen Reihe und Legendeneintrag kann jedoch veloren gehen, wenn die Legende explizit mittels SetLegende() verändert worden ist. In diesem Fall sollte *mitlegende* auf False gesetzt werden.

Siehe auch SetAxLegPos()

AxDrawKreis

Syntax: AxDrawKreis(AxBox ax, XPunkt x, Real y, Real radius, Real san, Real ean, Bool fill)
ax: eine AxBox
x: Zeitpunkt oder Real
y:
radius: Radius in cm
san: Startwinkel des Kreisbogens
ean: Endwinkel des Kreisbogens
fill: TRUE=gefüllt

Beispiel:
```
AxDrawKreis ( ax, @"1985", 20, 4.5, 0, 360)
```

Beschreibung: Zeichnet einen Kreisbogen mit Mittelpunkt (x,y) in die AxBox *ax*. *radius* gibt den Radius des Kreises in cm auf dem Papier/Canvas an. *san* und *ean* bestimmen die Ausdehnung des Kreisbogens in Grad. Im Beispiel wird ein vollständiger Kreis gezeichnet. *fill* legt fest, ob der Kreis gefüllt werden soll.

Die Farbe des Kreises wird mit AxSetColor() festgelegt.

Siehe auch AxDrawLine(), AxDrawSymbol() und AxDrawText().

AxDrawLine

Syntax: AxDrawLine(AxBox ax, XPunkt x1, Real y1, XPunkt x2, Real y2 [, Bool oben])
ax: eine AxBox
x1: Zeitpunkt oder Real
y1:
x2: Zeitpunkt oder Real
y2:
oben: optional: True=Linie wird über die Zeitreihen gezeichnet. Standard: False

Beispiel:
```
AxDrawLine ( ax, @"1985", 100, @"1986", 0 )
```

Beschreibung: Zeichnet eine Linie vom Punkt (x1,y1) zum Punkt (x2,y2) in der AxBox *ax*.

Die Farbe der Linie wird mit AxSetColor() festgelegt, die Liniendicke mit AxSetLineWidth() und das Linienmuster mit SetLineStyle().

Standardmäßig werden alle Linien vor den Zeitreihen in die Axbox gezeichnet. Die Linie wird also von jenen überdeckt. Wenn dies nicht gewünscht ist, die Linie also die Zeitreihen überdecken soll, gibt man als letzten Parameter `True` an. Dies ist vor allem sinnvoll, wenn man erreichen will, dass die X-Achse nicht von den Zeitreihen (z.B. Lücken oder Säulen) durchbrochen wird.

Siehe auch AxDrawSymbol(), AxDrawText() und AxDrawKreis().

AxDrawSymbol

Syntax: AxDrawSymbol(AxBox ax, XPunkt x, Real y, Real sym)
ax: eine AxBox
x: Zeitpunkt oder Real
y:
sym: Symbol (siehe SetSymbolTyp())

Beispiel:
```
AxDrawSymbol ( ax, @"1985", 34.5, 3)
```

Beschreibung: Zeichnet das Symbol *sym* an die Stelle (x,y) der AxBox *ax*.

Die Farbe des Symbols wird mit AxSetColor() festgelegt.

Siehe auch AxDrawLine(), AxDrawText() und AxDrawKreis().

AxDrawText

Syntax: AxDrawText(AxBox ax, XPunkt x, Real y, String txt)
ax: eine AxBox
x: Zeitpunkt oder Real
y:
txt: zu zeichnender Text

Beispiel:
```
AxDrawText ( ax, @"1985", 20, "Heute Stichtag")
```

Beschreibung: Zeichnet den Text *txt* an die Stelle (x,y) der AxBox *ax*.

Die Farbe des Textes wird mit AxSetColor() festgelegt. Dessen Ausrichtung mit AxSetAlign().

Siehe auch AxDrawLine(), AxDrawSymbol() und AxDrawKreis().

AxInfo

Syntax: AxInfo (AxBox ax) : Tupel
ax: Eine AxBox

Beispiel:
```
attr := AxInfo (ax)
```

Beschreibung: Liefert die gegenwärtigen Eigenschaften der AxBox *ax*. Rückgabewert ist ein Tupel mit folgendem Aufbau

Feldname	Feldtyp	Beschreibung
vollbild	Bool	Ist die AxBox großgeklickt?
sichtbar	Bool	Ist die AxBox dargestellt?
aktiv	Bool	ist die AxBox die aktive Axbox?
hidebs	Bool	werden Bearbeitungsstände farbig hinterlegt?
axanz	Number1	Anzahl der Y-Achsen (1-4)
*unit*0	String20	Einheit der ersten Y-Achse
*unit*1	String20	Einheit der zweiten Y-Achse
*unit*2	String20	Einheit der dritten Y-Achse
*unit*3	String20	Einheit der vierten Y-Achse

Zum Auslesen der Attribute kann z.B. GetBool() benutzt werden.

Siehe auch AxLageLU(), AxLageRO() und AxLegPos().

AxLageLU

Syntax: AxLageLU (AxBox ax) : GeoPoint
ax: Eine AxBox

Beispiel:
```
plu := AxLageLU (ax)
```

Beschreibung: Liefert die Koordinate der linken unteren Ecke der AxBox *ax* auf dem Papier (oder dem Canvas, dem Report). Die Angaben sind in cm.

Siehe auch AxLageRO(), SetAxLage(), PagePos() und Plot().

AxLageRO

Syntax: AxLageRO (AxBox ax) : GeoPoint
ax: Eine AxBox

Beispiel:
```
plu := AxLageRO (ax)
```

Beschreibung: Liefert die Koordinate der rechten oberen Ecke der AxBox *ax* auf dem Papier (oder dem Canvas, dem Report). Die Angaben sind in cm.

Siehe auch AxLageLU(), SetAxLage(), PagePos() und Plot().

AxLegPos

Syntax: AxLegPos (AxBox ax) : GeoPoint
ax: Eine AxBox

Beispiel:
```
proleg := AxLegPos (ax)
```

Beschreibung: Liefert die Koordinate der linken oberen Ecke der Legende der AxBox *ax*. Die Angaben sind in cm.

Siehe auch SetAxLegPos(), AxLageLU() und AxLageRO().

AxMarkerBereich

Syntax: AxMarkerBereich (AxBox ax) : Intervall
ax: eine AxBox

Beispiel:
```
i := AxMarkerBereich (ax)
```

Beschreibung: Fragt die Position der Markierungen ab, die mit AxSetMarker() gesetzt wurden. Das gelieferte Intervall ist ein Zeitintervall (siehe Links() und Rechts()) oder ein Realintervall (siehe LinksReal() und RechtsReal()) , je nach Typ der X-Achse von *ax*.

Falls keine Markierungen gesetzt wurden, wird ein ungültiges Intervall geliefert (siehe IsValid()).

AxReplaceYTexts

Syntax: AxReplaceYTexts (AxBox ax, String liste)
ax:
liste: Liste von Y-Skalierungstexten mit Leerzeichen getrennt

Beispiel:
```
AxReplaceYTexts (axoben, "Nairobi Thika Kiambu Kajiado")
```

Beschreibung: Die Beschriftung der Y-Achse wird ersetzt durch Texte in *liste*. An jeder Stelle, an der normalerweise eine Zahl stehen würde, wird, in aufsteigender Reihenfolge, ein Text aus *liste* eingesetzt. *liste* ist eine durch Leerzeichen getrennte Liste von Wörtern (siehe Token()). Falls Leerzeichen in den Wörtern auftauchen, müssen die Wörter in Anführungszeichen eingefasst sein.

Überschüssige Wörter in *liste* werden ignoriert. Sind zuwenig Wörter vorhanden, bleiben die restlichen Texte unverändert Zahlen.

Zu beachten ist, dass nicht an alle Skalierungsstriche Zahlen geschrieben werden. Die hier übergebene Liste mit Ersatztexten wird nur auf ursprünglich beschriftete Skalierungstexte angewendet.

Siehe auch SetAxScalDist() und SetAxLage().

AxSetAlign

Syntax: AxSetAlign(AxBox ax, String align)
ax: eine AxBox
align: Ausrichtung von Texten

Beispiel:
```
AxSetAlign ( ax, "Mitte")
```

Beschreibung: Setzt die Ausrichtung der Texte der AxBox *ax* auf *align*. *align* kann die Werte LINKS, RECHTS oder MITTE (oder LEFT,RIGHT,CENTER) annehmen. Diese Textausrichtung gilt für alle im Folgenden mit AxDrawText() ausgegebenen Texte.

AxSetColor

Syntax: AxSetColor(AxBox ax, String farbe)
ax: eine AxBox
farbe: Siehe ZRInAxBox()

Beispiel:
```
AxSetColor ( ax, "Blau")
```

Beschreibung: Setzt die Farbe der AxBox *ax* auf *farbe*. Diese Farbe wird benutzt, wenn mit den Funktionen AxDrawLine(), AxDrawSymbol(), AxDrawText() und AxDrawKreis() direkt in die AxBox gezeichnet wird. Sie hat keinen Einfluss auf die Darstellung von Zeitreihen.

AxSetLineWidth

Syntax: AxSetLineWidth(AxBox ax, Real breite)
axbox:
breite: Liniendicke in cm

Beispiel:
```
AxSetLineWidth(axoben, 0.1 )
```

Beschreibung: Legt die Liniendicke fest, mit der alle folgenden Zeitreihen gezeichnet werden. Voreinstellung ist 0.02 cm.

Legt auch die Liniendicke fest, mit der Linien mittels AxDrawLine() gezeichnet werden.

Siehe auch ZRInAxBox und AxSetSymSize()

AxSetMarker

Syntax: AxSetMarker (AxBox ax, Real symbol, XPunkt von, XPunkt bis)
ax: eine AxBox
symbol: siehe SetSymbolTyp()
von: Position des linken Markers (Zeitpunkt oder Real)
bis: Position des rechten Markers (Zeitpunkt oder Real)

Beispiel:
```
AxSetMarker (ax, 6, zpvon, zpmax)
```

Beschreibung: Setzt zwei Markierungen unter die X-Achse von *ax*. Diese können mit der Maus bewegt werden. Die aktuelle Position beider Markierungen kann mit AxMarkerBereich() abgefragt werden.

Um die Markierungen abzuschalten, übergibt man an Stelle eines Symbols die Zahl -1.

AxSetSymSize

Syntax: AxSetSymSize(AxBox ax, Real size)
ax: eine AxBox
size: Größe von Symbolen in cm

Beispiel:
```
AxSetSymSize (ax, 0.2)
```

Beschreibung: Legt die Größe von Symbolen fest, mit der alle folgenden Zeitreihen gezeichnet werden.

Legt auch die Größe fest, mit der Symbole mittels AxDrawSymbol() gezeichnet werden.

Siehe auch AxSetLineWidth() und AxSetColor().

AxSetTextsize

Syntax: AxSetTextsize (AxBox ax, Real size)
ax:
size: Höhe in cm

Beispiel: `AxSetTextsize (axoben, 0.4)`

Beschreibung: Setzt die Größe von Texten, die in der AxBox gezeichnet werden. Das sind Kommentare, Beschriftungen von Konstanten, Legendentexte und Texte, die mittels AxDrawText() gezeichnet werden.

Die Voreinstellung ist 0,2 cm.

Die Größe der Skalierungstexte wird mit der Prozedur SetAxTextsize() gesetzt.

AxSetXGitter

Syntax: AxSetXGitter(AxBox ax, Bool b [, String farbe])
ax:
b:
farbe: optional: Farbe der Gitterstriche, Voreinstellung: Schwarz

Beispiel: `AxSetXGitter( axbox1, TRUE )`

Beschreibung: Legt fest, ob in die AxBox senkrechte Gitterlinien gezeichnet werden sollen. Zu *farbe* siehe auch ZRInAxBox().

Siehe auch AxSetYGitter() und SetAxGitter()

AxSetYGitter

Syntax: AxSetYGitter(AxBox ax, Bool b [, String farbe])
ax:
b: farbe: optional: Farbe der Gitterstriche, Voreinstellung: Schwarz

Beispiel: `AxSetYGitter( axbox1, TRUE, "Hellgrau" )`

Beschreibung: Legt fest, ob in die AxBox waagerechte Gitterlinien gezeichnet werden sollen. Zu *farbe* siehe auch ZRInAxBox().

Siehe auch AxSetXGitter() und SetAxGitter()

AxXBereich

Syntax: AxXBereich(AxBox ax) : Intervall
ax: Eine AxBox

Beispiel:
```
bereich := AxXBereich (ax)
```

Beschreibung: Liefert den Bereich der X-Achse der AxBox *ax*. Dieser Bereich ist entweder ein Zeitintervall oder ein Realintervall.

Siehe auch AxYBereich() und SetAxXBereich()

AxYBereich

Syntax: AxYBereich(AxBox ax) : Intervall
ax: Eine AxBox

Beispiel:
```
bereich := AxYBereich (ax)
```

Beschreibung: Liefert den Bereich der Y-Achse der AxBox *ax*. Dieser Bereich ist ein Realintervall.

Siehe auch AxXBereich() und SetAxYBereich()

AxZRColor

Syntax: AxZRColor (AxBox ax, ZR zr [, Real qual]) : String
ax: Eine AxBox
zr: Eine Reihe der AxBox
qual: optional: Qualität der Reihe

Beispiel:
```
farbe := AxZRColor (ax, zr)
```

Beschreibung: Liefert die Farbe, in der die Reihe *zr* in der AxBox *ax* dargestellt wird. Ist *zr* nicht in *ax* enthalten, wird ein Leerstring geliefert. Ist die Reihe mehrfach (mit unterschiedlichen Qualitäten) in der AxBox enthalten, kann man mit dem optionalen Parameter *qual* auswählen, welche gemeint ist.

Siehe auch AxZRLineWidth(), AxZRLineStyle() und ZRInAxBox().

AxZRLineStyle

Syntax: AxZRLineStyle (AxBox ax, ZR zr [, Real qual]) : Real
ax: Eine AxBox
zr: Eine Reihe der AxBox
qual: optional: Qualität der Reihe

Beispiel:
```
ls := AxZRLineStyle (ax, zr)
```

Beschreibung: Liefert die Linienart, in der die Reihe *zr* in der AxBox *ax* dargestellt wird. Ist *zr* nicht in *ax* enthalten, wird 0.0 geliefert. Ist die Reihe mehrfach (mit unterschiedlichen Qualitäten) in der AxBox enthalten, kann man mit dem optionalen Parameter *qual* auswählen, welche gemeint ist.

Siehe auch SetLineStyle(), AxZRLineWidth() und AxZRColor().

AxZRLineWidth

Syntax: AxZRLineWidth (AxBox ax, ZR zr [, Real qual]) : Real
ax: Eine AxBox
zr: Eine Reihe der AxBox
qual: optional: Qualität der Reihe

Beispiel:
```
lw := AxZRLineWidth (ax, zr)
```

Beschreibung: Liefert die Liniendicke, in der die Reihe *zr* in der AxBox *ax* dargestellt wird. Ist *zr* nicht in *ax* enthalten, wird 0.0 geliefert. Ist die Reihe mehrfach (mit unterschiedlichen Qualitäten) in der AxBox enthalten, kann man mit dem optionalen Parameter *qual* auswählen, welche gemeint ist.

Siehe auch AxZRColor(), AxZRLineStyle() und ZRInAxBox().

AxZRList

Syntax: AxZRList (AxBox ax) : ZRList
ax:

Beispiel:
```
zrl := AxZRList (axoben)
```

Beschreibung: Liefert die Zeitreihen, die zurzeit in der AxBox dargstellt werden oder werden sollen. Siehe auch ClearAxBox(), AxDelZR() und ZRInAxBox().

Base64ToStr

Syntax: Base64ToStr(String s) : String
s : Ein String

Beispiel:
```
wally := Base64ToStr ("Z2FzdDpnYXN0")
```

Beschreibung: Wandelt einen Base64-String in Klartext um. Dies geschieht nach RFC3548, wobei, sowohl `_` und `-` als auch `+` und `/` erkannt werden.

Siehe auch StrToBase64() und MD5Sum().

BearbStaende

Syntax: BearbStaende (ZR reihe, Intervall focus) : ZR
reihe: zu behandelnde Reihe
focus:

Beispiel:
```
stzr := BearbStaende (niederzr, niederzr.MaxFocusZR())
```

Beschreibung: Liefert die Abfolge der Bearbeitungsstände auf *focus* der Zeitreihe *reihe* als Intervallreihe. Die Ergebniszeitreihe ist temporär.

Siehe auch ZRBearbStand().

Beep

Syntax: Beep()

Beispiel: `Beep()`

Beschreibung: Gibt einen Signalton aus.

Benutzer

Syntax: Benutzer () : Tupel

Beispiel: `user := Benutzer ()`

Beschreibung: Gibt ein Tupel mit Information über den augenblicklichen Benutzer zurück. Der Aufbau des Tupels ist applikationsspezifisch. Bei unbekanntem Benutzer ist das Tupel ungültig (IsValid()).

Siehe ADBInit() und UpdateBenutzer().

BitmapOnPage

Syntax: BitmapOnPage(Page page, GeoPoint pos, GeoPoint size, String datei)
page:
pos: obere, linke Ecke
size: Breite, Höhe
datei: Grafikdatei (zurzeit nur BMP möglich)

Beispiel: `BitmapOnPage(page, {16,25}, {3,2}, "logo.bmp")`

Beschreibung: Zeichnet ein Bitmap aus der datei *datei* auf die Seite *page*. *pos* und *size* geben dabei die Lage des Bitmaps an. Zurzeit werden nur Ausgaben im PDF-Format unterstützt.

Siehe auch PrintPage() und DrawTextOnPage().

BitMask

Syntax: BitMask (Real wert, Real lsb, Real msb) : Real
wert: Ein ganzahliger Wert
lsb: Nummer des untersten Bits (beginnt bei 0)
msb: Nummer des obersten Bits

Beispiel:
```
maskiert := BitMask (zahl, 8, 10)
```

Beschreibung: BitMask ist eine Funktion, die auf Bitebene mit ganzzahligen Werten (Integern) arbeitet.

Liefert den Ausschnitt von Bit *lsb* bis Bit *msb* einschließlich.

Beispiel: Die Zahl 12345 hat die Binärdarstellung 11000000111001. `BitMask(12345, 4, 7)` liefert binär 0011, also 3.

Siehe auch CharVal().

Breite

Syntax: Breite (Intervall i) : Distanz
i: Zeitintervall

Beispiel:
```
br := Breite (i)
```

Beschreibung: Die Breite eines Zeitintervalls als Differenz der rechten und der linken Grenze des Intervalls. Diese Funktion ist nicht auf Realintervall anwendbar.

CanvasOnPage

Syntax: CanvasOnPage (Page seite, GeoPoint lu, GeoPoint ro)
seite: Eine Reportseite
lu: Punkt links unten in cm
ro: Punkt rechts oben in cm

Beispiel:
```
CanvasOnPage (seite, {2,10}, {28,20})
```

Beschreibung: Zeichnet den gesamten Canvas des aktuellen Windows auf die Seite *seite*. Die Punkte *lu* und *ro* geben die Lage des Canvas auf der Seite in cm an. Der Canvas kann also in x- und y-Richtung gedehnt oder gestaucht werden!

Siehe auch AxBoxOnPage(), AGAxboxList() und GeoCanvasOnPage().

CatchSignal

Syntax: CatchSignal (Real sig, String handle)
sig: Signalnummer
handle: Name einer Azurprozedur

Beispiel:
```
CatchSignal(15, "MachWas")
```

Beschreibung: Legt fest, dass die Prozedur *handle* aufgerufen wird, wenn der Prozess das Signal *sig* empfängt. Die Signalnummer wird *handle* übergeben.

Beispiel für einen Handler: `MachWas (Real sig)`

Siehe auch SendSignal() und GetPID().

ChangeDir

Syntax: ChangeDir (string name)
name: Name des Verzeichnisses

Beispiel:
```
ChangeDir("daten")
```

Beschreibung: Wechselt in das Verzeichnis *name* (siehe den Betriebssystem-Befehl cd).

Siehe auch MakeDir().

Char

Syntax: Char (Real num) : String
num:

Beispiel:
```
s := s + Char(27)
```

Beschreibung: Liefert das Ascii-Zeichen mit der Nummer *num*. Im Beispiel wird ein Escape-Zeichen erzeugt und an einen String angehängt. Diese Funktion ist notwendig, um Zeichen, die nicht als Sonderzeichen mittels \ erzeugt werden können, zu benutzen, z.B. für die Bildschirmsteuerung.

Siehe auch CharVal().

CharVal

Syntax: CharVal (String was [, Real idx]) : Real
was: String
idx: Position im String (erste Position ist 0), default ist 0

Beispiel:
```
wert := CharVal("A")
```

Beschreibung: Liefert den Ascii-Code eines Zeichens. Ohne Angabe von *idx* wird das erste Zeichen des Strings ausgewertet, sonst das an Position *idx*.

Ist *was* ein Leerstring oder verweist *idx* nicht auf eine korrekte Position im String, so wird -1 geliefert.

Siehe auch Char().

ClearAxBox

Syntax: ClearAxBox(AxBox ax)
ax:

Beispiel:
```
ClearAxBox(axoben)
```

Beschreibung: Löscht die Liste der Zeitreihen, die Legende und alle weiteren Einträge der AxBox *ax*. Siehe auch AxZRList und ZRInAxBox.

ClearCanvas

Syntax: ClearCanvas()

Beispiel:
```
ClearCanvas()
```

Beschreibung: Löscht die gesamte Graphik auf dem Canvas des Aquagramms. Siehe Plot().

ClearModify

Syntax: ClearModify(Tupel t)
t: Tupel

Beispiel: `ClearModify(tup)`

Beschreibung: Löscht explizit die Marke, die anzeigt, dass das Tupel sich verändert hat.

Siehe auch IsModified().

ClearPage

Syntax: ClearPage(Page page)
page: eine Reportseite

Beispiel: `ClearPage(page)`

Beschreibung: Löscht den Inhalt der Seite. Siehe auch NewPage().

ClientExec

Syntax: ClientExec (String s)
s: String, der den Systemaufruf enthält.

Beispiel: `ClientExec ("cmd.exe")`

Beschreibung: Setzt den String *s* als Systemaufruf auf dem Client ab.

Diese Prozedur wird benötigt, wenn das Azurprogramm auf einem Server läuft, der mittels AQTP an einen Client angebunden ist, der die Benutzerschnittstelle bereitstellt.

Falls kein AQTP eingesetzt wird, ist diese Prozedur identisch mit System().

ClientPathList

Syntax: ClientPathList (String path, Bool dirs) : Array
path: Pfad
dirs: True=Verzeichnisse, False=Dateien

Beispiel:
```
A := ClientPathList ("eingang", False)
```

Beschreibung: Liefert die Liste mit Verzeichnissen (oder Dateien) des Pfads *path* auf dem Client.

Diese Funktion wird nur für den AquaWeb-Betrieb benötigt.

Siehe auch DirList() und FileList().

CloseFile

Syntax: CloseFile(String filename)
filename: Name der Datei

Beispiel:
```
CloseFile( "daten.txt" )
```

Beschreibung: Schließt die Datei *filename*.

Die Anzahl offener Dateien ist beschränkt (Unix ca. 60, DOS ca. 15). Da ein ReadLine aus einer Datei diese automatisch öffnet und offenhält, kann diese Anzahl leicht überschritten werden. CloseFile nimmt die Datei aus der Liste offener Dateien heraus.

Wenn nach einem CloseFile auf eine Datei zugegriffen wird (ReadLine()), dann wird sie abermals geöffnet, befindet sich jedoch wieder am Anfang (siehe Rewind()).

CollectAll

Syntax: CollectAll (Relation R, Bool sel) : Relation
R: Memory-Relation
sel: True=wähle selektierte, False=wähle unselektierte

Beispiel: `mrel := CollectAll(stamm, True)`

Beschreibung: Liefert alle selektierten oder alle unselektierten Tupel der Relation *R*. Das Ergebnis ist eine Memory-Relation. *R* muss ebenfalls eine Memory-Relation sein, da nur Tupel aus Memory-Relationen den Status selektiert beibehalten.

Tupel können mittels SelectAll() oder TupSelect() selektiert oder deselektiert werden.

Siehe auch TupIsSelected().

CompileStr

Syntax: CompileStr () : String

Beispiel: `print (CompileStr())`

Beschreibung: Liefert einen String, der die Erstellungzeiten der EXE-Datei und des ao-Moduls enthält. Sind mehrere, dynamisch dazugeladene ao-Module im Spiel, wird das ausgewählt, welches das Element erzeugt hat, dessen Handle ausgeführt wurde.

Der String hat die Form `EXE:`exe-Zeit`,`ao-Name `:`ao-Zeit.

CopyTupel

Syntax: CopyTupel (Tupel quelle) : Tupel
quelle: Quell-Tupel

Beispiel: `tup := CopyTupel(anderes)`

Beschreibung: Kopiert den Inhalt des Tupels *quelle* Feld für Feld in ein neues Tupel , welches zurückgegeben wird.

Wenn *quelle* ungültig ist, wird auch der Rückgabewert auf ungültig gesetzt.

Da der Rückgabewert immer eine Kopie ist, wird er in jedem Fall als freies Tupel betrachtet, das keinen Bezug zu einer Relation mehr aufweist. Darauf ist zu achten, wenn man die Funktion Rewrite() benutzt.

CorbaClose

Syntax: CorbaClose (String url, String rootname)
url: host:port
rootname: Name des Wurzelobjekts

Beispiel:
```
CorbaClose ("corba:4995", "PMSXRoot")
```

Beschreibung: Schließt die Verbindung zum Corba-Server, der unter *url* erreicht wurde, und gibt die damit verbundenen Ressourcen frei.

Siehe auch CorbaOpen().

CorbaOpen

Syntax: CorbaOpen (String url, String rootname) : String
url: host:port
rootname: Name des Wurzelobjekts

Beispiel:
```
id := CorbaOpen ("corba:4995", "PMSXRoot")
```

Beschreibung: Öffnet eine Verbindung mit einem Corba-Server, der unter *url* erreichbar ist und fragt beim dortigen Name-Service das Objekt *rootname* an.

Liefert den Zugriff zu dem angeforderten Objekt in Stringform.

Siehe auch CorbaClose().

Cos

Syntax: Cos (Real r) : Real
r:

Beispiel:
```
arg := Cos (winkel)
```

Beschreibung: Liefert den Cosinus des Winkels r im Bogenmaß.

Covarianz

Syntax: Covarianz (ZR zr1, ZR zr2, Intervall bereich) : Real
zr1:
zr2:
bereich: Auswertungszeitraum

Beispiel:
```
cov := Covarianz (zr1, zr2, bereich)
```

Beschreibung: Berechnet die Kovarianz der Zeitreihen $zr1$ und $zr2$ über *bereich*. Durch Angabe von *bereich* wird die Stichprobe festgelegt. Die Zeitreihen müssen äquidistante Intervall-Zeitreihen sein (diskrete Zufallsvariable) mit gleicher Intervallblockung. Siehe auch Mittel(), Median(), Varianz() und Korrelation().

CreateIndex

Syntax: CreateIndex (Relation R, String feldname)
R:
feldname: Name des Feldes, für das ein Schlüssel erzeugt werden soll

Beispiel:
```
CreateIndex (R, "Station")
```

Beschreibung: Erzeugt einen Index auf R nach dem feld $feldname$. Das Feld $feldname$ muss vom Typ String sein. Dies bewirkt, dass auf die Tupel nach Schlüssel zugegriffen werden kann (Siehe Search() und SearchNum().).

Wenn sich der Schlüssel über mehrere Felder erstrecken soll, dann werden die einzelnen Felder in $feldname$ mit + getrennt angegeben. Beispiel:

```
CreateIndex (R, "DBMSNR+DATVON")
tup := Search (R, "3001005+19902306")
```

Wenn schon ein Index existiert, wird er ersetzt.

Siehe auch CreateSortIndex().

CreateSem

Syntax: CreateSem (String semname, Real timeout) : Bool
semname: Name der Semaphore
timeout: Wartezeit in Sekunden

Beispiel: `CreateSem ("aquacall.sem", 20)`

Beschreibung: Prüft, ob eine Semaphore angelegt werden kann, und legt sie an.

Semaphoren werden benutzt, um geteilten Zugriff auf Ressourcen (z.B. dbf-Dateien) zwischen mehreren nebenläufigen Prozessen zu gewährleisten. Diese Prozesse können auf verschiedenen Rechnern laufen, daher ist die Semaphore eine Datei (und kein Eintrag im Speicher).

Wenn *semname* noch nicht existiert, wird sie angelegt. Die Überprüfung und das Anlegen werden in einem echt atomaren Schritt durchgeführt (mittels Hardlinks, also auch unter NFS lauffähig). So ist sichergestellt, dass zwischen Überprüfen und Anlegen kein anderer Prozess die Semaphore anlegen kann.

Falls *semname* schon existiert, ein anderer Prozess also Zugriff auf die Ressource beansprucht, wird gewartet, bis die Semaphore wieder gelöscht wird, höchstens aber *timeout* Sekunden. Dieses Timeout verhindert, dass die Prozesse unendlich lange aufeinander warten, wenn ein anderer Prozess stirbt, während er die Ressource benutzt (Lifelock des eigenen Prozesses).

Nach dem Aufruf ist die Semaphore angelegt, die Ressource also beansprucht.

Das Rückgabeergebnis ist in der Regel False, es sei denn, das Timeout wurde erreicht, dann ist es $True$.

Diese Funktion arbeitet nur korrekt, wenn Schreibrechte auf das aktuelle Verzeichnis existieren. Unter MS-Windows funktioniert diese Funktion nicht.

CreateSortIndex

Syntax: CreateSortIndex (Relation R, String feldname)
R: eine Memory-Relation
feldname: Name des Feldes, nach dem sortiert werden soll

Beispiel:
```
CreateSortIndex (R, "Station")
```

Beschreibung: Erzeugt einen Sortierindex auf R nach dem feld *feldname*. Das Feld *feldname* muss vom Typ String sein. Dies bewirkt, dass auf die Tupel sortiert zugegriffen werden kann (z.B. bei NewDBGrid()).

Wenn sich der Schlüssel über mehrere Felder erstrecken soll, dann werden die einzelnen Felder in *feldname* mit + getrennt angegeben. Beispiel:

```
CreateSortIndex (R, "DBMSNR+DATVON")
```

Wenn schon ein Sortierindex existiert, wird er ersetzt.

Die Sortierung erfolgt von kleinsten zum größten Wert. Soll anders herum sortiert werden, muss dem Feldnamen ein - vorangestellt werden. Beispiel:

```
CreateSortIndex (R, "-DBMSNR")
```

CreateTar

Syntax: CreateTar (Array dateien, String tardatei)
dateien: Liste der einzupackenden Dateien
tardatei: name der Ausgabedatei

Beispiel:
```
CreateTar (FileList("*.zpa"), "zpas.tar")
```

Beschreibung: Packt alle Dateien, deren Namen in *dateien* übergeben werden, im tar-Format in die Datei *tardatei*.

Verzeichnisse, die in *dateien* enthalten sind, werden rekursiv eingepackt. Sind in *tardatei* bereits Dateien enthalten, werden diese überschrieben.

Siehe auch ExtractTar().

CSVToTup

Syntax: CSVToTup (Tupel t, String csvdata, String trenner)
t: Tupel
csvdata: CSV-String
trenner: Trennzeichen

Beispiel:
```
CSVToTup (t, zeile, ";")
```

Beschreibung: Füllt das Tupel *t* mit Inhalten, die im CSV-Format vorliegen. Enthält das Tupel mehr Felder, als *csvdata*, bleiben die überschüssigen Felder unverändert. Hat *t* weniger Felder als *csvdata*, werden die überschüssigen Daten in *csvdata* ignoriert.

Siehe auch TupToCSV().

Dauerlinie

Syntax: Dauerlinie (ZR z, Intervall i, Bool b) : ZR
z:
i: Berechnungszeitraum
b: Temporärflag

Beispiel:
```
dl := Dauerlinie(z, i, b)
```

Beschreibung: Diese Funktion berechnet zu einer Zeitreihe die mathematisch korrekte Überschreitungs-Dauerlinie als weitere Zeitreihe. Ist b `FALSE`, dann bleibt diese im Datenpool als selbstständige Zeitreihe erhalten. Die Aussage wird auf Dau gesetzt. Wird die Zeitreihe in einem Aquagramm dargestellt, dann werden auf der X-Achse statt Zeitpunkten Zeitdistanzen erzeugt.

Die Berechnung ist sehr aufwändig, die Rechenzeit entsprechend groß. In anderen Systemen werden lediglich die Wertepaare nach Y sortiert. Dies ist aber **nur** zulässig, wenn es sich um äquidistante Intervall-Zeitreihen handelt. Bei kontinuierlichen Zeitreihen versagt eine Sortierung ganz.

Die Funktion Dauerlinie berechnet dagegen aus **kontinuierlichen** Zeitreihen **kontinuierliche** Dauerlinien. Dazu wird zu jeder linearen Strecke („Quant“) berechnet, ob und wie lange es einen bestimmten Y-Wert annimmt, diese Dauern werden addiert und später zur Dauerlinie zusammengefügt. Es genügt, als Y-Werte lediglich die Knickpunkte der Zeitreihe zu betrachten, da zwischen diesen die Zeitreihe, und damit auch die Dauer eines Schnittes mit einem Y-Wert, linear verläuft.

Siehe auch Dauertabelle und IntDauerlinien.

Dauertabelle

Syntax: Dauertabelle(ZR z, Real jahr, Intervall lang, String form, Bool alles, Bool strecken) : Relation
z: Intervall-Zeitreihe, Tageswerte
jahr: Wasserwirtschaftsjahr
lang: Jahre der langjährigen Reihe
form: Realformat “gesamt.nachkomma“ (z.B. “8.3“)
alle: alle Tage oder ausgewählte
strecken: soll bei lückenbehafteter Reihe gestreckt werden?

Beispiel:
```
tab := Dauertabelle(zr,1989,[@"1971",@"1990"],"5.1",FALSE)
```

Beschreibung: Berechnet Dauerzahlen und trägt sie in eine Relation (Tabelle) ein. Grundlage ist immer die Intervall-Zeitreihe *z*, welche Tageswerte enthalten muss, typischerweise Tagesmittelwerte (siehe IntervallMittel()). Natürlich kann die Intervall-Aussage beliebig sein. Die Reihe muss mindestens über dem Zeitintervall *lang* Daten enthalten. Berechnet werden Unterschreitungsdauern für *jahr*, sowie obere Grenzwerte, mittlere Werte und untere Grenzwerte für die langjährige Reihe über *lang*. Ist *alles* `TRUE`, dann wird für jeden Tag ein Wert ausgegeben, ansonsten nur für die üblichen (siehe DGJ-Blatt). Die langjährige Berechnung erfolgt genau über dem angegeben Intervall, also im Beispiel bis einschließlich dem Wasserwirtschaftsjahr 1989.

Standardmäßig beginnt ein Wasserwirtschaftsjahr am 1.11. Dies kann jedoch mit SetZPRaster() übersteuert werden.

Behandlung von Lücken:

Falls *strecken* `FALSE` ist: Ist die Zeitreihe im Jahr *jahr* lückenbehaftet, dann enthält die Jahresspalte in den oberen Zeilen leere Werte (0.0). Da zu Tagen mit Lücken **keine** Aussage gemacht werden kann, also auch keine Unterschreitungsentscheidung getroffen werden kann, sind die betreffenden Tage nicht in der Stichprobe enthalten, zu der die Dauerlinie erstellt wird. Die Dauerlinie ist demnach kürzer, es fehlen also die obersten Werte. Die langjährigen Spalten lassen sowohl bei der Extremwertbetrachtung als auch bei der Mittelwertbetrachtung Lückenwerte in der jeweiligen Dauerlinie außer Acht. Die Mittelwerte können demnach jeweils aus einer unterschiedlichen Anzahl von Werten gebildet worden sein.

Falls *strecken* `TRUE` ist: Die obige, statistisch richtige, Betrachtung wird außer Betracht gelassen. Die Dauerlinien werden jeweils auf ein ganzes Jahr gestreckt, **alle** Werte sind demnach unkorrekt und statistisch nicht mit anderen Tabellen vergleichbar. Lediglich die Form wird gewahrt.

DBAddRel

Syntax: DBAddRel (Datenbank db, Relation rel)
db: eine Datenbank
rel: Relation, die in *db* aufgenommen werden soll

Beispiel:
```
DBAddRel (db, rel)
```

Beschreibung: Nimmt die Relation *rel* in die Datenbank *db* auf.

Siehe auch DBGetRel().

DBDelRel

Syntax: DBDelRel (Datenbank db, String relname)
db: eine Datenbank
relname: Relationsname

Beispiel:
```
DBDelRel (db, "0mscode")
```

Beschreibung: Löscht die Relation mit dem Namen *relname* aus der Datenbank.

Siehe auch DBGetRel().

DBFilter

Syntax: DBFilter (Relation R, Tupel von [,Tupel bis]) : Relation
R:
von: (minimales) MusterTupel
bis: optional maximales MusterTupel

Beispiel:
```
Nstamm := DBFilter (stamm, muster)
```

Beschreibung: Filtert alle Tupel aus der Relation R und legt alle gefilterten Tupel in einer internen Relation ab.

Das Muster ist dabei entweder durch ein Mustertupel (*von*) oder durch zwei Mustertupel (*von* und *bis*) gegeben.

Wird nur *von* angegeben, dann werden die Tupel der Relation R jeweils auf Gleichheit getestet. Wildcards, wie z.B. `*hausen`, sind erlaubt. Unbesetzte Felder werden wie * behandelt.

Bei Angabe von *bis* wird ein Bereichsvergleich durchgeführt. Alle Tupel, die (lexigraphisch oder numerisch) zwischen *bis* und *von* liegen, matchen. Es findet also der Vergleich: $von <= t <= bis$ statt.

Statt ein *bis*-Tupel zu übergeben, kann man für einzelne Felder auch Ober- oder Untergrenzen im Feldinhalt übergeben. Dazu muss im Feld an der ersten Stelle ein < bzw. > stehen. Die Grenzen sind mit enthalten (also gilt <= bzw. >=).

Die Ergebis-Relation erbt den Namen der Ausgangsrelation.

Siehe auch DBInvFilter().

DBFlush

Syntax: DBFlush(Relation R)
R:

Beispiel:
```
DBFlush (stamm)
```

Beschreibung: Macht alle Änderungen in der Relation R dauerhaft, leert also alle Zwischenbuffer. Dies ist nötig, wenn auf eine Relation zugegriffen werden soll, **während** diese sich in einem Azurprogramm befindet, welches noch nicht verlassen wurde.

Siehe auch Relation().

DBGetRel

Syntax: DBGetRel (Datenbank db, String name) : Relation
db: eine Datenbank
name: Name der relation

Beispiel:
```
rel := DBGetRel (db, "stamnied")
```

Beschreibung: Holt die Relation mit dem Namen *name* aus der Datenbank *db*. Ist keine Relation mit diesem Namen vorhanden, wird eine ungültige Relation geliefert (siehe IsValid()).

Siehe auch DBAddRel().

DBInfo

Syntax: DBInfo (Datenbank db) : Relation
db: eine Datenbank

Beispiel:
```
rel := DBInfo (db)
```

Beschreibung: Erzeugt eine Relation, die zu jeder Relation in *db* ein Tupel enthält. Die Tupel haben die Struktur `Name#28S,AnzTup#10N,Struktur#S`. Diese Felder enthalten den Namen der Relation, die Anzahl Tupel und die Struktur.

Siehe auch DBGetRel().

DBInvFilter

Syntax: DBInvFilter (Relation R, Tupel von [,Tupel bis]) : Relation
R:
von: (minimales) MusterTupel
bis: optional: maximales MusterTupel

Beispiel:
```
Nstamm := DBInvFilter (stamm, muster)
```

Beschreibung: Filtert alle Tupel aus der Relation R und legt alle Tupel, die **nicht** auf das Muster passen in einer internen Relation ab.

Siehe DBFilter().

DBRead

Syntax: DBRead (String filename) : Datenbank
filename: Name der Datenbankdatei

Beispiel:
```
db := DBRead ("nrw")
```

Beschreibung: Liest eine Datenbank aus der Datei $filename$ aus. Wenn die Endung .rdb nicht angegeben wird, so wird sie ergänzt.

Siehe auch DBWrite() und NewDatenbank().

DBWrite

Syntax: DBWrite (Datenbank DB, String filename)
DB: eine Datenbank
filename: Name der Datenbankdatei

Beispiel:
```
DBWrite (db, "nrw")
```

Beschreibung: Schreibt die Datenbank in eine Datenbankdatei. Ist die Endung .rdb nicht angegeben, wird sie ergänzt.

Siehe auch NewDatenbank() und DBAddRel().

DefArt

Syntax: DefArt (ZR z) : String
z:

Beispiel:
```
art := DefArt( z )
```

Beschreibung: Gibt die Definitionsart der Zeitreihe zurück. K für kontinuierliche Zeitreihen, I für Intervall-Zeitreihen und M für Momentan-Zeitreihen.

DelAGElement

Syntax: DelAGElement(String name)
name: Name des aqua_gramm-Elements

Beispiel:
```
DelAGElement( "LosButton" )
```

Beschreibung: Löscht ein graphisches Element des aktuellen AGWindows. Es können auch mehrere Elemente gleichzeitig gelöscht werden. Dazu übergibt man die Namen mit Leerzeichen getrennt. Siehe auch ImportVar() und ExportVar().

DelAGWindow

Syntax: DelAGWindow(String name)
name: Name des AGWindows

Beispiel:
```
DelAGWindow( "AGPop2" )
```

Beschreibung: Löscht das AGWindow mit dem Namen *name*. Das aktuelle AGWindow und das AGWindow AG0 dürfen nicht gelöscht werden.

Siehe auch SetAGWindow(), GetAGWindow() und NewAGWindow().

DelAllTupels

Syntax: DelAllTupels (Relation R)
R:

Beispiel: `DelAllTupels (filterrel)`

Beschreibung: Löscht alle Tupel aus der Relation R.

Siehe auch DelTupel().

DelCanvas

Syntax: DelCanvas()

Beispiel: `DelCanvas()`

Beschreibung: Löscht den Canvas des aktuellen AGWindows. Die Scroll- und Zoom-Buttons werden ebenfalls gelöscht.

Siehe auch NewCanvas().

DeleteModRecords

Syntax: DeleteModRecords (ZR z)
z:

Beispiel: `DeleteModRecords (zr1)`

Beschreibung: Löscht alle in z eingetragenen Änderungseinträge.

Siehe auch ModifiedSince().

DelQuality

Syntax: DelQuality (ZR z, Intervall bereich, Real quality)
z:
bereich: Zeitbereich, auf dem gelöscht werden soll
quality: zu löschende Qualität

Beispiel: `DelQuality(zr, WWJ(1989), 3)`

Beschreibung: Löscht die Werte im angegebenen Bereich der Qualität *quality*. Die Texte werden auch gelöscht.

Dies hat zur Folge, dass nun die Werte der unteren Qualität durchscheinen. Für den Benutzer ergibt sich demnach, dass die Werte der Qualität *quality* durch die Werte der Qualität *quality* − 1 ersetzt werden.

Fallen die Grenzen von *bereich* nicht genau auf vorhandene Stützstellen, dann wird der zu löschende Bereich erweitert, sodass er vom letzten Stützpunkt vor dem Bereich bis zum ersten Stützpunkt nach dem Bereich geht. Dadurch verändert sich ggf. der Verlauf der Randquante. **Achtung: dies war bis Juni 2005 anders, es wurden Werte am Rand interpoliert!**

Werden dadurch **alle** Werte der Qualität gelöscht, so wird die Qualität aus der Zeitreihe herausgenommen. Wird die höchste Qualität gelöscht, wird demnach MaxQualitaet() geändert.

Soll explizit die gesamte Qualität gelöscht werden, so ist als *bereich* die Konstante MAXFOCUS anzugeben.

Das Löschen der Qualität 0 stellt einen Sonderfall dar: Da es keine darunter liegende Qualität gibt, können auch keine Werte von unten durchscheinen (siehe oben). Stattdessen werden alle Werte in *bereich* entfernt und durch eine Lücke ersetzt. Die Qualität 0 wird jedoch nie ganz entfernt, selbst wenn alle Werte gelöscht würden.

Mit DelTextQuality() können Texte auch gesondert gelöscht werden.

DelQuant

Syntax: DelQuant (QuantList ql, R nr)
ql: eine Quantenliste
nr: Reihenfolgenummer des Quants

Beispiel: `DelQuant (ql, 0)`

Beschreibung: Löscht das Quant mit der Laufnummer *nr* aus der Quantenliste. Das erste Quant hat die Nummer 0.

Siehe auch AppendQuant(), PrependQuant() und QuantNr().

DelSelected

Syntax: DelSelected (Relation R)
R: Memory-Relation

Beispiel: `DelSelected (stamm)`

Beschreibung: Löscht alle selektierten Tupel aus *R*.

Siehe auch SelectAll() und TupSelect().

DeltaZR

Syntax: DeltaZR(ZR z, Intervall i, Distanz d, String p, Bool tmp) : ZR
z:
i: Berechnungszeitraum
d: Breite der Intervalle
p: Parameter der Ergebniszeitreihe
tmp: Temporärflag

Beispiel: `zr2 := DeltaZR(z, WWJ(1982), ~"24 Stunden", "Fracht", FALSE)`

Beschreibung: DeltaZR erzeugt eine Intervall-Zeitreihe, mit Intervallbreite *d*. Diese Zeitreihe erbt alle Attribute von *zr* außer dem Parameter, der auf *p* gesetzt wird. Der Y-Wert eines Intervalles ergibt sich aus der Differenz der Werte in *zr* an der rechten und der linken Intervallgrenze (rechts-links).

Diese Funktion sollte mit Bedacht verwendet werden. In der Regel macht ihre Anwendung nur Sinn bei Zeitreihen, die ihrem Wesen nach Summenlinien sind (z.B. Gewicht eines Lysimetertopfes). Hat eine Zeitreihe die Aussage Sum, dann sollte man nicht DeltaZR() benutzen, sondern stattdessen auf die Ursprungszeitreihe (aus der die Summen-ZR hervorgegangen ist) die Funktion SummenWerte() anwenden, was das gleiche Ergebnis liefert.

DelTextQuality

Syntax: DelTextQuality (ZR z, Intervall bereich, Real quality)
z:
bereich: Zeitbereich, auf dem gelöscht werden soll
quality: zu löschende Qualität

Beispiel: `DelQuality(zr, WWJ(1989), 3)`

Beschreibung: Löscht die Texte im angegebenen Bereich der Qualität *quality*.

Der Algorithmus ist in DelQuality() beschrieben.

DelTupel

Syntax: DelTupel(Relation R, Tupel t)
R:
t: Tupel

Beispiel: `DelTupel( Stamm, t )`

Beschreibung: Löscht das Tupel t aus der Relation R.

Ist R eine dbf-Relation, dann wird das Tupel nur auf ungültig gesetzt. Die so als gelöscht markierten Tupel können mit dem externen Programm [**?**] entfernt werden.

Siehe auch AppTupel().

DirList

Syntax: DirList (String pattern) : Array
pattern: Dirnamenmuster

Beispiel: `liste := DirList ("m*")`

Beschreibung: Liefert ein Array, das alle Verzeichnisse enthält, die auf das Muster *pattern* passen. Wird, kein Verzeichnisname mit angegeben, so werden die Verzeichnisse im aktuellen Verzeichnis durchsucht. Auf die Verzeichnisnamen im Rückgabearray kann z.B. mittels des FORALL-Operators zugegriffen werden.

Wichtig: Wenn *pattern* lediglich den Namen eines Verzeichnisses enthält (z.B. `liste := DirList("/aquaplan")`), werden die Unterverzeichnisse nicht mit ausgegeben, da lediglich das Verzeichnis `/aquaplan` selbst auf das Muster `/aquaplan` passt. Wenn man alle Unterverzeichnisse im Verzeichnis erhalten möchte, so muss man `liste := DirList("/aquaplan/*")` angeben.

Siehe auch FileList().

DistMaxima

Syntax: DistMaxima (ZR z, Intervall i, Distanz d, Bool temp) : ZR
z: Ausgangszeitreihe
i: Berechnungszeitraum
d: Breite der Intervalle, für die die Berechnung stattfindet
temp: Temporärflag

Beispiel: `momzr := DistMaxima (zr, bereich, ~"1 Tag", False)`

Beschreibung: Erzeugt eine Momentanzeitreihe, die auf dem Berechnungszeitraum i für jedes Intervall der Breite d das Maximum mit dem zugehörigen Zeitpunkt enthält.

Damit sichergestellt ist, dass es pro Intervall genau einen Wert gibt, werden Maxima, die genau auf das Intervallende fallen, um 5 Sekunden nach links verschoben, damit sie nicht mit dem Maximum des jeweils nächsten Intervalls in Konflikt geraten.

Die Minima lassen sich mit DistMinima() berechnen.

Siehe auch IntervallMax() und LokaleMaxima().

DistMinima

Syntax: DistMinima (ZR z, Intervall i, Distanz d, Bool temp) : ZR
z: Ausgangszeitreihe
i: Berechnungszeitraum
d: Breite der Intervalle, für die die Berechnung stattfindet
temp: Temporärflag

Beispiel:
```
momzr := DistMinima (zr, bereich, ~"1 Tag", False)
```

Beschreibung: Erzeugt eine Momentanzeitreihe, die auf dem Berechnungszeitraum i für jedes Intervall der Breite d das Minimum mit dem zugehörigen Zeitpunkt enthält.

Damit sichergestellt ist, dass es pro Intervall genau einen Wert gibt, werden Minima, die genau auf das Intervallende fallen, um 5 Sekunden nach links verschoben, damit sie nicht mit dem Minimum des jeweils nächsten Intervalls in Konflikt geraten.

Die Maxima lassen sich mit DistMaxima() berechnen.

Siehe auch IntervallMin() und LokaleMinima().

DistStr

Syntax: DistStr (Distanz d) : String
d: Distanz

Beispiel:
```
a := "Breite: " + DistStr(d)
```

Beschreibung: Liefert einen String, der die Distanz als lesbaren Text enthält. Dieser Text enthält ganze Anteile der Felder Jahr, Monat, Tag, Stunde, Minute und Sekunde (z.B. 6h 3min 25s). Im Gegensatz zu Str() kann dieser Text nicht mittels ~ in eine Distanz zurückverwandelt werden.

DistStunden

Syntax: DistStunden (Distanz d) : Real
d: Distanz

Beispiel: `r := DistStunden(d)`

Beschreibung: Liefert die Breite der Distanz *d* in Stunden zurück. **Zu beachten** ist, dass eine Distanz natürlich im Allgemeinen keine feste Länge hat, sondern sich erst bei der Addition oder Subtraktion mit einem Zeitpunkt eine Länge ergibt. Den meisten Distanzen können dennoch Längen zugewiesen werden (ein Tag hat immer 24 Stunden). Lediglich die Distanz Jahr=8766h und Monat=720h werden besonders behandelt.

DivGroesse

Syntax: DivGroesse (ZR z, String sg, Intervall i, String sp, Bool b) : ZR
z: eine Zeitreihe
sg: Größe
i: Berechnungszeitraum
sp: Parameter der Ergebniszeitreihe
b: [**?**]

Beispiel: `diff := DivGroesse (zr1, "42.3km^2", MAXFOCUS, "Hoehe", TRUE)`

Beschreibung: Teilt die Zeitreihe durch die Größe *sg*, welche eine einheitenbehaftete Zahl ist. Die Ergebniszeitreihe erhält *sp* als Parameter. Die Verknüpfung der Einheiten findet automatisch statt.

DivZR

Syntax: DivZR (ZR z1, ZR z2, Intervall i, String param, Bool b) : ZR
z1:
z2:
i: Berechnungszeitraum
param: Parameter der Ergebniszeitreihe
b: [**?**]

Beispiel:
```
diff := DivZR( z1, z2, i, s, b )
```

Beschreibung: Teilt die Zeitreihe $z1$ durch die Zeitreihe $z2$. Die Divison erfolgt auch bei kontinuierlichen Zeitreihen mathematisch korrekt, siehe dazu auch [**?**]. Die Berechnung der Ergebniseinheit findet automatisch statt.

DoAt

Syntax: DoAt (Real millis, String funktion)
millis: wieviel Millisekunden gewartet werden soll
funktion: Azurfunktion, die ausgeführt werden soll

Beispiel:
```
DoAt (3000, "MachWas")
```

Beschreibung: Führt die Azurfunktion *funktion* zeitversetzt aus. Die Zeit, die bis zur Ausführung verstreichen soll, wird mit *millis* in Millisekunden angegeben. Die erreichbare Genauigkeit dieser Angabe hängt von der Rechenleistung der jeweiligen CPU ab.

funktion wird so aufgerufen, als hätte der Benutzer zur entsprechenden Zeit im aktuellen Window einen Button namens `@timer-`*millis* gedrückt.

DoModRecording

Syntax: DoModRecording (ZR z, Bool b)
z:
b: True oder False

Beispiel:
```
DoModRecording(zr1, False)
```

Beschreibung: Legt fest, ob beim Schreiben in die Zeitreihe z aufgezeichnet wird, welche Monate (für aktuelle Werte auch Tage) verändert wurden. Die Voreinstellung ist, dass es aufgezeichnet wird.

Dieses Attribut wird nicht permanent in der Zeitreihe gespeichert.

Siehe auch ModifiedSince() und DeleteModRecords().

DrawBoxOnPage

Syntax: DrawBoxOnPage (Page page, GeoPoint p1, GeoPoint p2, String farbe)
page:
p1: Unterer, linker Punkt der Box in cm
p2: Oberer, rechter Punkt der Box in cm
farbe: Farbe der Fläche (siehe ZRInAxBox()

Beispiel:
```
DrawBoxOnPage (page, {4.5,6}, {7.5,6}, "Blau")
```

Beschreibung: Zeichnet eine gefüllte Box auf die Seite. Die Angabe der Koordinaten erfolgt in cm. Koordinaten, die als Spalte, Zeile vorliegen, können mit PagePos() umgerechnet werden.

Siehe auch NewPage() und DrawLineOnPage().

DrawLineOnPage

Syntax: DrawLineOnPage(Page page, GeoPoint p1, GeoPoint p2, Real width)
page:
p1: Startpunkt der Linie in cm
p2: Endpunkt der Linie in cm
width: Breite der Linie in cm

Beispiel:
```
DrawLineOnPage(page, {4.5,6}, {7.5,6}, 0.01)
```

Beschreibung: Zeichnet eine Linie auf die Seite. Die Angabe der Koordinaten erfolgt in cm. Koordinaten, die als Spalte, Zeile vorliegen, können mit PagePos() umgerechnet werden.

Siehe auch NewPage() und DrawTextOnPage().

DrawSymOnPage

Syntax: DrawSymOnPage (Page page, GeoPoint p, Real was, Real size)
page:
p: Lage des Symbols
was: Symboltyp
size: Höhe des Symbols cm

Beispiel:
```
DrawSymOnPage(page, {4.5,6}, 5, 0.3)
```

Beschreibung: Zeichnet eine Symbol auf die Seite. Die Angabe der Koordinaten erfolgt in cm. Das Symbol wird um p zentriert ausgegeben. Koordinaten, die als Spalte, Zeile vorliegen, können mit PagePos() umgerechnet werden.

Die Symboltypen sind in SetSymbolTyp() beschrieben.

Siehe auch DrawTextOnPage() und SetPageColor().

DrawTextOnPage

Syntax: DrawTextOnPage(Page page, GeoPoint wo, String text, Real style, [Real winkel — String ausricht])
page:
wo: Koordinate in cm
text:
style: Siehe TextOnPage()
winkel: optional. 0 bedeutet waagerecht. oder:
ausricht: siehe dazu TextOnPage()

Beispiel:
```
DrawTextOnPage( page, {3.5,7.89}, "Haupttabelle", BOLD+UNDERLINE )
```

Beschreibung: Setzt einen Text an der angegebenen Position auf die Seite. Die linke untere Ecke hat die Koordinate (0,0). Siehe auch TextOnPage(), DrawLineOnPage() und PlotTextOnPage()

DruckerWahl

Syntax: DruckerWahl()

Beispiel:
```
DruckerWahl()
```

Beschreibung: Öffnet (nicht unter Unix) ein Dialogfenster, mit dem der Drucker gewählt werden kann, auf den die folgenden Ausgaben erfolgen.

Siehe auch PrintPage().

Einheit

Syntax: Einheit (ZR z) : String
z:

Beispiel:
```
einh := Einheit( z )
```

Beschreibung: Liefert das Attribut Einheit der Zeitreihe.

ElementBox

Syntax: ElementBox (String frage, String elemtyp, ...) : String
frage: beliebiger Text
elemtyp: Typ des AG-Elements

Beispiel:
```
dateiname := ElementBox("Name", "Eingabe", 100, "out.dat")
```

Beschreibung: Erzeugt ein neues Fenster, welches den Text *frage*, das angegebene Element, einen OK-Button und einen Abbruch-Button enthält.

Das AGWindow, aus dem die Element-Box gestartet wurde, ist solange inaktiv, bis der Benutzer den OK-Button gedrückt oder das Element aktiviert hat. Der Programmablauf wird solange angehalten.

Die möglichen Werte für *elemtyp* ergeben sich aus den möglichen AG-Elementen (z.B. Eingabe, Auswahl, Slider usw.).

Die darauf folgenden Parameter entsprechen denen der jeweiligen Azurfunktion (z.B. NewEingabe()), jeweils ohne die Koordinaten, den Namen und das Azurprogramm.

Der Rückgabewert ist der Inhalt des jeweiligen AG-Elements. Falls der Abbruch-Button gedrückt wurde, wird ein Leerstring zurück gegeben.

Siehe auch OkBox(), SelectBox(), InputBox() und MultiBox().

EndOfFile

Syntax: EndOfFile ([String filename]) : Bool
filename: Name der Datei

Beispiel:
```
WHILE (NOT EndOfFile( "daten.txt" ))
```

Beschreibung: Gibt an, ob in der Datei *filename* noch Zeilen folgen. Dieses Flag wird **nach** dem Lesen der letzten Zeile auf TRUE gesetzt. Wird der Parameter *filename* weggelassen, so wird die Standardeingabe abgefragt (Ctrl-D über die Tastatur oder Ende der Pipe). Siehe auch ReadLine(), Rewind() und FileForward().

ExistsAGWindow

Syntax: ExistsAGWindow(String name) : Bool
name: Name des AGWindows

Beispiel:
```
IF (ExistsAGWindow("AGPop2"))
```

Beschreibung: Testet, ob das AGWindow *name* existiert.

Siehe auch AGToTop().

ExistsFile

Syntax: ExistsFile (String filename) : Bool
filename: Name der Datei

Beispiel:
```
IF (ExistsFile( "daten.txt" ))
```

Beschreibung: Gibt an, ob die Datei $filename$ existiert. $filename$ kann dabei auch ein Verzeichnis sein. Siehe auch FileLesbar(), EndOfFile(), ReadLine() und Rewind().

Export

Syntax: Export(ZR z, Intervall i, Real qualy, String format, String outdatei [, String neuort [, String neuparam [, String info]]])
z:
i: Ausgabeintervall
qualy: Qualität, die exportiert werden soll
format: Format der Exportdatei
outdatei: Name der Exportdatei
neuort: optional: Neuer Ort
neuparam: optional: Neuer Parameter
info: optional: Zusatzinformation zum Exportieren

Beispiel:
```
Export (zr1, bereich, 0, "DVWK", "datei.dwd")
```

Beschreibung: Exportiert die Zeitreihe in ein Fremdformat. Mögliche Formate sind ASCII, DVWK, NASIMBLOCK, NASIMXY, UVF, LWAFLUT, NRT, LWAPEGEL, STAWANTAGE, GEOTRON, ED, MD und SMUSI. Die Parameter *neuort* und *neuparam* können von rechts weggelassen werden. Sie übersteuern die Attribute Ort und Parameter der Reihe z. Dies findet vor allem bei der Bereitstellung von Eingabedaten für Simulationssysteme mit Einfach-Zeitreihenverwaltung Anwendung. Der Parameter $info$ dient dazu, Zusatzinformation an ein Exportformat zu übergeben. Zurzeit ist Folgendes wählbar:

- ASCII-Format: FAST, für schnellen Datenaustausch
- ASCII-Format: NOTEXT, die Ausgabe von Textwerten wird unterdrückt
- UVF-Format: i=links, bei Intervallwerten wird der Zeitpunkt der linken Grenze ausgegeben.
- UVF-Format: LUAEXPORT, das Format wird in einer speziellen Variante für das LUA erzeugt.
- UVF-Format: APPEND, die Datei wird nicht neu geschrieben, sondern die Daten werden angehängt.

Will man nur $info$, ohne *neuort* und *neuparam*, angeben, übergibt man Leerstrings:
```
Export (zr1, bereich, 0, "UVF", "datei.uvf", "", "", "APPEND")
```

Siehe auch ExportListe().

ExportDBF

Syntax: ExportDBF(ZR z, Intervall i, Real qualy, Real maxtxt, Bool mitzeit, String outrel)
z:
i: Ausgabeintervall
qualy: Qualität, die exportiert werden soll
maxtxt: Breite des Textfeldes (0=keins)
mitzeit: Soll ein Feld für Stunde-Minute-Sekunde erzeugt werden?
outrel: Name der Relation

Beispiel: `ExportDBF(zr1, bereich, MAXQUAL, 20, TRUE, "jahr89")`

Beschreibung: Exportiert die Zeitreihe z auf dem Bereich i in der Qualität *qualy* in die DBF-Datei *outrel*.dbf. Erzeugt werden die Felder dat, welches Tag-Monat-Jahr eines Zeitpunkts enthält, zeit, das Stunde-Minute-Sekunde eines Zeitpunkts enthält (ss:mm:ss) und wert, in dem der Wert zum Zeitpunkt als String gespeichert wird. Wenn *maxtxt* einen positiven Wert enthält, dann wird ein Feld text mit der Breite *maxtxt* erzeugt. Falls *mitzeit* `FALSE` ist, dann wird das Feld zeit nicht erzeugt.

Für das Beispiel lautete das Format: `dat#D,zeit#8S,wert#11S,text#20S`. Das Feld Wert ist ein Textfeld, weil es auch den Text `Luecke` enthalten können muss. Es ist 11 Zeichen breit, da eine Zahl mit 8 Stellen Genauigkeit inklusive Dezimalpunkt, führender 0 und Minuszeichen hineinpassen muss.

Da offensichtlich lediglich Wertepaare gespeichert werden, ist der Benutzer selbst dafür verantworlich, welcher Zeitreihe und Qualität die Werte zuzuordnen sind.

Dieses Format ist also **kein** Zeitreihen-Export-Format, sondern nur ein Wertepaar-Export-Format. Es kann dazu benutzt werden, einen Wertepaar-Editor zu erstellen.

Siehe auch ImportDBF().

ExportListe

Syntax: ExportListe(ZRList zrl, Real qual, S outdatei, S format, S info)
zrl: Reihenliste
qual: Qualität, die exportiert werden soll
outdatei: Name der Exportdatei
format: Ascii oder UVF
info: Zusatzinformation zum Exportieren

Beispiel:
```
ExportListe(zrl, 0, "datei.asc", "Ascii", "")
```

Beschreibung: Exportiert alle Zeitreihen aus *zrl* in die Datei *outdatei*. Der zu exportierende Bereich wird dem Fokus von *zrl* entnommen, siehe dazu SetFocus() und GetFocus().

Mit *format* legt man fest, ob die Ausgabe im AQZ-Ascii-Listenformat, im UVF-Format oder NRT-Format erfolgen soll.

Taucht in *info* der Begriff `APPEND` auf, so wird die Datei nicht neu erzeugt, sondern die Daten an die bestehende Datei angehängt.

Siehe auch Export().

ExportQF

Syntax: ExportQF (ZR zr, ZI focus, B ascii, S einh, S release) : String
zr: eine Zeitreihe
focus: der zu exportierende Bereich ascii: True=ASCII-Format, sonst binär
einh: die Einheit der Werte
release: Version des Datenformats, sollte 1 sein.

Beispiel:
```
data := ExportQF (zr, mf, False, "cm", "1")
```

Beschreibung: Wandelt *zr* in ein Format (ASCII oder binär), das mittels WebPut() an einen TSTP-Server verschickt werden kann.

Siehe auch ImportQF().

ExportVar

Syntax: ExportVar(String name, (...) par)
name: Name des Aquagramm-Elements
par: Inhalt mit passendem Typ

Beispiel: `ExportVar( "Variable", 12345 )`

Beschreibung: Mit Hilfe dieser Prozedur kann der Inhalt von Aquagramm-Elementen gesetzt werden, z.B. von Texten oder Eingaben.

Was das Setzen im Einzelfall genau bewirkt, ist für die jeweiligen Elemente in der entsprechenden New..-Funktion beschrieben (z.B. NewListe()).

ExpWert

Syntax: ExpWert (ZR zr, Real exp, Intervall i, String param, Bool b) : ZR
zr:
exp: Exponent
i: Berechnungszeitraum
param: Parameter der Ergebniszeitreihe
b: Temporärflag

Beispiel: `mul := ExpWert (z1, 2/3, i, s, b)`

Beschreibung: Potenziert alle Werte in *zr* auf dem Bereich *i* mit *exp* und legt das Ergebnis als Zeitreihe mit dem neuen Parameter *param* ab.

Die Berechnung erfolgt Wert für Wert. Die Zwischenbereiche werden **nicht** entsprechend der FToleranz() angepasst.

Die Werte der Zeitreihe dürfen **nicht negativ** sein. Für negative Werte wird ersatzweise 0 eingesetzt.

Siehe auch MulGroesse() und MulZR().

ExtractTar

Syntax: ExtractTar (String tardatei, String outdir) : Array
tardatei: tar-Archiv
outdir: Ausgabeverzeichnis

Beispiel:
```
dateien := ExtractTar ("zpas.tar", "")
```

Beschreibung: Packt alle Dateien aus dem tar-Archiv *tardatei* aus. Wenn *outdir* leer ist, werden die Dateien in das aktuelle Verzeichnis geschrieben, sonst in *outdir*.

Die Liste der ausgepackten Dateien wird zurückgegeben.

Siehe auch CreateTar().

ExtrahiereEreignis

Syntax: ExtrahiereEreignis (ZR zr, ZI ergbereich, Real quantilbreite) : QuantList
zr: Intensitätszeitreihe (z.B. Niederschlag)
ergbereich: Zeitspanne des Ereignisses
quantilbreite: Breite eines Quantils in %

Beispiel:
```
qf := ExtrahiereEreignis (nzr, bereich, 10)
```

Beschreibung: Erzeugt Quantile für ein Ereignis. Quantile sind Zeitpunkte des Erreichens einer Teilsumme. Das 0%-Quantil bezeichnet also den Ereignisbeginn und das 100%-Quantil das Ereignisende. Das 50%-Quantil wird auch Median (Zentralwert) genannt.

Der Bereich *ergbereich* kann vorher mit der Funktion FindeEreignis() ermittelt werden.

Das Ergebnis ist eine Quantliste, die je Quantil ein Quant enthält. Die rechte Seite des XBereich() jedes Quants enthält den Zeitpunkt, der angibt, wann die entsprechende Teilsumme erreicht ist. Diese Summe ist pro Quant als Y-Wert (YRechts()) abgelegt. Das erste Quant ist das 0%-Quantil, das letzte das 100%-Quantil.

Wenn die Eingangsdaten nicht korrekt waren oder aus einem anderen Grund keine Quantile zu ermitteln waren, so wird eine leere Quantliste zurückgeliefert (siehe AnzQuanten()). Siehe SynthetisiereEreignis() und NextLuecke().

FieldModified

Syntax: FieldModified (Tupel t, String feld) : Bool
t: Tupel
feld: Feldname

Beispiel: `IF (FieldModified(tup, "ORT"))`

Beschreibung: Zeigt an, ob das Feld *feld* des Tupels *t* seit dem letzten ClearModify() bzw. der Erzeugung des Tupels verändert wurde.

Es ist nicht möglich, den Modified-Zustand einzelner Felder zu löschen. Es ist aber möglich, mit ClearModify() das Tupel inkl. aller Felder zurückzusetzen.

Siehe auch RelModified() und IsModified().

FileCopy

Syntax: FileCopy (String quelle, String ziel)
quelle: name der Quelldatei
ziel: name der Zieldatei

Beispiel: `FileCopy ("mde.prc", palmtausch)`

Beschreibung: Kopiert die Datei *quelle* in die Datei *ziel*. Beide Dateinamen können ggf. einen Pfad enthalten.

Ist Quelle nicht vorhanden oder nicht lesbar, ist das Verzeichnis von *ziel* nicht vorhanden oder ist *ziel* nicht schreibbar, wird eine entsprechende Fehlermeldung ausgegeben.

Siehe auch ReadFile() und Rename().

FileDate

Syntax: FileDate(String filename) : Real
filename: Name der Datei

Beispiel:
```
secs := FileDate( "daten.txt" )
```

Beschreibung: Gibt das Datum der letzten Veränderung von *filename* zurück. Da dieses auf die Sekunde genau sein soll, kommt der Rückgabetyp Zeitpunkt nicht in Betracht. Stattdessen wird die Anzahl der Sekunden seit dem 1.1.1970 0:00 (bezogen auf UTC (GMT)) geliefert. Wenn getestet werden soll, ob eine Datei neuer als eine andere ist, oder ob eine Datei sich seit dem letzten Zugriff verändert hat, ist diese Funktion völlig ausreichend.

FileDate bietet sich besonders bei der Programmierung von Multi-User-Anwendungen an.

Siehe auch ExistsFile().

FileForward

Syntax: FileForward (String filename)
filename: Name der Datei

Beispiel:
```
FileForward ("daten.txt")
```

Beschreibung: Setzt den Lesekopf der Datei *filename* hinter die letzte Zeile. Dies ist nützlich, wenn man in *filename* neu eintreffende Daten erwartet.

Siehe auch Rewind(), ReadLine() und EndOfFile().

FileLesbar

Syntax: FileLesbar (String filename) : Bool
filename: Name der Datei

Beispiel:
```
IF (FileLesbar( "daten.txt" ))
```

Beschreibung: Gibt an, ob die Datei *filename* lesbar ist. Siehe auch FileSchreibbar() und ExistsFile().

FileList

Syntax: FileList (String pattern) : Array
pattern: Filenamenmuster

Beispiel:
```
liste := FileList ("*.dat")
```

Beschreibung: Liefert ein Array, das alle Dateien enthält, die auf das Muster *pattern* passen. Wird, wie im Beispiel, kein Verzeichnisname mit angegeben, so werden die Dateien im aktuellen Verzeichnis gesucht. Auf die Dateinamen im Rückgabearray kann z.B. mittels des FORALL-Operators zugegriffen werden.

Siehe auch DirList().

FileSchreibbar

Syntax: FileSchreibbar (String filename) : Bool
filename: Name der Datei

Beispiel:
```
IF (FileSchreibbar( "ausgang/logs/daten.txt" ))
```

Beschreibung: Gibt an, ob die Datei *filename* schreibbar ist. Falls die Datei nicht vorhanden ist, wird zurückgegeben, ob man die Datei anlegen könnte.

Siehe auch FileLesbar() und ExistsFile().

FileSize

Syntax: FileSize (String filename) : Real
filename: Name der Datei

Beispiel:
```
 len := FileSize ("daten.txt"))
```

Beschreibung: Ermittelt die Länge einer Datei in Byte. Existiert die Datei nicht, wird -1 geliefert.

Siehe auch FileLesbar() und ExistsFile().

FindeEreignis

Syntax: FindeEreignis (ZR zr, ZI bereich, Real bsum, ZD bdist, R esum, Z edist, R minintens) : ZI
zr: Intensitätszeitreihe (z.B. Niederschlag)
bereich: Auswertungszeitraum
bsum: minimale Summe für Ereignisstart
bdist: Distanz, über der bsum vorliegen muss
esum: maximale Summe für Ereignisende
edist: Distanz, über der esum vorliegen muss
minintens: minimale Intensität (Rauschfilter)

Beispiel:
```
zi := FindeEreignis (nzr, bereich, 0.3, ~"1h", 0.05, ~"1h", 0.01)
```

Beschreibung: Findet in der Reihe *zr* auf dem Zeitintervall *bereich* ein Ereignis.

Ein Ereignis beginnt, wenn die Summe über einer Distanz *bdist* mindestens *bsum* beträgt (z.B. in einer Stunde mindestens 0,3 mm Niederschlag fällt). Entsprechend hört es auf, wenn die Summe über *edist* nicht mehr als *esum* beträgt.

minintens muss angegeben werden, damit ein Grundrauschen in der Reihe unterdrückt wird. Dieses ist durch die Genauigkeit der Datenaufnahme bestimmt. Für Niederschlagszeitreihen beträgt es meist 0.01 mm/h.

Wird kein Ereignis gefunden, so wird ein ungültiges Intervall (siehe IsValid()) zurückgeliefert.

Siehe ExtrahiereEreignis(), SynthetisiereEreignis() und NextLuecke().

FindPoly

Syntax: FindPoly(Karte K, GeoPoint p) : Polygon
K: eine Karte

Beispiel:
```
MyKlickFunc (Karte AktMap, GeoPoint AktPoint)
   messstelle := FindPoly (AktMap, AktPoint)
END
```

Beschreibung: Alle aktiven (`@Active`) Layer der Karte K werden durchsucht.

Zuerst werden alle Einpunkt-Polygone (z.B. Messstellen) durchsucht. Liegt ein Polygon innerhalb des Fangradius (5 Pixel), so wird es zurückgegeben. **Zu beachten ist**, dass die Schätzung, wieviel Meter 5 Pixel entsprechen, auf Gauß-Krüger-Koordinaten basiert. Für Koordinatenangaben in Grad ist diese Funktion unbrauchbar.

Darauf werden die Linien-Polygone durchsucht. Ist der Abstand von p zu einem Linienzug innerhalb des Fangradius, so wird dieses Polygon zurückgegeben.

Schließlich werden alle Flächen-Polygone durchsucht. Ist der Punkt p in der Fläche (aber nicht in einer Insel) enthalten, dann wird dieses Flächen-Polygon zurückgeliefert.

Wird kein Polygon gefunden, dann wird ein ungültiges zurückgeliefert (IsValid()).

Diese Funktion findet vor allem Anwendung in Azurfunktionen, die durch das Betätigen der mittleren Maustaste in einer AquaGramm-Karte aufgerufen wurden. Diese Funktionen nennt man **Handles**, sie werden z.B. aufgerufen, wenn ein Button gedrückt wird. Mit der Funktion SetHandle() kann diese Funktion gesetzt werden. Der Name des AGElements ist dabei der Name des GeoCanvas, auf dem die jeweilige Karte liegt.

Betätigt der Benutzer nun die mittlere Maustaste innerhalb der Karte, so wird der Handle aufgerufen. Die beiden Sonderparameter **AktMap** und **AktPoint** enthalten die Karte und den Punkt, der angeklickt wurde, in Gauß-Krüger-Koordinaten (siehe Beispiel oben).

Siehe auch PolyInside(), Selection() und GeoPoint().

FirstTupel

Syntax: FirstTupel(Relation R [, String idxfeld]) : Tupel
R: eine Mem-Relation oder eine UVS-Relation
optional: idxfeld: maßgebliches Indexfeld

Beispiel:
```
t := FirstTupel(Stamm, "ORT")
```

Beschreibung: Liefert das erste Tupel der Relation R zurück. Das für die Sortierung maßgebliche Feld wird mit $idxfeld$ angegeben. Ohne Angabe eines Indexfelds wird der Standardindex benutzt (siehe CreateIndex()) oder, wenn dieser nicht erstellt wurde, die unsortierte Reihenfolge.

Diese Funktion ist nur für Mem-Relationen oder UVS-Relationen zu verwenden!

Siehe auch LastTupel().

FirstZR

Syntax: FirstZR (ZRList zrl) : ZR
zrl: eine Zeitreihenliste

Beispiel:
```
zr1 := FirstZR( zrsel )
```

Beschreibung: Liefert die erste Zeitreihe einer Zeitreihenliste. Ist die Liste leer, so wird eine ungültige Zeitreihe zurückgegeben.

Siehe auch AnzZR() und ZRNr().

FolgenIDs

Syntax: FolgenIDs (ZR zr) : Relation
zr: eine Folgen-Zeitreihe

Beispiel:
```
vrel := FolgenIDs (zrf)
```

Beschreibung: In den Formeln, die bereichsweise die Berechnung der ZRFolge festlegen (siehe FolgenRel()), werden die Ausgangs-Zeitreihen als IDs abgelegt (z.B. zr1). Welche Zeitreihe zu welcher ID gehört, kann mit FolgenIDs abgefragt werden.

Die Struktur der Relation ist in ZRFolge() beschrieben.

FolgenRel

Syntax: FolgenRel (ZR zr) : Relation
zr: eine Folgen-Zeitreihe

Beispiel:
```
vrel := FolgenRel (zrf)
```

Beschreibung: Liefert die Folgenvorschrift einer Folgen-Zeitreihe.

Die Struktur des Ergebnisses ist `VonD#D,VonZ#T,Formel#250S`.

Siehe auch ZRFolge() und FolgenIds().

FToleranz

Syntax: FToleranz (ZR z) : Real
z:

Beispiel:
```
tol := FToleranz( z )
```

Beschreibung: FToleranz ist ein Attribut der Zeitreihe, die Fehlertoleranz. Siehe auch SetF-Toleranz().

FTPDel

Syntax: FTPDel (String url, String user, String passwd) : Bool
url: Name des Server inkl. Pfadname
user: Username zum Anmelden
passwd: Passwort zum Anmelden

Beispiel:
```
gut := FTPDel ("ftp.eraser.com/pub/killme.txt", "ftp", "ftp"))
```

Beschreibung: Löscht eine Datei auf einem FTP-Server. *url* enthält den Namen des FTP-Severs und den Namen der Datei inkl. Pfad. *user* und *passwd* werden für das Einloggen auf dem FTP-Server benutzt. Wenn man sie auf Leerstring setzt, erfolgt das Einloggen als Gast.

Falls das Löschen fehlschlägt (z.B. wegen fehlender Berechtigung), wird False zurückgeliefert, sonst True.

Siehe auch FTPGet(), FTPPut(), FTPDir() und FTPRename().

FTPDir

Syntax: FTPDir (String url, String user, String passwd) : Relation
url: Name des Server inkl. Pfadname
user: Username zum Anmelden
passwd: Passwort zum Anmelden

Beispiel:
```
Rel := FTPDir ("ftp.funet.fi/pub", "ftp", "ftp"))
```

Beschreibung: Holt den Inhalt eines Verzeichnisses auf einem FTP-Server. *url* enthält den Namen des FTP-Severs und den Pfadnamen. *user* und *passwd* werden für das Einloggen auf dem FTP-Server benutzt. Wenn man sie auf Leerstring setzt, erfolgt das Einloggen als Gast.

Das Ergebnis ist eine Relation, die die Felder Name, Dir, Groesse, Datum und Zeit enthält. Pro Datei im Verzeichnis wird ein Tupel in der Relation erzeugt. Dir ist Bool-Feld, das anzeigt, ob es sich um ein Verzeichnis handelt (True) oder eine Datei (False).

Falls die Verbindung fehlschlägt, wird eine entsprechende Fehlermeldung ausgegeben und eine ungültige Relation zurück gegeben.

Siehe auch FTPGet(), FTPPut() und FTPDel().

FTPGet

Syntax: FTPGet (String url, String user, String passwd) : String
url: Adresse der zu holenden Datei
user: Username zum Anmelden
passwd: Passwort zum Anmelden

Beispiel:
```
datei := FTPGet ("ftp.funet.fi/pub/README", "ftp", "ftp"))
```

Beschreibung: Holt eine Datei von einem FTP-Server. *url* enthält den Namen des FTP-Severs und den Dateinamen. *user* und *passwd* werden für das Einloggen auf dem FTP-Server benutzt. Wenn man sie auf Leerstring setzt, erfolgt das Einloggen als Gast.

Die Datei wird im aktuellen Verzeichnis gespeichert, ihr Name wird zurück gegeben.

Falls die Verbindung fehlschlägt, wird eine entsprechende Fehlermeldung ausgegeben und der Rückgabewert auf Leerstring gesetzt.

Siehe auch FTPMultiGet(), FTPPut() und FTPDir().

FTPMultiGet

Syntax: FTPMultiGet (String urldir, Array files, String user, String passwd) : Array
urldir: Adresse des FTP-Verzeichnisses
files: Liste mit Dateinamen
user: Username zum Anmelden
passwd: Passwort zum Anmelden

Beispiel:
```
A := FTPMultiGet ("ftp.funet.fi/pub", files, "ftp", "ftp"))
```

Beschreibung: Holt mehrere Dateien aus einem Verzeichnis von einem FTP-Server. *urldir* enthält den FTP-Server inkl. des Verzeichnisnamens. $files$ enthält die Dateinamen (siehe z.B. StrSplit()). *user* und *passwd* werden für das Einloggen auf dem FTP-Server benutzt. Wenn man sie auf Leerstring setzt, erfolgt das Einloggen als Gast.

Die Dateien werden im aktuellen Verzeichnis gespeichert, ihre Namen werden zurück gegeben.

Falls die Verbindung fehlschlägt, wird eine entsprechende Fehlermeldung ausgegeben und ein leeres Array zurück gegeben.

Siehe auch FTPGet() und FTPDir().

FTPPut

Syntax: FTPPut (String datei, String url, String user, String passwd) : Bool
datei: zu übertragende Datei
url: Ziel-Adresse
user: Username zum Anmelden
passwd: Passwort zum Anmelden

Beispiel:
```
gut := FTPPut ("aachen.zpa", "ftp.aquaplan.de/pub/aachen.zpa", "ftp", "f
```

Beschreibung: Speichert die Datei *datei* auf einen FTP-Server. *url* enthält den Namen des FTP-Severs und den Dateinamen, unter dem *datei* gespeichert werden soll. *user* und *passwd* werden für das Einloggen auf dem FTP-Server benutzt. Wenn man sie auf Leerstring setzt, erfolgt das Einloggen als Gast.

Der Rückgabewert gibt an, ob das Speichern erfolgreich war.

Falls die Verbindung fehlschlägt, wird zusätzlich eine entsprechende Fehlermeldung ausgegeben.

Siehe auch FTPGet() und FTPDir().

FTPRename

Syntax: FTPRename (String url, String user, String passwd, String newpath) : Bool
url: Name des Server inkl. Pfadname
user: Username zum Anmelden
passwd: Passwort zum Anmelden
newpath: Neuer Name

Beispiel:
```
gut := FTPRename ("ftp.test.pe/abc.txt", "ben", "nuk", "/def.txt"))
```

Beschreibung: Benennt eine Datei auf einem FTP-Server um. *url* enthält den Namen des FTP-Severs und den Namen der Datei inkl. Pfad. *user* und *passwd* werden für das Einloggen auf dem FTP-Server benutzt. Wenn man sie auf Leerstring setzt, erfolgt das Einloggen als Gast. *newpath* enthält den neuen Namen der Datei inkl. Pfad.

Falls das Umbenennen fehlschlägt (z.B. wegen fehlender Berechtigung), wird False zurückgeliefert, sonst True.

Siehe auch FTPDel(), FTPPut() und FTPDir().

FuellenZR

Syntax: FuellenZR(ZR reihe, Intervall b, Real sch, Real fuell0, String param, Real version, Bool tmp) : ZR
reihe: Ausgangsreihe
b: Auswertungszeitraum
sch: Schwellwert
fuell0: Fülle zum Beginn der Berechnung
param: Parameter der Ergebniszeitreihe
version: Attribut Version der Ergebniszeitreihe
tmp: Temporärflag

Beispiel:
```
szr := FuellenZR(q3, bereich, 3.5, 0, "Fuelle", 0, FALSE)
```

Beschreibung: Erzeugt eine Reihe mit Füllen eines (fiktiven) Speichers. *reihe* muss die Intensitäten eines Parameters (z.B. Abfluss in m^3/s) enthalten. Der Speicher wird gefüllt, wenn *reihe* größer als der Schwellwert *sch* ist. Er leert sich entsprechend, wenn diese Schwelle unterschritten wird. Die Fülle des Speichers kann nicht negativ werden.

$fuell0$ ist die Fülle des Speichers zu Anfang der Berechnung.

Beispiel: Der Speicher ist ein Becken, das parallel zu einem Fließgewässer geschaltet ist, dessen Gerinne bis zu *sch* m^3/s transportieren kann. Die darüber hinaus gehende Wassermenge läuft in das Becken. Ist der Abfluss kleiner als *sch*, spendet das Becken dem Gerinne die Differenz zu *sch*, solange bis es leer ist.

Siehe auch SchwellenMax(), IntZR(), SchwellenZR() und SubZR().

GeoCanvasOnPage

Syntax: GeoCanvasOnPage (Page seite, GeoPoint lu, GeoPoint ro)
seite: Eine Reportseite
lu: Punkt links unten in cm
ro: Punkt rechts oben in cm

Beispiel: `GeoCanvasOnPage (seite, {2,10}, {28,20})`

Beschreibung: Zeichnet den gesamten GeoCanvas des aktuellen Windows auf die Seite *seite*. Die Punkte *lu* und *ro* geben die Lage des GeoCanvas auf der Seite in cm an. Der GeoCanvas kann also in x- und y-Richtung gedehnt oder gestaucht werden!

Siehe auch PlotKarte() und CanvasOnPage().

GeoDist

Syntax: GeoDist (GeoPoint p1, GeoPoint p2) : Real
p1,p2: GeoPoints

Beispiel: `abstand := GeoDist (mitte, oben)`

Beschreibung: Berechnet den Abstand der Punkte $p1$ und $p2$.

Siehe auch PolyDist().

GeoPoint

Syntax: GeoPoint (Real rx, Real ry) : GeoPoint
rx:
ry:

Beispiel: `point := GeoPoint (25501345, 5600450)`

Beschreibung: Macht aus rx und ry einen GeoPoint mit der X-Koordinate rx und der Y-Koordinate ry. Typischerweise sind dies Rechts- und Hochwert einer Gauß-Krüger-Koordinate. Statt dieser Funktion kann auch der Operator `{ , }` benutzt werden (z.B. `{10+a,73}`).

GetAGWindow

Syntax: GetAGWindow() : String

Beispiel:
```
name := GetAGWindow()
```

Beschreibung: Liefert den Namen des aktuellen AGWindows.

Der Name wird stets in Großbuchstaben geliefert, unabhängig davon, wie er erzeugt wurde!

Siehe auch SetAGWindow(), NewAGWindow() und DelAGWindow().

GetBool

Syntax: GetBool(Tupel t, String feld) : Bool
t: Tupel
feld: Name des Feldes im Tupel

Beispiel:
```
r := GetBool( t, "Ausgang" )
```

Beschreibung: Liefert den Inhalt des Feldes $feld$ des Tupels t. $feld$ muss ein Boolfeld sein.

Siehe auch SetBool().

GetDatenpool

Syntax: GetDatenpool() : String

Beispiel:
```
pool := GetDatenpool()
```

Beschreibung: Liefert den Namen des aktuellen Datenpools. Siehe auch SetDatenpool().

GetDatum

Syntax: GetDatum (Tupel t, String feld) : Zeitpunkt
t: Tupel
feld: Name des Feldes im Tupel

Beispiel:
```
zp := GetDatum(t, "Wann")
```

Beschreibung: Liefert den Inhalt des Feldes $feld$ des Tupels t. $feld$ muss ein Datumfeld sein. Stunde, Minute und Sekunde werden auf 0 gesetzt.

Siehe auch GetText(), GetZahl() und SetDatum().

GetDir

Syntax: GetDir () : String

Beispiel:
```
dir := GetDir()
```

Beschreibung: Liefert den Namen des aktuellen Verzeichnisses.

Siehe auch ChangeDir() und MakeDir().

GetEnv

Syntax: GetEnv(String name) : String
name: Name der Umgebungsvariablen

Beispiel:
```
code := GetEnv( "Code")
```

Beschreibung: Liefert den Inhalt der Umgebungsvariablen *name*. Ist diese nicht gesetzt, so wird der String UNSET geliefert.

Diese Funktion entspricht dem Betriebssytembefehl set bzw. printenv.

Siehe auch SetEnv().

GetFocus

Syntax: GetFocus (ZRList liste) : Intervall
liste: Reihenliste

Beispiel:
```
focus := GetFocus (liste)
```

Beschreibung: Ermittelt den Bearbeitungszeitraum einer Reihenliste.

GetGlobal

Syntax: GetGlobal(String name) : String
name: Name einer globalen String-Variablen

Beispiel:
```
wert := GetGlobal( "Betreiber" )
```

Beschreibung: Liefert den Inhalt einer globalen Variablen. Dieser muss vorher mit der Funktion SetGlobal() gesetzt worden sein. Ist keine globale Variable namens *name* vorhanden, dann wird ein Leerstring geliefert.

Siehe auch SetGlobal().

GetLegende

Syntax: GetLegende (AxBox ax, Real rx, Real ry) : String
ax: Die AxBox, zu der ein Legendeneintrag abgefragt wird
rx: Legendenspalte
ry: Legendenzeile

Beispiel:
```
s := GetLegende (axunten, 1, 1)
```

Beschreibung: Liefert den Eintrag mit der Spalte rx und der Zeile ry der Legende der AxBox ax.

Liegen rx oder ry außerhalb des gültigen Bereichs, dann wird ein Leerstring zurückgeliefert.

Siehe auch SetLegende().

GetPID

Syntax: GetPID () : Real

Beispiel:
```
pid := GetPID()
```

Beschreibung: Liefert die Prozess-ID des laufenden Prozesses.

Siehe auch System().

GetText

Syntax: GetText (Tupel t, String feld [, Bool komma[, Bool coderesolve]]) : String
t: Tupel
feld: Name des Feldes im Tupel
komma: optional: Dezimalkomma verwenden, Voreinstellung: False
coderesolve: optional: Codes in Klartext wandeln, Voreinstellung: False

Beispiel:
```
s := GetText (t, "Name")
```

Beschreibung: Liefert den Inhalt des Feldes $feld$ des Tupels t.

Wenn $feld$ kein Textfeld ist, wird ein passender String benutzt. Realfelder werden entsprechend ihres Formats ausgegeben, optional kann man dabei zwischen Dezimalpunkt (Voreinstellung) oder Dezimalkomma ($komma$=True) wählen.

Optional kann für mit ADBOpenRel() geöffnete Relationen angegeben werden, dass, falls das Feld ein Code-Feld ist, statt des Inhalts der Klartext geliefert werden soll. Wie man an den Klartext gelangt, ist mit ADBSetResolveInfo() festlegbar. Falls keine Regel zum auflösen des Codes festgelegt ist, wird der Inhalt selbst geliefert.

Um den Inhalt mehrerer Felder zu erhalten, übergibt man die Feldnamen mit + getrennt. Der Rückgabestring enthält dann die Inhalte mit Leerzeichen getrennt.

GetZahl

Syntax: GetZahl (Tupel t, String feld) : Real
t: Tupel
feld: Name des Feldes im Tupel

Beispiel:
```
r := GetZahl (t, "Betrag")
```

Beschreibung: Liefert den Inhalt des Feldes $feld$ des Tupels t. $feld$ muss ein Zahlfeld sein.

GetZeit

Syntax: GetZeit (Tupel t, String feld [, Zeitpunkt datum]) : Zeitpunkt
t: Tupel
feld: Name des Feldes im Tupel
datum: optional Maske für Tage, Monat, Jahr

Beispiel:
```
zp := GetZeit (t, "Wann", @"15.9.1992") oder
zp := GetZeit (t, "Zeit", GetDatum(t,"Datum"))
```

Beschreibung: Liefert den Inhalt des Feldes $feld$ des Tupels t. $feld$ muss ein Zeitfeld sein. Tag, Monat und Jahr werden aus dem Zeitpunkt *datum* übernommen, wenn dieser angegeben ist. Sonst werden sie auf 1.1.1900 gesetzt.

Um Zeitpunkte in Tupeln abzulegen, gibt es auch die Funktionen GetZP() und SetZP().

Siehe auch GetText(), GetZahl(), GetDatum() und SetZeit().

GetZP

Syntax: GetZP (Tupel t, String feld) : Zeitpunkt
t: Tupel
feld: Name des Feldes im Tupel

Beispiel:
```
zp := GetZP (t, "Wann")
```

Beschreibung: Liefert den Inhalt des Feldes $feld$ des Tupels t. $feld$ muss ein ZP-Feld (Zeitpunkt-Feld) sein.

Siehe auch SetZP(), GetText(), GetZahl(), GetDatum() und GetZeit().

GetZR

Syntax: GetZR(String ort, String param, String aussage, String defart,
String herkunft, String reihenart, Distanz distanz,
Real version) : ZR
ort: Gewünschter Ort der Zeitreihe (u.U. mit SubOrt)
param: Gewünschter Parameter der Zeitreihe
aussage: Gewünschte Aussage der Zeitreihe
defart: Gewünschte Definitionsart der Zeitreihe `"K"`, `"I"` oder `"M"`
herkunft: Herkunft der Reihe (u.U. inkl. Quelle) (siehe Anhang)
reihenart: Art der Reihe, z.B. Zeitreihe (siehe Anhang)
distanz: für äquidistante Intervall-Zeitreihen (siehe Anhang)
version: Version (siehe Anhang)

Beispiel: `z := GetZR("12345", "Niederschlag", "", "K")`

Beschreibung: Liefert zu den angegebenen Attributen die passende Zeitreihe.

Wird *aussage*, wie im Beispiel, auf Leerstring gesetzt, so findet sie bei der Auswahl keine Beachtung und es wird die erste passende Zeitreihe gewählt. Um eine Zeitreihe zu erreichen, die eine leere Aussage hat, setzt man *aussage* auf `"."`.

Die Parameter *herkunft*, *reihenart*, *distanz* und *version* können, wie im Beispiel auch, (von links nach rechts) weggelassen werden. Falls zu den angegebenen Parametern mehrere Reihen existieren, wird die erste passende genommen.

Der SubOrt kann dem Parameter *ort* mit einem `|` getrennt angefügt werden. Beispiel `z := GetZR("Lohhausen|PumpeIII", "Abfluss", "", "K")`

Um die Quelle anzugeben, wird diese einem `|`-Zeichen an die Herkunft ängehängt. Wenn die Herkunft `F` und die Quelle `P` sein soll, gibt man an:
`z := GetZR("12345", "Niederschlag", "", "K", "F|P")`

GetZRList

Syntax: GetZRList (String datei) : ZRList
datei: Name der Datei

Beispiel:
```
zrl := GetZRList ("zrsel.dat")
```

Beschreibung: Liefert eine Zeitreihenliste aus der Datei *datei*.

GKToGrad

Syntax: GKToGrad(GeoPoint p) : GeoPoint
p: Punkt in Gauß-Krüger-Koordinaten

Beispiel:
```
p2 := GKToGrad(p)
```

Beschreibung: Rechnet den Gauß-Krüger-Punkt p um in geografische Koordinaten. Der X-Wert von p ist der Rechtswert und der Y-Wert der Hochwert. Im Ergebnis ist der X-Wert die geografische Länge und der Y-Wert die geograische Breite.

Siehe auch GradToGK() und GKTrans().

Zur Transformation in andere Koordinatensysteme siehe auch LambertToGrad() und UTMToGrad().

GKTrans

Syntax: GKTrans (GeoPoint p, Real meridian) : GeoPoint
p:
meridian: Hauptmeridian

Beispiel:
```
p2 := GKTrans (p, 3)
```

Beschreibung: Rechnet den Gauß-Krüger-Punkt p auf den Hauptmeridian *meridian* um und gibt den neuen Punkt zurück.

Glaetten

Syntax: Glaetten (ZR z, Intervall be, Real toleranz [, Bool neu) : ZR
z: Ausgangsreihe
be: Berechnungszeitraum
toleranz: Fehlertoleranz der Ergebnisreihe
neu: optional: neue Reihe anlegen. Voreinstellung: True

Beispiel:
```
zrg := Glaetten(reihe, bereich, 0.1, True)
```

Beschreibung: Die Ausgangsreihe z wird auf dem Bereich *be* geglättet.

Wenn *neu* True ist, dies ist die Voreinstellung, wir die Reihe z dabei nicht verändert. Es wird eine neue, temporäre Reihe angelegt, die als Ergebnis zurück geliefert wird.

Wenn *neu* False ist, findet die Glättung direkt auf der Reihe z statt. Die alten Daten gehen dadurch verloren. Die Attribute, eingeschlossen der FToleranz(), werden nicht verändert.

Zur Vorgehensweise beim Glätten siehe [**?**, **?**].

Siehe auch GleitWerte()

GleitWerte

Syntax: GleitWerte (ZR z, Intervall be, Distanz d, String aus, Bool temp) : ZR
z: Ausgangsreihe
be: Berechnungszeitraum
d: Mittelbildner-Breite
aus: Aussage: Mit, Min oder Max
temp: Temporärflag

Beispiel:
```
zrg := GleitWerte(reihe, bereich, ~"1 Tag", "Mit", True)
```

Beschreibung: Auf die Ausgangszeitreihe z wird das gleitende Mittel/Maximum/Minimum angewendet. Jeder Wert wird durch das Mittel/Min/Max des Intervalls der Breite d ersetzt (zentriert um den Punkt). Am Schluss werden die so gewonnenen Daten mit der Fehlertoleranz (FToleranz()) der Ausgangszreihe entkolinearisiert.

Siehe auch Glaetten().

GradToGK

Syntax: GradToGK(GeoPoint p) : GeoPoint
p: Punkt in geografischen Koordinaten

Beispiel:
```
p2 := GradToGK(p)
```

Beschreibung: Rechnet den Punkt p in Gauß-Krüger-Koordinaten um. Der X-Wert von p ist die geografische Länge und der Y-Wert die geografische Breite. Im Ergebnis ist der X-Wert der Rechtswert und der Y-Wert der Hochwert.

Siehe auch GKToGrad() und GKTrans().

Zur Transformation in andere Koordinatensysteme siehe auch GradToLambert() und GradToUTM().

GradToLambert

Syntax: GradToLambert (GeoPoint p, Bool nord) : GeoPoint
p: Punkt in geografischen Koordinaten
nord: True=Algerien Nord, False= Algerien Süd

Beispiel:
```
p2 := GradToLambert (p, True)
```

Beschreibung: Berechnet die Lambert-Koordinaten des Punkts p, der in geografischen Koordinaten (Längengrad, Breitengrad) angegeben ist.

Zugrunde liegt der Ellipsoid nach Clarke,1880. Die Bezugsbreiten (Schnittkegel, **nicht** Berührkegel) sind fest vorgegeben und passen auf den Staat Algerien. Dieser ist in eine nördliche und eine südliche Zone eingeteilt.

Siehe auch LambertToGrad().

Zur Transformation in andere Koordinatensysteme siehe auch GradToGK() und GradToUTM().

GradToUTM

Syntax: GradToUTM (GeoPoint p) : String
p: Punkt in geografischen Koordinaten

Beispiel:
```
p2 := GradToUTM (p)
```

Beschreibung: Berechnet die UTM-Koordinaten des Punkts p, der in geografischen Koordinaten (Längengrad, Breitengrad) angegeben ist.

Der Bezugsmeridian, das Breitenband und das Quadrat werden automatisch berechnet.

Siehe auch UTMToGrad().

Zur Transformation in andere Koordinatensysteme siehe auch GradToGK() und GradToLambert().

GridPosTup

Syntax: GridPosTup(String name) : Tupel
name: Name eines DBGrids

Beispiel:
```
tup := GridPosTup( "tab")
```

Beschreibung: Liefert ein Tupel, das für jede Komponente der Relation des DBGrids *name* ein numerisches Feld enthält. In diesen Feldern sind die X-Koordinaten der Pixelpositionen der jeweiligen Tabellenspalten abgelegt. Die Position gibt die Lage der linken Begrenzungslinie der Komponente an.

Diese Positionen müssen nicht notwendigerweise auf dem Window liegen, da der Benutzer die Tabelle horizontal scrollen kann.

Falls auf dem aktuellen Window kein DBGrid mit Namen *name* enthalten ist, wird ein ungültiges Tupel geliefert.

Siehe auch AGElemPos() und NewDBGrid().

GStr

Syntax: GStr(Real r [,Bool komma [,Bool ttrenner]]) : String
r: eine Realzahl
komma: optional: Dezimalkomma verwenden, Voreinstellung: False
ttrenner: optional: Tausender-Trenner verwenden, Voreinstellung: False

Beispiel:
```
s := GStr( 30 )
```

Beschreibung: Wandelt die Zahl *r* formatfrei in einen String um.

Formatfrei bedeutet, dass das Format `"g"` benutzt wird, unabhängig von dem mit RealFormat() gewählten. `"g"` erzeugt genau so viele Vor- und Nachkommastellen, wie benötigt werden, um die Zahl darzustellen.

Ist *komma* True, so wird statt eines Dezimalpunkts ein Dezimalkomma verwendet.

Ist *ttrenner* True, so werden die Tausenderblöcke mit einem Trennzeichen getrennt. Wenn *komma* True ist, wird dazu ein Punkt benutzt, sonst ein Komma.

Siehe auch Str(), RStr() und ZPStr().

GueltBis

Syntax: GueltBis (ZR z) : Zeitpunkt
z:

Beispiel:
```
gueltig := GueltBis (z)
```

Beschreibung: Liefert das Attribut GueltBis der Zeitreihe. Weitere Erklärungen finden sich bei SetGueltBis().

GueltVon

Syntax: GueltVon (ZR z) : Zeitpunkt
z:

Beispiel:
```
gueltig := GueltVon (z)
```

Beschreibung: Liefert das Attribut GueltVon der Zeitreihe.Weitere Erklärungen finden sich bei SetGueltBis().

Haeufigkeiten

Syntax: Haeufigkeiten (ZR z, Intervall bereich, Bool temp, Array grenzen) : ZR
z: Ausgangszeitreihe
bereich: Auswertungsbereich
temp: Temporärflag
grenzen: Klassengrenzen

Beispiel:
```
von := Haeufigkeiten (tzr, bereich, TRUE, StrToArray("0 10 50 100 250"))
```

Beschreibung: Erzeugt die Häufigkeitsverteilung einer Intervall- oder Momentan-Reihe. Das Ergebnis ist eine Intervall-Reihe.

Die Intervallgrenzen sind jeweils die Klassengrenzen g_i bis g_{i+1}. Der dazu abgelegte Wert ist die Anzahl Werte y, für die gilt, $g_i < y \leq g_{i+1}$. Werte, die exakt auf der untersten Klassengrenze liegen, fallen in die erste Klassse (z.B. 0). Werte kleiner als die erste Klassengrenze oder größer als die letzte Klassengrenze werden nicht berücksichtigt.

Die Summenhäufigkeit kann durch Anwenden der Funktion IntZR() auf das Ergebnis erzeugt werden (dabei Realintervall statt Zeitintervall benutzen).

Siehe auch Array() und StrToArray().

HEreignisse

Syntax: HEreignisse (ZR zr, Intervall bereich, Real M, [Real mind]) : ZR
zr: Abfluss- oder Wasserstands-Zeitreihe
bereich: Auswertungszeitraum
M: langjähriges Mittel (Q oder W)
mind: optional: Mindesthöhe eines Ereignisses

Beispiel:
```
zr2 := HEreignisse(zr, bereich, 0.35)
```

Beschreibung: Erzeugt eine Momentanzeitreihe mit Ereignissen (Scheitelwerten). Diese Ereignisse werden aus einer Abfluss- oder einer Wasserstands-Zeitreihe *zr* gebildet.

Ereignisse werden nach der Pegelvorschrift (Stammtext, S. 29) separiert. Ergänzend zur Vorschrift wird ein Ereignis nur als solches erkannt, wenn es mindestens die Höhe M (also MQ oder MW) erreicht. Diese Bedingung kann durch explizite Angabe des Mindestwertes *mind* noch verschärft werden.

Diese Funktion ist implizit in der Berechnung von partiellen und jährlichen Serien enthalten (siehe HPartSerie() und HJahrSerie()).

Der Parameter M kann z.B. mit `M := Mittel (zr, MaxFocusZR(zr))` bestimmt werden. Diese Auswertung kann jedoch lange dauern, weshalb dieser Parameter auch explizit vorgegeben werden kann.

Herkunft

Syntax: Herkunft (ZR z) : String
z:

Beispiel: `von := Herkunft( z )`

Beschreibung: Liefert das Attribut Herkunft der Zeitreihe. Siehe Anhang.

HexToReal

Syntax: HexToReal (String hex) : Real
hex : Ein Hex-String

Beispiel: `z := HexToReal (hex)`

Beschreibung: Wandelt einen String, der in Hexadezimaldarstellung angegeben ist, in die entsprechende Realzahl. Leerzeichen, Tabs und Zeilenumbrüche werden ignoriert.

Die höherwertigen Bytes stehen links.

Beispiel: `HexToReal("06 9D 4D E8")` ergibt `110972392`

Siehe auch StrHex().

HideZR

Syntax: HideZR (ZR reihe, Intervall bereich, Bool an)
reihe: zu stempelnde Reihe
bereich: Bereich, auf dem gestempelt wird
an: True=sperren, False=entsperren

Beispiel: `HideZR (niederzr, bereich, True)`

Beschreibung: Setzt die Reihe *reihe* auf dem Bereich *bereich* auf den Zustand **gesperrt** (*an*=True), oder hebt diesen Zustand wieder auf (*an*=False).

Der Ausgangszustand einer Reihe ist vollständig **entsperrt**.

Siehe auch SetZRBearbStand() und Stempeln().

HJahrSerie

Syntax: HJahrSerie (ZR zr, Intervall bereich, Real mittel [,Bool kal]) : ZR
zr: kontinuierliche Abfluss- oder Wasserstands-Zeitreihe
bereich: Auswertungszeitraum
mittel: das langjährige Mittel (MQ oder MW)
kal: optional, Kalenderjahr oder WWJ (Voreinstellung FALSE=WWJ)

Beispiel: `jserie := HJahrSerie(zr1, dekade, 0.5)`

Beschreibung: **Erweitert** die jährliche Serie der jeweiligen Zeitreihe. Ist die jährliche Serie leer, dann wird sie angelegt. Die Ergebniszeitreihe erbt alle Attribute von *zr*, mit Ausnahme von Aussage, welches auf jSe gesetzt wird.

Die jährliche Serie ist eine Momentan-Zeitreihe. Es wird pro Jahr ein Maximalwert in die jährliche Serie übernommen. Ein **Jahr** ist dabei ein Wasserwirtschaftsjahr. Wird *kal* jedoch auf TRUE gesetzt, dann ist ein **Jahr** ein Kalenderjahr.

mittel ist der langjährige mittlere Abflusss (MQ) oder Wasserstand (MW). Es wird benötigt, um die Hochwasserscheitel nach [1] zu separieren. Siehe dazu HEreignisse().

Zur Berechnung siehe [1] und HPartSerie().

Hoehe

Syntax: Hoehe(ZR zr) : Real
zr: Eine Reihe

Beispiel: `h := Hoehe ( niederzr )`

Beschreibung: Liefert das Attribut Hoehe der Reihe. Diese Angabe sollte in mNN sein. Siehe auch SetHoehe().

Hostname

Syntax: Hostname () : String

Beispiel: `host := Hostname()`

Beschreibung: Liefert den Namen des Rechners.

HPartSerie

Syntax: HPartSerie (ZR zr, Intervall bereich, Real mittel, [Real anz]) : ZR
zr: kontinuierliche Abfluss- oder Wasserstands-Zeitreihe
bereich: Auswertungszeitraum
mittel: das langjährige Mittel (MQ oder MW)
anz: optional: Voreinstellung 2.5

Beispiel: `pserie := HPartSerie(zr, dekade, 0.5, 4)`

Beschreibung: Erzeugt die partielle Serie der Zeitreihe *zr* auf dem Auswertungszeitraum *bereich*. Ist die partielle Serie leer, dann wird sie angelegt. Die Ergebniszeitreihe erbt alle Attribute von *zr*, mit Ausnahme von Aussage, welches auf pSe gesetzt wird.

Die Partielle Serie ist eine Momentan-Zeitreihe.

Der optionale Parameter *anz* legt fest, wieviele Werte die partielle Serie enthalten soll. Die Anzahl Werte ist *anz*· Anzahl Jahre in *bereich*.

mittel ist der langjährige mittlere Abfluss (MQ) oder Wasserstand (MW). Es wird benötigt, um die Hochwasserscheitel nach [1] zu separieren. Siehe dazu HEreignisse().

Zur Berechnung siehe [1].

HWVerteilung

Syntax: HWVerteilung (ZR zr, ZI zi, S typ, S pstyp, R maxtn, R version, B tmp) : ZR
zr: partielle oder jährliche Serie
zi: Auswertungszeitraum
typ: Typ der Verteilungsfunktion (AE, E1, ME, P3, WB3, LN3, LP3)
pstyp: Typ der Parameterschätzung (MM, WGM, MLM)
maxtn: maximale Jährlichkeit
version: Attribut Version der Ergebnisreihe
tmp: Temporärflag(true,false)

Beispiel:
```
vzr := HWVerteilung (izr, focus, "LN3", "WGM", 100, 0, False)
```

Beschreibung: Erzeugt die statistische Verteilungsfunktion des gewünschten Typs aus einer partiellen oder jährlichen Serie *zr*.

Mit *pstyp* legt man die Methode fest, die zur Parameterschätzung benutzt werden soll: MM=Momentenmethode, WGM=Wahrscheinlichkeitsgewichtete Momemtenmethode und MLM=Maximum-Likelihood-Methode. Siehe *version* legt das Attribut Version der Ergebnisreihe fest.

Die Stichprobe ergibt sich aus allen Werten in *zr*, die in *zi* liegen. Die Parameter der entsprechenden Verteilungsfunktion werden berechnet. Für einen Bereich knapp über 0 bis *maxtn* werden an ausreichend vielen Stützstellen die Funktionswerte zu den Jährlichkeiten berechnet. Diese Werte werden in die Ergebnisreihe eingetragen (siehe [9]).

tmp bestimmt Herkunft der Reihe: Momentan if *tmp* true ist oder Abgeleitet sonst.

Das Ergebnis ist eine kontinuierliche Realreihe, die alle Attribute von *zr* erbt, mit Ausnahme von: XEinheit (*a*) und Aussage, die *typ* entspricht.

IEC870Recv

Syntax: IEC870Recv (Real port, Real timeout) : String
port: Portnummer der seriellen Schnittstelle, 1-100.
timeout: Timeout in Sekunden

Beispiel:
```
daten := IEC870Recv (2, 5)
```

Empfängt ein Telegramm des Protokolls IEC 60870. Falls *timeout* Sekunden kein Telegramm zu empfangen ist, wird ein Leerstring geliefert.

Geliefert werden nur die Nutzdaten, bei Kurztelegrammen also zwei Bytes. Ein einzelnes E5 wird als solches geliefert.

Die Schnittstelle muss mit SerOpen() geöffnet worden sein.

Siehe auch IEC870Send().

IEC870Send

Syntax: IEC870Send (Real port, String daten)
port: Portnummer der seriellen Schnittstelle, 1-100.
daten: Nutzdaten

Beispiel:
```
IEC870Send (2, StrFromHex("C9 01"))
```

Sendet ein Telegramm des Protokolls IEC 60870.

Die Nutzdaten *daten* werden in einem entsprechenden Telegrammrahmen verpackt. Besteht *daten* nur aus E5, wird kein Rahmen erzeugt. Sind genau zwei Bytes in *daten*, so wird ein Kurztelegramm erzeugt, andernfalls ein Langtelegramm.

Die Schnittstelle muss mit SerOpen() geöffnet worden sein.

Siehe auch IEC870Recv().

ImpCSV

Syntax: ImpCSV (String datei, String param, String feld [, String invfeld]) : Relation
datei: Datei mit CSV-Daten
param: Parameter der Zeitreihe
feld: Name des Feldes mit IO-Nummern
optional: invfeld: Name des Feldes, in dem steht ob die Werte mit -1 multipliziert werden sollen

Beispiel: `mrel := ImpCSV("20011004.csv", "Abstich", mstammrel, "IONR")`

Beschreibung: Importiert Daten, die im CSV-Format vorliegen (Comma Separated Values) in eine oder mehrere Zeitreihen.

Das Format der CSV-Datei ist weitestgehend fix. Pro Tag und Messstelle gibt es eine Zeile. Die Felder müssen mit einem Semikolon getrennt sein. Das erste Feld enthält die Messstellennummer im fremden System, also die IO-Nummer. *feld* gibt den Namen des Felds mit IO-Nummer an. Das nächste Feld enthält einen Kommentar, der jedoch verworfen wird.

Die weiteren Felder enthalten pro Zeitschritt einen Wert. Die Breite des Zeitschritts ergibt sich aus der Anzahl der Felder der Zeile. Die Zeile umfasst immer einen vollständigen Tag.

Beispiel einer CSV-Datei (alles eine Zeile):

```
341101m8;Hoehe Bach;15.10.2001;0.69;0.59;0.63;0.67;0.67;0.59;0.63;
0.68;0.65;0.60;0.67;0.69;0.60;0.60;0.66;0.69;0.58;0.61;0.65;0.67;
0.58;0.62;0.67;0.68;0.58;0.62;0.67;0.65;0.60;0.65;0.68;0.67;0.59;
0.62;0.66;0.68;0.69;0.60;0.63;0.67;0.69;0.65;0.59;0.65;0.67;0.59;
0.65;0.69;0.63;0.61;0.66;0.69;0.59;0.63;0.68;0.67;0.59;0.64;0.69;
0.62;0.60;0.66;0.69;0.60;0.62;0.66;0.67;0.59;0.63;0.69;0.59;0.63;
0.68;0.66;0.58;0.63;0.68;0.61;0.61;0.67;0.67;0.59;0.64;0.69;0.60;
0.62;0.68;0.66;0.60;0.67;0.69;0.58;0.61;0.66;0.65;0.58
```

Die Attribute der Zeitreihe (bestehende Zeitreihen werden erweitert) ergeben sich zu: Ort wird über die IO -Nummer in den Stammdaten gefunden (siehe ADBZROrt(), Parameter ist *param*, die DefArt ist fix K und die Reihenart fix Z. Falls kein Ort zu einer in der Datei auftauchenden IO-Nummer gefunden werden kann, wird keine Zeitreihe erzeugt und eine Fehlermeldung in die Rückgabe-Relation geschrieben.

Das Format der zurückgegebenen Relation ist ORT#20S,TAG#D,FEHLER#30S. ORT enthält den Ort der Zeitreihe oder, wenn dieser nicht gefunden wurde, die IO-Nummer. TAG enthält den importierten Tag, FEHLER bei Fehler eine Fehlermeldung, sonst einen Leerstring.

Siehe auch Import().

Import

Syntax: Import(String datei, String info [, Real qual [, Bool tmp]]) : ZR
datei: Datei mit Zeitreihe im Fremdformat
info: Formatspezifische Zusatzinfo
qual: optional Qualitätsstufe, Voreinstellung 0
tmp: optional Temporärflag, Voreinstellung FALSE

Beispiel:
```
z := Import("muehlen.82", "")
```

Beschreibung: Importiert die in einem Fremdformat vorliegende Zeitreihe in das System. Das Format wird anhand markanter Stellen in der Datei automatisch erkannt. Verarbeitet werden unter anderem DVWK, ED, MD, NasimBlock, NasimXY, UVF, LWAFlut, Ascii, Ott, Geotron, MDSII, DIALON und IDC. $info$ kann Zusatzinformation enthalten, die spezifisch für ein bestimmtes Format ist.

Möglichkeiten für *info*:

- Für Format DVWK können mit *info*=MESSER die Messerwerte mit importiert werden, was standardmäßig nicht der Fall ist. Der dritte Parameter *qual* muss nicht angegeben werden und ist dann 0.
- Bei allen Formaten können mit O=Ort, P=Parameter, D=DefArt (gilt nicht für Ott-Format), E=Einheit, F=FToleranz und I=Zeitdistanz die jeweiligen ZR-Attribute übersteuert werden. Bei einigen Formaten sind nur wenige oder gar keine Attribute automatisch bestimmbar, so dass diese Angaben nötig sind.
- Beim Format DVWK-Pegel kann mit HEUTE=Datum eingestellt werden, wann der Abruf erfolgt ist. In diesem Format wird das Jahr nicht angegeben, daher muss es aus dem Abrufdatum berechnet werden. Wenn der Import mehr als ein Jahr nach dem Abruf ist, ist diese Angabe zwingend. Beispiel HEUTE=28.06.2002.

Mit dem optionalen Parameter *tmp* wird gesteuert, ob die enstehende(n) Zeitreihe(n) temporär ist (sind) oder nicht. Auf Anfrage kann diese Liste von Formaten beliebig erweitert werden.

Wenn in der Datei mehrere Zeitreihen enthalten sind, empfiehlt sich die Benutzung der Funktion ImportListe(), da Import nur die erste importierte Zeitreihe liefert.

Siehe auch ZRFormat().

ImportDBF

Syntax: ImportDBF(ZR z, Real qualy, String zeit, String inrel)
z:
qualy: Qualität, in die importiert werden soll
zeit: HH:MM:SS, wenn zeit-Feld fehlt
inrel: Name der Relation

Beispiel:
```
ImportDBF( zr1, MAXQUAL, 20, "", "jahr89" )
```

Beschreibung: Importiert Wertepaare aus der Relation *inrel* in die Qualität *qualy* der Zeitreihe *z*.

Falls die Relation kein zeit-Feld besitzt, kann der Zeitpunkt mit dem Parameter *zeit* im Format HH:MM:SS gesetzt werden. Enthält *zeit* kein gültiges Format, dann wird 7:30 angenommen.

Siehe auch ExportDBF().

ImportListe

Syntax: ImportListe(String datei, String info [, Real qual [, Bool tmp]]) : ZRList
datei: Datei mit Zeitreihe im Fremdformat
info: Formatspezifische Zusatzinfo
qual: optional Qualitätsstufe, Voreinstellung 0
tmp: optional Temporärflag, Voreinstellung FALSE

Beispiel:
```
zrl := ImportListe("pegel.roh", "", 0, True)
```

Beschreibung: Importiert die in einem Fremdformat vorliegenden Zeitreihen und gibt diese in einer Liste zurück. Die weiteren Parameter sind in Import() beschrieben.

ImportQF

Syntax: ImportQF (String data) : QL
data: binäres TSTP-Format einer Quantenfolge

Beispiel:
```
qf := ImportQF (data)
```

Beschreibung: Produziert aus *data* eine QuantList (Quantenfolge). *data* wurde vorher mitttels WebGet() von einem TSTP-Server geholt.

Siehe auch ExportQF().

ImportRel

Syntax: ImportRel(Relation R, ZR z, Real qualy, Bool entkol, Bool mischen)
R: relation, aus der Werte importiert werden
z: Zeitreihe, in die Werte importiert werden
qualy: Qualität, in die importiert werden soll
entkol: soll entkolinearisiert werden?
mischen: sollen die Werte zu den vorhandenen dazugemischt werden?

Beispiel:
```
ImportRel(vwrel, zr, 0, True, False)
```

Beschreibung: Importiert Wertepaare aus der Relation *inrel* in die Qualität *qualy* der Zeitreihe z. Die Struktur der Relation ist in ScanRel() beschrieben.

Berücksichtigt werden alle Tupel, deren erstes Feld den Ort der Zeitreihe z enthält.

Im Beispiel werden überflüssige Stützstellen entfernt (entkolinearisiert) und die bestehenden Werte ersetzt.

Siehe auch ScanRel().

ImportVar

Syntax: ImportVar (String elename [,String agname]) : String
elename: Name des Aquagramm-Elements
agname: optional Name des Aquagramm-Fensters

Beispiel:
```
name := ImportVar ("eingabe2")
```

Beschreibung: Liefert den Inhalt des Aquagramm-Elements *elename* als String. Wird nur ein Aquagramm-Fenster benutzt, dann kann auf die Aquagramm-Elemente auch über die Parameterliste zugegriffen werden. Diese decken jedoch nur die Elemente des aktuellen Fensters ab.

Soll auf Elemente aus anderen Aquagramm-Fenstern zugegriffen werden, kann diese Funktion benutzt werden. Das gewünschte Fenster kann entweder mittels SetAGWindow() eingestellt oder als zweiter Parameter an ImportVar übergeben werden.

Falls das Element *elename* oder das Fenster *agname* nicht existiert, wird ein Leerstring zurückgegeben.

Der Rückgabewert ist immer ein String. Komplexe Typen müssen mittels einer Hilfsfunktion aus diesem String gewonnen werden. Siehe dazu StrToArray(), StrToAxBox(),StrToKarte() und StrToReal().

Für die Kommunikation mit Listen und DBGrids siehe NewListe() bzw. NewDBGrid().

Siehe auch SetAGWindow() und ExportVar().

IngRunden

Syntax: IngRunden (Real zahl, Real stellen) : Real
zahl: Zahl, die gerundet werden soll
stellen: Anzahl signifikater Stellen

Beispiel: `a := IngRunden(345.3, 2)`, → a = 350
`b := IngRunden(0.03467,2)`, → b = 0.03

Beschreibung: Rundet *zahl* auf *stellen* Anzahl Stellen genau. Diese Funktion entspricht nicht dem Runden von Zahlen in anderen Programmiersprachen. Wenn *zahl* mindestens *stellen* **Vor**kommastellen hat, dann wird die Mantisse der Zahl auf *stellen* Stellen genau gerundet (Beispiel a). Ist die Anzahl Vorkammstellen kleiner als *stellen*, dann wird *zahl* auf (*stellen* - Anzahl Vorkommastellen) Nachkommastellen genau gerundet.

InputBox

Syntax: InputBox (String text, Real breite, String vorgabe [,Bool abbr [,Real maxz [, Bool geheim]]]) : String
text: beliebiger Text
breite: Breite des Eingabefeldes in Pixeln
vorgabe: Vorbelegung des Eingabefeldes
abbr: optional: es wird ein Abbruchbutton erzeugt
maxz: optional: Maximale Anzahl eingebbarer Zeichen
geheim: optional: Eingabe erfolgt verdeckt

Beispiel:
```
dateiname := InputBox("Bitte Dateinamen eingeben", 100, "out.dat")
```

Beschreibung: Erzeugt ein neues Fenster, welches den Text *text*, ein Eingabefeld und einen OK-Button enthält. *breite* ist die Breite dieses Eingabefeldes und *vorgabe* dessen Vorbelegung.

Mit *abbr* kann eingestellt werden, ob neben dem OK-Button auch ein Abbruch-Button erzeugt werden soll.

Mit *maxz* kann optional angegeben werden, wieviel Zeichen der Benutzer maximal eingeben kann. Die Voreinstellung ist 0, also beliebig viele.

Wenn *geheim* True ist, erfolgt die Eingabe verdeckt. Die Voreinstellung ist False.

Das AGWindow, aus dem diese Input-Box gestartet wurde, ist solange inaktiv, bis der Benutzer den OK-Button (oder den Abbruch-Button) gedrückt hat oder im Eingabefeld Return drückt. Der Programmablauf wird solange angehalten.

Der Rückgabewert ist der Inhalt des Eingabefeldes, es sei denn der Abbruch-Button wurde gedrückt, dann ist der Inhalt das Ascii-Zeichen 3. (Char(3))

Siehe auch OkBox(), SelectBox(), ElementBox(), MultiBox() und ZIBox().

IntDauerlinien

Syntax: IntDauerlinien (ZR z, Intervall focus, Bool b) : ZR
z:
focus: Berechnungszeitraum
b: Temporärflag

Beispiel: `idl := IntDauerlinien (zr, langfocus, True)`

Beschreibung: Berechnet langjährige Dauerlinien pro Monat. Für die Berechnung siehe Dauerlinie().

Die Vorgehensweise sei am Beispiel für langjährige Dauerlinien für den Fokus 1935 bis 2005 mit Tagesmitteln erläutert:

Pro Monat wird eine Dauerlinie über alle Tagesmittel des Monats in allen Jahren erstellt, für den Januar sind das 2170, für den Februar 1978 usw. Die Dauerlinien werden anschließend jeweils auf die Breite eines Monats für das Jahr 1900 (der Februar hat also 28 Tage) gestaucht. Dadurch enstehen für den Januar Intervalle mit Breite 20:30 min und 20:35 min, je nach Rundungsabschlag.

Die so gewonnene Zeitreihe kann dazu benutzt werden, langjährige **Quantile** zu bestimmen. Beispielsweise berechnet das 5%-Quantil des Monats Januar sich, indem auf den 1.1.1900 0,95*31 Tage addiert werden und an diesem Zeitpunkt (29.1.1900 10:48) der Wert der Zeitreihe geholt wird.

Siehe auch Dauertabelle().

Interprete

Syntax: Interprete(String s)
s: Azurbefehle

Beispiel: `Interprete( userinput )`

Beschreibung: Führt die in *s* enthaltenen Azur-Befehle aus.

Wenn im Beispiel *userinput* den Wert `print ("AQTP header sent")` enthält, dann erzeugt Interprete die entsprechende Ausgabe.

IntervallMax

Syntax: IntervallMax (ZR z, Intervall i, Distanz d, Bool b [, R l]) : ZR
z:
i: Berechnungszeitraum
d: Breite der Intervalle, für die die Berechnung stattfindet
b: Temporärflag
l: optional: Lückenlimit in % (Voreinstellung 100)

Beispiel: `zrmax := IntervallMax( z, i, ~"3 Monate", b )`

Beschreibung: Erzeugt eine Intervall-Zeitreihe, deren Intervalle die Breite d haben. d kann z.B. `~"3 Monate"` sein. Der Y-Wert zu diesen Intervallen ist jeweils das Maximum aller Werte, die in dieses Intervall fallen, stützpunkt-unabhängig (Ränder interpoliert). Die Aussage der Ergebniszeitreihe wird auf Max gesetzt, die Intervallbreite wird in den Attributen XDistanz und XFaktor (siehe dazu Zeitschritt()) vermerkt, die Herkunft wird auf A gesetzt, alle weiteren Attribute vererbt.

Der optionale Parameter l legt fest, ab wieviel Prozent lückenhafter Daten eine Lücke erzeugt wird. Wird dieser Wert nicht angegeben, so wird erst eine Lücke für das Intervall erzeugt, wenn die Zeitreihe dort ausschließlich Lücke ist.

Um neben den Werten auch die Zeitpunkte der Maxima zu erhalten, wendet man die Funktion DistMaxima() an.

IntervallMin

Syntax: IntervallMin (ZR z, Intervall i, Distanz d, Bool b [, R l]) : ZR
z:
i: Berechnungszeitraum
d: Breite der Intervalle, für die die Berechnung stattfindet
b: Temporärflag
l: optional: Lückenlimit in % (Voreinstellung 100)

Beispiel: `zrmin := IntervallMin( z, i, ~"1 Tag", b )`

Beschreibung: Erzeugt eine Intervall-Zeitreihe, deren Intervalle die Breite d haben. d kann z.B. `~"1 Tag"` sein. Der Y-Wert zu diesen Intervallen ist jeweils das Minimum aller Werte, die in dieses Intervall fallen, stützpunkt-unabhängig (Ränder interpoliert). Die Aussage der Ergebniszeitreihe wird auf Min gesetzt, die Intervallbreite wird in den Attributen XDistanz und XFaktor (siehe dazu Zeitschritt()) vermerkt, die Herkunft wird auf A gesetzt, alle weiteren Attribute vererbt.

Der optionale Parameter l legt fest, ab wieviel Prozent lückenhafter Daten eine Lücke erzeugt wird. Wird dieser Wert nicht angegeben, so wird erst eine Lücke für das Intervall erzeugt, wenn die Zeitreihe dort ausschließlich Lücke ist.

Um neben den Werten auch die Zeitpunkte der Minima zu erhalten, wendet man die Funktion DistMinima() an.

IntervallMittel

Syntax: IntervallMittel (ZR z, Intervall i, Distanz d, Bool b [, R l]) : ZR
z:
i: Berechnungszeitraum
d: Breite der Intervalle, für die die Berechnung stattfindet
b: Temporärflag
l: optional: Lückenlimit in % (Voreinstellung 100)

Beispiel: `zrmittel := IntervallMittel( z, MAXFOCUS, ~"15 Minuten", FALSE)`

Beschreibung: Erzeugt eine Intervall-Zeitreihe, deren Intervalle die Breite *d* haben. *d* kann z.B. `~"15 Minuten"` sein. Der Y-Wert zu diesen Intervallen ist jeweils der Mittelwert aller Werte, die in dieses Intervall fallen, stützpunkt-unabhängig (Fläche / Breite). Die Aussage der Ergebniszeitreihe wird auf Mit gesetzt, die Intervallbreite wird in den Attributen XDistanz und XFaktor (siehe dazu Zeitschritt()) vermerkt, die Herkunft wird auf A gesetzt, alle weiteren Attribute vererbt. Die Berechnung der Einheit findet automatisch statt.

Der optionale Parameter *l* legt fest, ab wieviel Prozent lückenhafter Daten eine Lücke erzeugt wird. Wird dieser Wert nicht angegeben, so wird erst eine Lücke für das Intervall erzeugt, wenn die Zeitreihe dort ausschließlich Lücke ist.

IntervallQuant

Syntax: IntervallQuant(Intervall xbereich, Real ywert) : Quant
xbereich: Intervall des Quants
ywert: y-Wert

Beispiel: `q := IntervallQuant( [von,bis], 10 )`

Beschreibung: Erzeugt ein IntervallQuant, das sich über das Intervall *xbereich* erstreckt. Der y-Wert wird auf *ywert* gesetzt.

IntZR

Syntax: IntZR(ZR z, Intervall i, String s, Bool tmp) : ZR
z:
i: Berechnungszeitraum
s: Parameter der Ergebniszeitreihe
tmp: [**?**]

Beispiel:
```
int := IntZR( z, i, "Fracht", FALSE)
```

Beschreibung: Berechnet das unbestimmte Integral der Zeitreihe z. Die Werte werden dazu integriert (zum Algorithmus siehe [**?**]).

Die Berechnung der Ergebniseinheit findet automatisch statt.

IsAQTP

Syntax: IsAQTP() : Bool

Beispiel:
```
IF (IsAQTP())
```

Beschreibung: Liefert den Wert `TRUE` zurück, wenn das Programm im AquaWeb-Betrieb läuft, sonst wird `FALSE` zurückgegeben.

Diese Funktion sollte nur in wenigen Ausnahmefällen angewendet werden und steht keinesfalls zur Vefügung um allgemeine Funktionalität zu implementieren. Ohne genaue Kenntnisse ist diese Funktion nicht zu verwenden.

IsDirectory

Syntax: IsDirectory (String path) : Bool
path: Name einer Datei oder eines Verzeichnisses

Beispiel:
```
IF (IsDirectory(tauschpath))
```

Beschreibung: Gibt an, ob *path* ein Verzeichnis ist.

IsModified

Syntax: IsModified (Tupel t) : Bool
t: Tupel

Beispiel:
```
IF (IsModified(tup))
```

Beschreibung: Zeigt an, ob das Tupel *t* seit dem letzten ClearModify() oder seit dem Auslesen aus einer Relation verändert wurde.

Siehe auch RelModified() und FieldModified().

Isoflaechen

Syntax: Isoflaechen (Layer L, R von, R bis, R delta, R mincol, R maxcol, S name) : Layer
L: ein PunktLayer
von: ab welcher Höhe sollen Isolinien erzeugt werden
bis: bis zu welcher Höhe sollen Isolinien erzeugt werden
delta: Isolinienraster
mincol: welcher Höhe ist die Farbe 100 zugeordnet
maxcol: welcher Höhe ist die Farbe 200 zugeordnet
name: Name des entstehenden Layers

Beispiel:
```
Isos := Isoflaechen (PunktLayer, 75, 100, 1, 0, 100, "isos");
```

Beschreibung: *PunktLayer* enthält Einpunktpolygone, die jeweils eine Höhe besitzen, die im NumAttribut abgelegt ist (siehe PolyAttr()). Aus dieser XYZ-Punktwolke werden Isoflächen berechnet.

Aus der Punktmenge wird mittels der Delaunay-Triangulierung ein Dreiecksnetz berechnet, aus dem wiederum **Isolinien** entstehen. Die Außenkanten des Dreiecksnetzes (die konvexe Hülle der Punktmenge) bilden den **Rand**. Dieser wird benötigt, um offene Isolinien zu schließen.

Von *von* bis *bis* werden in der Schrittweite *delta* Isolinien erzeugt.

Die Isoflächen werden anhand der Höhe ihrer unteren Grenze eingefärbt. Dazu werden generell die Farben 100 bis 200 verwendet. Damit Ergebnisse verschiedener Rechenläufe vergleichbar sind, orientiert sich die Zuordnung der Höhen zu den Farben nicht an der aktuellen Spanne $von - bis$ sondern an der von außen vorzugebenden Spanne $mincol - maxcol$. Alle Flächen außerhalb dieser Spanne werden schwarz und nicht gefüllt.

Siehe auch Isolinien() und AGSetShading().

Isolinien

Syntax: Isolinien (Layer L, Real von, Real bis, Real delta, String name) : Layer
L: ein PunktLayer
von: Minimale Isolinie
bis: Maximale Isolinie
delta: Isolinienraster
name: Name des entstehenden Layers

Beispiel:
```
Isos := Isolinien (PunktLayer, 0, 100, 10, "isos");
```

Beschreibung: Berechnet Isolinien (Linien gleicher Höhe). Grundlage ist eine Menge von Punkten, die als PunktLayer übergeben wird. Jedes Polygon des Layers muss aus einem Punkt bestehen. Die X- und Y-Koordinate des Punktes ist die Lagekoordinate und die Z-Koordinate die maßgebende Zahl für das Berechnen der Isolinien.

Die Isolinien werden ab der Höhe *von* in *delta*-Schritten bis zur Höhe *bis* berechnet.

name ist der Name, den der Layer erhalten soll.

Höhe kann z.B. die Niederschlagssumme einer Station, das Temperaturmittel oder die Grundwasserhöhe sein.

Die Berechnung der Isolinien basiert auf einem Dreiecksnetz, das automatisch berechnet wird. Man kann dieses Netz auch explizit mit der Funktion LayerNetz() berechnen.

Siehe auch Isoflaechen(), Polygon(), Koord(), GeoPoint() und WriteLayer().

Isotachen

Syntax: Isotachen (Layer L, Real min, Real max, Real delta, String name) : Layer
L: ein PunktLayer
min: Minimale Geschwindigkeit
max: Maximale Geschwindigkeit
delta: Geschwindigkeitenraster
name: Name des entstehenden Layers

Beispiel:
```
Isos := Isotachen (PunktLayer, 0, 100, 10, "isotachen");
```

Beschreibung: Berechnet Isotachen (Linien gleicher Geschwindigkeit). Grundlage ist eine Menge von Punkten, die als PunktLayer übergeben wird. Jedes Polygon des Layers muss aus einem Punkt bestehen. Die X- und Y-Koordinate des Punktes ist die Lagekoordinate und die Z-Koordinate die maßgebende Zahl für das Berechnen der Isotachen.

Die Isotachen werden ab der Geschwindigkeit *min* in *delta*-Schritten bis zur Geschwindigkeit *max* berechnet.

name ist der Name, den der Layer erhalten soll.

Der Algorithmus berechnet ein entspanntes Dreiecksnetz und bildet auch selbständig einen Rand, welcher nicht zwingend konvex ist. Dies kommt bei Flussbetten genau so vor. Da bei der Berechnung der Isotachen von Geschwindigkeitsmessungen an mehreren Messlotrechten ausgegangen wird, sind die Werte in einem Raster angeordnet, was der Algorithmus sich zu nutze macht, um das entspannte Netz und vor allem den Rand zu berechnen.

Die Farbverteilung in diesem Netz kann mit AGSetShading() bestimmt werden.

Siehe auch Isolinien(), Isoflaechen(), Polygon(), Koord(),GeoPoint() und WriteLayer().

IsReal

Syntax: IsReal(String s) : Bool
s: ein String, der eine Realzahl repräsentiert

Beispiel: `IF (IsReal(instr))`

Beschreibung: Liefert den Wert TRUE zurück, wenn der String *s* eine Realzahl enthält, sonst FALSE. Diese Funktion ist sinnvoll, um Benutzereingaben auf Gültigkeit zu überprüfen, bevor sie mit StrToReal() umgewandelt werden.

IsUnix

Syntax: IsUnix() : Bool

Beispiel: `IF (IsUnix())`

Beschreibung: Liefert den Wert TRUE zurück, wenn das Programm auf einem Unix-System läuft, sonst wird FALSE zurückgegeben.

Für AquaWeb-Server siehe auch IsUnixClient().

IsUnixClient

Syntax: IsUnixClient() : Bool

Beispiel: `IF (IsUnixClient())`

Beschreibung: Liefert den Wert `TRUE` zurück, wenn der Client auf einem Unix-System läuft, sonst wird `FALSE` zurückgegeben. Läuft das Programm nicht auf einem Server, wird der Wert von IsUnix() geliefert.

IsUntrusted

Syntax: IsUntrusted (Tupel t, String feld) : Bool
t: Tupel
feld: Name des Feldes im Tupel

Beispiel: `IF (IsUntrusted (t, "WERT"))`

Beschreibung: Zeigt an, ob das Feld mit Namen $feld$ des Tupels t einen ungewissen, oder einen vertrauenswürdigen Wert enthält.

Siehe auch SetUntrusted().

IsValid

Syntax: IsValid((...) p) : Bool
p: Parameter beliebigen Typs.

Beispiel: `IF (IsValid (zr1))`

Beschreibung: Liefert den Wert TRUE zurück, wenn p einen gültigen Wert hat, ansonsten FALSE. Diese Funktion findet ihren sinnvollen Einsatz vor allem bei Zeitpunkten und Zeitreihen.

IsWildcard

Syntax: IsWildcard (Tupel t, String feld) : Bool
t: Tupel
feld: Name des Feldes im Tupel

Beispiel: `IF (IsWildcard (t, "UHRZEIT"))`

Beschreibung: Überprüft, ob das Feld $feld$ des Tupels t Wildcard ist.

Siehe SetWildcard().

IsZD

Syntax: IsZD (String s) : Bool
s: ein String, der eine Zeitdistanz enthalten könnte

Beispiel: `IF (IsZD(instr))`

Beschreibung: Liefert den Wert TRUE zurück, wenn der String *s* eine gültige Zeitdistanz enthält, sonst FALSE. Diese Funktion ist sinvoll, um Benutzereingaben auf Gültigkeit zu überprüfen, bevor sie in eine Zeitdistanz umgewandelt werden.

IsZP

Syntax: IsZP (String s) : Bool
s: ein String, der einen Zeitpunkt enthalten könnte

Beispiel: `IF (IsZP(instr))`

Beschreibung: Liefert den Wert TRUE zurück, wenn der String *s* einen gültigen Zeitpunkt enthält, sonst FALSE. Diese Funktion ist sinvoll, um Benutzereingaben auf Gültigkeit zu überprüfen, bevor sie in einen Zeitpunkt umgewandelt werden.

JaehrlicheSerie

Syntax: JaehrlicheSerie(ZR zr, ZR tagesw, Intervall bereich, Bool temp, [,Bool kal]) : ZR
zr: Zeitreihe (kontinuierlich oder 5-Minuten-Summen)
tagesw: Zeitreihe mit Tageswerten
bereich: Auswertungszeitraum
temp: Temporärflag
kal: optional, Kalenderjahr oder WWJ (Voreinstellung FALSE=WWJ)

Beispiel: `sserie := JaehrlicheSerie(zr, zrt, dekade, false)`

Beschreibung: Erzeugt die jährliche Serie der jeweiligen Niederschlags-Zeitreihe. Alle bestehenden Werte in der Serie werden gelöscht. Ist die jährliche Serie nicht vorhanden, dann wird sie angelegt. Das Attribut Ort ergibt sich aus dem Ort der Zeitreihe *zr*. Das Attribut Aussage wird auf jSe gesetzt.

Die jährliche Serie ist eine Momentan-Zeitreihe mit 21 Qualitätsschichten. Jede Schicht enthält die Menge der Extremwerte mit Zeitbezug zu einer Dauerstufe. Die Dauerstufe ist als Klartext zum ersten Zeitpunkt abgelegt. Dauerstufen sind fest: 5, 10, 15, 20, 30, 45, 60 und 90 Minuten, 2, 3, 4, 6, 9, 12 und 18 Stunden und 1, 2, 3, 4, 5 und 6 Tage.

Es wird pro Jahr ein Maximalwert in die jährliche Serie übernommen. Ein **Jahr** ist dabei ein Wasserwirtschaftsjahr. Wird *kal* jedoch auf TRUE gesetzt, dann ist ein **Jahr** ein Kalenderjahr.

Zur Berechnung siehe [3] und PartielleSerie. Für die weitere Bearbeitung siehe Verteilungsparameter.

Jahr

Syntax: Jahr (Zeitpunkt zp) : Real
zp:

Beispiel:
```
j := Jahr (tmpzp)
```

Beschreibung: Liefert das Jahr eines Zeitpunkts. Ein Zeitpunkt kann zwischen dem 1.1.1801 und dem 1.1.2450 00:00:00 liegen.

Siehe auch Monat().

KalibZR

Syntax: KalibZR (ZR zr, Intervall be, ZR kzr, Real vqual, R bqual, Bool neu, Bool diff) : ZR
zr: Ausgangsreihe
be: Berechnungszeitraum
kzr: Momentan-Reihe mit Kalibrierungspunkten
vqual: Ausgangsqualität aus *zr*
bqual: Zielqualität
neu: True=neue Reihe anlegen
diff: False=Quotienten, True=Differenzen, siehe unten

Beispiel:
```
zr := KalibZR (zr,bereich, kalibzr, 0, 1, False, False)
```

Beschreibung: Kalibriert die Y-Werte eine Reihe mit Hilfe einer Kalibrierungs-Reihe. Die zu kalibrierenden Daten werden der Qualität *vqual* der Reihe *zr* entnommen. Das Ergebnis wird entweder in die Qualität *bqual* derselben Reihe, oder in einer neuen, temporären Reihe abgelegt.

Die Reihe, in die die kalibrierten Daten geschrieben werden, wird als Ergebnis zurückgeliefert.

kzr enthält Kalibrierungspunkte (Kontrollwerte). Diese sind als Sollwerte aufzufassen. Sie liegen in Qualität 2 von *kzr*. Die Ist-Werte werden *zr* entnommen. Es ist aber auch möglich, Ist-Werte in *kzr* zu hinterlegen. Diese werden in Qualität 1 abgelegt. Sie werden herangezogen, wenn *zr* zu einem Soll-Wert keine Daten findet (wenn *zr* dort also eine Lücke aufweist). Da die Ist-Werte in *zr* an dem Zeitpunkt aufgesucht werden, an dem die Sollwerte in *kzr* vorliegen, ist sicherzustellen, dass *zr* keinen Zeitfehler gegenüber *kzr* aufweist. Gegebenenfalls müssen beide Reihen vorher aufeinander synchronisiert werden.

Kalibriert wird immer zwischen zwei Kalibrierungspunkten. Wenn links oder rechts von *be* kein Kalibrierungspunkt vorhanden ist, wird an der entsprechenden Seite von *be* ein Kalibrierungspunkt angenommen, der auf sich selbst verweist. Die Daten aus *zr* werden wertemäßig so gestaucht oder gestreckt und verschoben, dass alle Kalibrierungspunkte exakt passen.

Es kann auf zwei Weisen kalibiriert werden, mit Quotienten oder mit Differenzen. Quotienten sind sinnvoll, wenn der Fehler, den man durch das Kalibrieren verbessern will, gradueller Natur ist, z.B. durch allmähliches Zusetzen eines Fühlers oder durch temperatur- oder luftfeuchtebedingte Beeinflussung des Messsystems. Differenzen sind probat, wenn ein Versatz vorliegt, z.B. durch Abrutschen des Seils von einer Umlenkrolle. Es wird also entweder der Quotient Ykzr/Yzr oder die Differenz Ykzr-Yzr gebildet. In beiden Fällen wird dieser Korrekturwert zwischen den Kalibrierpunkten lienar interpoliert.

Siehe auch SyncZR().

Karte

Syntax: Karte (String name) : Karte
name: Name der Karte

Beispiel:
```
Map := Karte ("Waldviertel")
```

Beschreibung: Erzeugt eine neue Karte und gibt diese zurück. Eine Karte dient dazu, eine Liste von Layern aufzunehmen und Selektionen von Polygonen zu verwalten.

Neue Layer können an die Karte mit dem +-Operator angehängt werden:

```
Map := Map + EinLayer
```

Siehe auch Layer() und Polygon().

KarteAttr

Syntax: KarteAttr(Karte K, String attr) : String
K: Ein Karte
attr: Name des Attributs

Beispiel:
```
gax := KarteAttr (kart, "GKAchsen")
```

Beschreibung: Liefert das Attribut *attr* der Karte P. Ist *attr* kein Attribut einer Karte, dann wird ein Leerstring zurückgeliefert.

Ist das Attribut eine Zahl, kann man diese mittels StrToReal() aus dem Rückgabestring gewinnen.

Ist das Attribut ein Bool, dann werden die Strings `"TRUE"` bzw. `"FALSE"` geliefert.

Attribute sind:

- Name (siehe auch Name() und SetName())
- GKAchsen `"TRUE"` = statt eines einfachen Rahmens werden Gauß-Krüger-Achsen gezeichnet.
- Geograph `"TRUE"` = die Koordinatendarstellung während des Mausbewegens erfolgt in geographischen Koordinaten (Breite, Länge) statt in Gauß-Krüger-Koordinaten
- Dirty `"TRUE"` = markiert die Karte als komplett neu zu Zeichnende
- TexteFix `"TRUE"` = die Textgröße ist fix und nicht maßstabsgetreu

Siehe auch KarteSetAttr() und LayerAttr().

KarteBereichLU

Syntax: KarteBereichLU (Karte K) : GeoPoint
K: Eine Karte

Beispiel:
```
lu := KarteBereichLU (Kart)
```

Beschreibung: Liefert die linke untere Ecke des aktuell gesetzten Ausschnitts der Karte *K*.

Siehe auch KarteBereichRO() und KarteSetBereich().

KarteBereichRO

Syntax: KarteBereichRO (Karte K) : GeoPoint
K: Eine Karte

Beispiel:
```
ro := KarteBereichRO (Kart)
```

Beschreibung: Liefert die rechte obere Ecke des aktuell gesetzten Ausschnitts der Karte *K*.

Siehe auch KarteBereichLU() und KarteSetBereich().

KarteBlatt

Syntax: KarteBlatt (Karte K, Real massstab, Real breite, Real hoehe, String handle)
K: Eine Karte, die auf einem GeoCanvas dargestellt ist
massstab: Maßstabszahl, z.B. 50000
breite: Breite des Blattes in cm
hoehe: Höhe des Blattes in cm
handle: Azurfunktion, die abschließend aufgerufen wird

Beispiel:
```
KarteBlatt (karte, 50000, 100, 60, "JetztDrucken")
```

Beschreibung: Diese Funktion dient zur Vorauswahl des Bereichs einer Karte durch den Benutzer, der auf ein Blatt passt, wenn dieses zu klein ist, um die gesamte Karte aufzunehmen. Die Blattgröße wird durch *breite* und *hoehe* angegeben. Die Ränder und gegebenenfalls die Breite des Gauß-Krüger-Rahmens müssen schon vorher abgezogen worden sein. Bei Ausgabe in der Ausrichtung Portrait ist *hoehe* größer als *breite*, bei der Ausgabe im Querformat (Landscape) ist *breite* größer als *hoehe*.

Nach Beendigung der Funktion wechselt der GeoCanvas in den Zustand des Bereichwählens. Der Umriss des Blattes wird als grüner Rahmen auf dem GeoCanvas dargestellt. Ein erneutes Aufrufen mit veränderten Parametern bewirkt eine Verändertung des grünen Rahmens.

Der Benutzer kann nun den Rahmen auf dem GeoCanvas mit der Maus bewegen. Hat er so einen Bereich gewählt, schließt er die Aktion durch Drücken der linken Maustaste ab. Daraufhin wird die Azurfunktion *handle* aufgerufen, der die Karte und der gewählte Ausschnitt als Parameter übergeben werden. Die Parameter der Funktion müssen so definiert sein:

```
JetztDrucken (Karte AktMap, GeoPoint KarteLU, GeoPoint KarteRO)
```

Um die Aktion abzubrechen, das heißt den Zustand des Bereichwählens vorzeitig zu beenden, wird die Funktion mit *massstab* 0 aufgerufen.

Siehe auch NewGeoCanvas(), KarteOnPage() und PlotKarte().

KarteOnPage

Syntax: KarteOnPage(Page seite, Karte K, GeoPoint lu, GeoPoint ro, Bool mitgkax)
seite: Eine Reportseite
K: Eine Karte
lu: Punkt in cm auf der Seite links unten
ro: und rechts oben
mitgkax: sollen Gauß-Krüger-Achsen gezeichnet werden?

Beispiel:
```
KarteOnPage (seite, Kart, {2,2}, {20,20}, TRUE)
```

Beschreibung: Zeichnet die Karte K im gegenwärtig gesetzten Ausschnitt (siehe KarteSetBereich()) auf die Reportseite *seite*. *lu* und *ro* legen die Lage der Karte auf dieser Seite in cm fest. Sie beschreiben dabei die Lage des Darstellungsbereichs ohne Achsen.

Wenn die Karte nicht aus einer Szenerie-Datei geladen wurde (siehe ReadKarte()), die den darzustellenden Ausschnitt vorgibt, so muss der Ausschnitt vorher mittels KarteSetBereich() oder KarteSetVoll() gesetzt werden.

Ist *mitgkax* `TRUE`, dann wird die Karte mit einem Gauß-Krüger-Rahmen umgeben, sonst wird keine Umrandung gezeichnet.

Siehe auch PlotKarte(), Karte() und ReadKarte().

KarteSelect

Syntax: KarteSelect(Karte K, Bool onoff)
K: Eine Karte
onoff: `TRUE` oder `FALSE`

Beispiel:
```
KarteSelect (karte, TRUE)
```

Beschreibung: Ist $onoff$ `TRUE`, dann werden alle Polygone aller aktiven (`@Active`) Layer der Karte selektiert. Ist $onoff$ `FALSE`, dann werden alle Polygone deselektiert.

Siehe auch PlotKarte(), Selection() und PolySelect().

Siehe besonders auch SelectPolys().

KarteSetAttr

Syntax: KarteSetAttr(Karte K, String attr, String wert)
K: Ein Karte
attr: Name des Attributs
wert: neuer Wert des Attributs

Beispiel:
```
KarteSetAttr (K, "GKAchsen", "TRUE")
```

Beschreibung: Setzt das Attribut *attr* der Karte *K* auf den Wert *wert*. Ist *attr* kein Attribut einer Karte, dann hat diese Funktion keine Wirkung.

Ist das Attribut eine Zahl, muss diese mittels Str() in einen String gewandelt werden.

Ist das Attribut ein Bool, dann werden die Strings `"TRUE"` bzw. `"FALSE"` benutzt.

Zu den Attributen siehe KarteAttr().

Siehe auch LayerSetAttr().

KarteSetBereich

Syntax: KarteSetBereich(Karte K, GeoPoint lu, GeoPoint ro)
K: Eine Karte
lu: Punkt in m auf der Seite links unten
ro: und rechts oben

Beispiel:
```
KarteSetBereich (Kart, {2520000,5635000}, {2540000,5650500})
```

Beschreibung: Setzt den aktiven Ausschnitt der Karte. Dieser Ausschnitt wird ausgewertet, wenn die Karte gezeichnet wird.

Siehe auch KarteSetVoll(), KarteOnPage(), Karte() und ReadKarte().

KarteSetVoll

Syntax: KarteSetVoll(Karte K)
K: Eine Karte

Beispiel: `KarteSetVoll (Kart)`

Beschreibung: Setzt den aktiven Ausschnitt der Karte auf deren maximale Ausdehnung. Dieser Ausschnitt wird ausgewertet, wenn die Karte gezeichnet wird.

Siehe auch KarteSetBereich(), PlotKarte() und KarteOnPage().

KeyComplete

Syntax: KeyComplete(Tupel t) : Bool
t: Tupel

Beispiel: `IF (NOT KeyComplete(tup))`

Beschreibung: Zeigt an, ob alle Key-Felder des Tupels ausgefüllt sind. Ausgefüllt ist ein Feld dann, wenn es auf einen gültigen Wert gesetzt wurde. Nach dem Anlegen eines Tupels (siehe Tupel()) sind alle Felder unausgefüllt.

Siehe auch KeyUnique().

KeyUnique

Syntax: KeyUnique(Relation R, Tupel t) : Bool
R: Relation
t: Tupel

Beispiel: `IF (KeyUnique(R, t))`

Beschreibung: Diese Funktion erfüllt zwei Aufgaben. Diese unterscheiden sich dadurch, ob t aus R stammt oder nicht.

Stammt t aus R (ist also die Instanz von t in R enthalten), dann prüft die Funktion, ob es noch weitere Tupel in R gibt, die den gleichen Key haben.

Stammt t nicht aus R, so prüft die Funktion, ob in R bereits ein Tupel mit dem Key von t enthalten ist.

Siehe auch KeyComplete().

Kommentar

Syntax: Kommentar (ZR z) : String
z:

Beispiel:
```
komm := Kommentar (z)
```

Beschreibung: Liefert das Attribut Kommentar der Zeitreihe

Koord

Syntax: Koord (ZR z) : GeoPoint
z:

Beispiel:
```
point := Koord (z)
```

Beschreibung: Die geografische Position der Zeitreihe (Gauß-Krüger-Koordinate). Siehe dazu auch GeoPoint(), XKoo() und YKoo().

Korrelation

Syntax: Korrelation (ZR zr1, ZR zr2, Intervall bereich) : Real
zr1:
zr2:
bereich: Auswertungszeitraum

Beispiel:
```
korkoeff := Korrelation (zr1, zr2, bereich)
```

Beschreibung: Berechnet den Korrelationskoefizienten der Zeitreihen $zr1$ und $zr2$. Mit der Angabe von *bereich* wird die Stichprobe festgelegt. Die Zeitreihen müssen äquidistante Intervall-Zeitreihen sein (diskrete Zufallsvariable) mit gleicher Intervallblockung. Siehe auch Mittel(), Median(), Varianz() und Covarianz().

Korrelogramm

Syntax: Korrelogramm (ZR z1,ZR z2, Intervall bereich, Real texte, String form) : ZR
z1: Intervall-Reihe
z2: Intervall-Reihe
bereich: Auswertungsbereich
texte: 0 oder 100, wieviele Punkte sollen beschriftet werden
form: Format der Beschriftungen

Beispiel:
```
kg := Korrelogramm (zr1, zr2, dekade, 100, "#d.#m.#Y")
```

Beschreibung: Das Korrelogramm der Zeitreihen $z1$ und $z2$ wird berechnet. Diese müssen Intervall-Reihen mit gleicher Intervallbreite sein. Das Ergebnis ist eine Real-Momentan-Reihe, die mit ZRInAxBox() in einer AxBox als Korrelogramm dargestellt werden kann.

Zu jedem zeitgleichen Y-Paar aus den beiden Zeitreihen wird ein Punkt erzeugt, X aus $z1$ und Y aus $z2$. Der jeweilige X-Wert zu den Y-Werten gibt die Reihenfolge der Punkte vor.

texte gibt an, ob die jeweiligen X-Werte Texte an die Punkte gezeichnet werden sollen: 0 keine Texte, 100 alle Texte. Bei Werten kleiner 100 werden entsprechend weniger Punkte beschriftet. Das Format, in dem die X-Werte ausgegeben werden, wird durch $form$ festgelegt (siehe zpmode() oder RealFormat()).

Siehe auch SetPunktArt() und SetSymbolTyp().

Die erzeugte Reihe ist temporär und muss in einer AxBox dargestellt werden, um als Korrelogramm zu erscheinen.

KSTest

Syntax: KSTest(ZR zre, ZR zrv, Real alpha) : Tupel
zre: beobachtete (empirische) Verteilung
zrv: theoretisch erwartete Verteilung
alpha: Signifikanzniveau (0,1)

Beispiel: `tup := KSTest(zrem, gumbelzr, 0.05)`

Beschreibung: Testet die Anpassung einer beobachteten an eine theoretisch erwartete Verteilung mittels des *Kolmogoroff-Smirnoff*-Tests. (Siehe [6]). Getestet wird, ob die theoretisch erwartete Verteilung als ungeeignet verworfen werden muss.

Das Ergebnis des Tests ist ein Tupel, das zwei Felder enthält: Das Boolfeld abgelehnt, das anzeigt, ob die Verteilung statistisch signifikant abgelehnt werden muss (`TRUE`), oder nicht abgelehnt werden kann (`FALSE`), und das Zahlfeld Dmax, das die maximale Abweichung enthält.

alpha (zwischen 0 und 1) legt das Niveau der Signifikanz des Tests und damit eine Grenze für Dmax fest. Je kleiner *alpha* ist, desto sicherer ist die Aussage des Tests.

Für ausreichend große Stichproben (mehr als 35 Werte), ist *alpha* frei wählbar (sinnvolle Werte liegen zwischen 0.001 und 0.2). Ist die Stichprobe kleiner, ist die zugrunde liegende Funktion zur Berechnung der Grenze für Dmax nicht anwendbar, die Werte werden dann einer Tabelle entnommen. *alpha* darf in diesem Fall nur die Werte 0.2, 0.1, 0.05, 0.02 und 0.01 annehmen. Für Tafelwerte und die Bestimmung von Dmax siehe [7].

Da der Ursprung der theoretisch erwarteten Verteilung *zrv* eine partielle Serie sein kann, ist die Größe der zugrunde liegenden Stichprobe nicht aus den Daten ermittelbar. Sie muss deshalb im Attribut Messgenau abgelegt sein. Die Anzahl der Jahre ergibt sich bei beiden Reihen aus den Attributen GueltVon und GueltBis.

Siehe auch Verteilung().

KubischerSpline

Syntax: KubischerSpline (QL qf, Real ftoleranz, Real gewicht) : QL
qf: Quantenfolge mit Punktquanten
ftoleranz: zum Linearisieren
gewicht: Maß zum Abwägen zwischen interpolierend oder approximierend

Beispiel: `kubisch := KubischerSpline(qf, 0.05, 10)`

Beschreibung: Aus den Datenpunkten in qf wird ein kubischer **approximierender** Spline berechnet. Dieser Spline wird immer an vorgegebene Randsteigungen angepasst. Der Spline wird zwischen den zweiten und vorletzten Punkt in qf gelegt. Aus dem ersten und letzten wird die linke bzw. rechte Randsteigung (Randableitung) berechnet. Ist die Steigung des Rands nicht bekannt, so fügt man links und/oder rechts einen Lücke-PunktQuant an. Die Funktion berechnet dann die Randsteigung aus dem 2. und 3. bzw. vorletzten und vorvorletzten Punkt.

Mit *gewicht* kann man entscheiden, ob der Algorithmus eher interpolierend arbeitet ($gewicht \to \infty$) oder die Punkte optimal mittelt ($gewicht \to 0$). Das Gewicht darf jedoch nicht 0 oder < 0 sein! Man gibt mit dem Gewicht also gleichsam an, wie weit man bereit ist, die Extremwerte ausschlagen zu lassen zu Gunsten des genauen Treffens der vorgegebenen Punkte. In der Praxis wird es ratsam sein, den Benutzer durch grafisch-interaktives Manipulieren des Gewichtes, dieses bestimmen zu lassen.

Siehe dazu [11].

Die $ftoleranz$ wird benötigt, um die kubischen Verläufe des Splines zu linearisieren.

Siehe auch AkimaSpline() (**interpolierender** Spline).

LambertToGrad

Syntax: LambertToGrad (GeoPoint p, Bool nord) : GeoPoint
p: Punkt in Gauß-Krüger-Koordinaten
nord: True=Algerien Nord, False= Algerien Süd

Beispiel:
```
p2 := LambertToGrad(p)
```

Beschreibung: Berechnet die geografischen Koordinaten (Länge, Breite) aus dem Punkt p, der in Lambert-Koordinaten angegeben ist.

Zugrunde liegt der Ellipsoid nach Clarke,1880. Die Bezugsbreiten (Schnittkegel, **nicht** Berührkegel) sind fest vorgegeben und passen auf den Staat Algerien. Dieser ist in eine nördliche und eine südliche Zone eingeteilt.

Siehe auch GradToLambert().

Zur Transformation in andere Koordinatensysteme siehe auch GKToGrad() und UTMToGrad().

LastTupel

Syntax: LastTupel(Relation R [, String idxfeld]) : Tupel
R: eine Mem-Relation oder eine UVS-Relation
optional: idxfeld: maßgebliches Indexfeld

Beispiel:
```
t := LastTupel(Stamm, "ORT")
```

Beschreibung: Liefert das letzte Tupel der Relation R zurück. Das für die Sortierung maßgebliche Feld wird mit $idxfeld$ angegeben. Ohne Angabe eines Indexfelds wird der Standardindex benutzt (siehe CreateIndex()) oder, wenn dieser nicht erstellt wurde, die unsortierte Reihenfolge.

Diese Funktion ist nur für Mem-Relationen oder UVS-Relationen zu verwenden!

Siehe auch FirstTupel().

Layer

Syntax: Layer (String name) : Layer
name: Name des Layers

Beispiel:
```
L := Layer ("Bodenarten")
```

Beschreibung: Erzeugt einen neuen Layer und liefert diesen zurück. Ein Layer enthält, neben seinem Namen, eine Liste von Polygonen. Diese Liste ist nach der Erzeugung leer.

Neue Polygone können an den Layer mit dem +-Operator angehängt werden:

```
L := L + poly
```

Siehe auch Polygon() und Karte().

LayerAnzPoly

Syntax: LayerAnzPoly (Layer L) : Real
L: Ein Layer

Beispiel:
```
anz := LayerAnzPoly (lay)
```

Beschreibung: Liefert die Anzahl der Polygone im Layer.

Siehe auch Layer().

LayerAttr

Syntax: LayerAttr(Layer L, String attr) : String
L: Ein Layer
attr: Name des Attributs

Beispiel:
```
farbe := LayerAttr (lay, "Farbe")
```

Beschreibung: Liefert das Attribut *attr* des Layers *L*. Ist *attr* kein Attribut eines Layers, dann wird ein Leerstring zurückgeliefert.

Ist das Attribut eine Zahl, kann man diese mittels StrToReal() aus dem Rückgabestring gewinnen.

Ist das Attribut ein Bool, dann werden die Strings `"TRUE"` bzw. `"FALSE"` geliefert.

Attribute sind:

- Name (siehe auch Name() und SetName())
- Farbe
- LabelSize Größe der Label in cm
- TexteAn `"TRUE"` = die Label der Polygone im Layer werden mitgezeichnet
- Active `"TRUE"` = der Layer ist zum Klicken aktiviert
- Hidden `"TRUE"` = der Layer wird nicht gezeichnet

Siehe auch LayerSetAttr() und PolyAttr().

LayerNetz

Syntax: LayerNetz (Layer L, Bool nursel) : Layer
L: ein PunktLayer
nursel: nur selektierte Polygone(Punkte) werden berücksichtigt

Beispiel:
```
netz := LayerNetz (PunktLayer, False);
```

Beschreibung: Berechnet ein Dreiecksnetz. Die Punkte in L werden dazu trianguliert. Wenn *nursel* True ist, werden nicht alle Punkte, sondern nur die selektierten herangezogen.

Siehe auch Isolinien().

LayerSetAttr

Syntax: LayerSetAttr(Layer L, String attr, String wert)
L: Ein Layer
attr: Name des Attributs
wert: neuer Wert des Attributs

Beispiel:
```
LayerSetAttr (L, "Farbe", "Rot")
```

Beschreibung: Setzt das Attribut *attr* des Layers *L* auf den Wert *wert*. Ist *attr* kein Attribut eines Layers, dann hat diese Funktion keine Wirkung.

Ist das Attribut eine Zahl, muss diese mittels Str() in einen String gewandelt werden.

Ist das Attribut ein Bool, dann werden die Strings `"TRUE"` bzw. `"FALSE"` benutzt.

Zu den Attributen siehe LayerAttr().

Siehe auch PolySetAttr() und KarteSetAttr().

Lebenslauf

Syntax: Lebenslauf (ZR z) : String
z:

Beispiel:
```
l := Lebenslauf (z)
```

Beschreibung: Liefert den Lebenslauf (die Entstehungsgeschichte) der Zeitreihe.

Siehe auch ZRModifyInfo().

Length

Syntax: Length(String s) : Real
s:

Beispiel:
```
len := Length (s)
```

Beschreibung: Gibt die Anzahl der Zeichen eines Strings zurück. Das erste Zeichen eines Strings hat die Position 0, das letzte die Position `Length(s)`-1.

LinesOnPage

Syntax: LinesOnPage (Page page) : Real
page:

Beispiel: `lines := LinesOnPage (page)`

Beschreibung: Ermittelt die Anzahl der Zeilen auf einer Page. Diese ist abhängig von den bei NewPage() übergebenen Parametern. Siehe auch NewPage().

Lingua

Syntax: Lingua () : String

Beispiel: `sprache := Lingua()`

Beschreibung: Liefert die Sprache, in der das Benutzer-Interface erscheint.

Siehe SetLingua().

LinguaSort

Syntax: LinguaSort (Array A) : Array

Beispiel: `A2 := LinguaSort(A)`

Beschreibung: Sortiert das Array *A* entsprechend der Reihenfolge der Texte in der aktuell gesetzten Sprache (siehe SetLingua()). Dazu werden die Keys des Array angepasst. Die alten Keys gehen verloren.

Um die neue Reihenfolge zu nutzen, muss das Feld nach Key sortiert durchlaufen werden. Beispiel:

```
SetLingua ("FR")
A := StrSplit ("Abfluss Hund Katze Pferd")
A2 := LinguaSort(A)
FORALL s IN A2 ("KEY")
print (s)
ENDFOR
```

ergibt die Ausgabe

```
Katze
Pferd
Hund
Abfluss
```

Siehe SetLingua().

LinienQuant

Syntax: LinienQuant(Intervall xbereich, Real ylinks, Real yrechts) : Quant
xbereich: Intervall des Quants
ylinks: linker y-Wert
yrechts: rechter y-Wert

Beispiel: `q := LinienQuant( [von,bis], 10,20 )`

Beschreibung: Erzeugt ein LinienQuant, das sich über das Intervall *xbereich* erstreckt. Der linke y-Wert wird auf *ylinks* gesetzt und der rechte auf *yrechts*.

Links

Syntax: Links (Intervall i) : Zeitpunkt
i:

Beispiel: `zpunkt := Links (i)`

Beschreibung: Liefert die linke Seite des (Zeit-)Intervalls *i*.

LinksReal

Syntax: LinksReal (Intervall i) : Real
i:

Beispiel: `rpunkt := LinksReal (i)`

Beschreibung: Liefert die linke Seite des (Real-)Intervalls i.

LnGamma

Syntax: LnGamma(Real x) : Real
x:

Beispiel: `a := LnGamma( 4.5 )`

Beschreibung: Liefert den natürlichen Logarithmus der Gammafunktion an der Stelle x. x muss größer 0 sein.

Der Algorithmus berechnet den Funktionswert mit 14 Stellen Genauigkeit nach Lanczos (siehe Lanczos, C. A precision approximation of the gamma function, J. SIAM Numer. Anal., B, 1, 86-96, 1964.)

LockTupel

Syntax: LockTupel(Relation R, Real num) : Bool
R: dbf-Relation
num: Nummer des Tupels in der Relation (0 ..)

Beispiel: `ok := LockTupel(Stamm, 100)`

Beschreibung: Das Tupel Nummer *num* der dbf-Relation R wird gelockt. Schreibzugriffe auf dieses Tupel sind nun nicht mehr möglich. Dieser Zustand kann mit der Funktion UnlockTupel() rückgängig gemacht werden.

Wenn das Tupel schon gelockt war, wird `FALSE` zurückgeliefert, sonst `TRUE`.

Siehe auch Relation() und TupRecNum().

Log

Syntax: Log (Real r) : Real
r:

Beispiel: `l := Log (r)`

Beschreibung: Berechnet den natürlichen Logarithmus von r.

LogTupImp

Syntax: LogTupImp (Tupel tup, String hinweis)
tup: Importtuel
hinweis: nur für Abstich: Hinweis-Text

Beispiel: `LogTupImp (tup, "Beeinflusst")`

Beschreibung: Diese Funktion wird von aqualog benutzt. Sie dient dazu, neue Daten, die in Tupelform vorliegen, in die Zeitreihe zu importieren. Anhand des Formats des Tupels wird der passende Parameter automatisch bestimmt. Importiert werden können Abstiche, Senkungen und Rohroberkanten.

LogUpdate

Syntax: LogUpdate (String station, String param)
station:
param:

Beispiel: `LogUpdate ("Rees", "Grundwasserstand")`

Beschreibung: Diese Funktion wird von aqualog benutzt. Sie dient dazu, die abhängigen Zeitreihen zu aktualisieren, wenn sich die Ausgangszeitreihen geändert haben.

Mögliche Werte für *param* sind

- Grundwasserstand: berechnet sich aus Rohroberkante und Abstich.
- Rohroberkante: auf Grundlage der Qualität 0 (Messungen) und der Senkungen (wenn vorhanden), wird die angepasste Rohroberkante in Qualität 1 berechnet.
- GOK: Geländeoberkante, entsprechend Rohroberkante
- Flurabstand: berechnet sich aus GOK und Grundwasserstand
- Kellerhöhe: wird auf Grundlage von GOK und der individuellen Höhe eines Kellers berechnet.

Die Aktualisierung erfolgt auf dem gesamten Zeitbereich.

LokaleMaxima

Syntax: LokaleMaxima (ZR z, Intervall b, Distanz tol, Bool tmp) : ZR
z:
b: Berechnungszeitraum
tol: Toleranz (in X-Richtung)
tmp: Temporärflag

Beispiel:
```
szr := LokaleMaxima (wzr, WWJ(1995), ~"1 Stunde", FALSE)
```

Beschreibung: Berechnet die lokalen Maxima der Zeitreihe. Diese werden als Momentan-Zeitreihe abgespeichert. Der Parameter *tol* setzt eine Toleranzbreite, die kleine Schwankungen unberücksichtigt lässt (zum Algorithmus siehe LokaleMinima()).

Die Ergebniszeitreihe erbt alle Attribute der Ausgangszeitreihe mit Ausnahme von Aussage, welche auf LMA gesetzt wird.

LokaleMinima

Syntax: LokaleMinima (ZR z, Intervall b, Distanz tol, Bool tmp) : ZR
z:
b: Berechnungszeitraum
tol: Toleranz (in X-Richtung)
tmp: Temporärflag

Beispiel:
```
szr := LokaleMinima (wzr, WWJ(1995), ~"1 Stunde", FALSE)
```

Beschreibung: Berechnet die lokalen Minima der Zeitreihe. Diese werden als Momentan-Zeitreihe abgespeichert. Der Parameter *tol* setzt eine Toleranzbreite, die kleine Schwankungen unberücksichtigt lässt. Innerhalb eines *tol* breiten Bereichs wird nach dem kleinsten lokalen Minimum gesucht. Wird eins gefunden, wird es eingetragen. Darauf wird dann vom gefundenen Zeitpunkt aus wieder ein *tol* breiter Bereich betrachtet, usw.

Die Ergebniszeitreihe erbt alle Attribute der Ausgangszeitreihe mit Ausnahme von Aussage, welche auf LMI gesetzt wird.

LowCase

Syntax: LowCase (String s) : String
s:

Beispiel:
```
Print LowCase ("AZUR")
```
$\Longrightarrow$ `azur`

Beschreibung: Wandelt alle Großbuchstaben des Strings *s* in Kleinbuchstaben um.

LueckenAnteile

Syntax: LueckenAnteile (ZR z, Intervall i, Distanz d, Bool b) : ZR
z:
i: Berechnungszeitraum
d: Breite der Intervalle, die ausgewertet werden
b: Temporärflag

Beispiel: `anteile := LueckenAnteile( z, i, d, b )`

Beschreibung: Arbeitet entsprechend IntervallMittel, die Y-Werte geben jedoch den Anteil von Lücken auf den Intervallen in Prozent an.

LueckenProz

Syntax: LueckenProz (ZR reihe, Intervall i) : Real
reihe:
i: Berechnungszeitraum

Beispiel: `r := LueckenProz (reihe, breite)`

Beschreibung: Ermittelt den Anteil der Lücken der Zeitreihe z über dem Zeitraum i in Prozent.

LueckenReihe

Syntax: LueckenReihe (ZR z, Intervall i, Real ywert, Bool b) : ZR
z:
i: Berechnungszeitraum
ywert: Werte, der für Nicht-Lücke-Zeiträume eingesetzt wird
b: Temporärflag

Beispiel: `anteile := LueckenReihe (zr, bereich, 0.0, FALSE)`

Beschreibung: Erzeugt eine Reihe, die dort, wo zr Lücke ist, ebenfalls Lücke ist, und sonst den Wert *ywert* annimmt.

MakeDir

Syntax: MakeDir (string name)
name: Name des neuen Verzeichnisses

Beispiel: `MakeDir("daten")`

Beschreibung: Erzeugt ein neues Verzeichnis *name*. Ist *name* schon vorhanden oder sind keine ausreichenden Zugriffsrechte vorhanden, so schlägt diese Aktion fehl. Das Vorhandensein eines Verzeichnisses kann man mit der Funktion DirList() überprüfen.

Siehe auch ChangeDir() und Remove().

MakeURLParams

Syntax: MakeURLParams (Array A) : String
A: Parameter

Beispiel: `url := MakeURLParams (A))`

Beschreibung: Erzeugt den Parameteranteil einer URL aus A. Das Fragezeichen wird nicht erzeugt. Sonderzeichen werden umgewandelt.

Siehe auch URLParams().

Match

Syntax: Match (Tupel t, Tupel von [, Tupel bis]) : Bool
t: ein Tupel
von:
bis:

Beispiel: `IF (Match (t, muster))`

Beschreibung: Gibt an, ob das Tupel *t* auf das Muster passt, das entweder durch ein Mustertupel (*von*) oder durch zwei Mustertupel (*von* und *bis*) gegeben ist.

Wird nur *von* angegeben, dann werden die Tupel t und *von* auf Gleichheit getestet. Wildcards, wie z.B. `*hausen`, sind erlaubt. Unbesetzte Felder werden wie `*` behandelt.

Bei Angabe von *bis* wird ein Bereichsvergleich durchgeführt. Alle Tupel, die (lexigraphisch oder nummerisch) zwischen *bis* und *von* liegen, matchen. Es findet also der Vergleich: $von <= t <= bis$ statt.

Siehe auch DBFilter().

Max

Syntax: Max (ZR reihe, Intervall i) : Real
reihe:
i: Berechnungszeitraum

Beispiel:
```
r := Max (reihe, breite)
```

Beschreibung: Ermittelt das Maximum der Zeitreihe *reihe* über dem Zeitraum i.

MaxFocusZR

Syntax: MaxFocusZR (ZR reihe) : Intervall
reihe: Reihe, für die der Maxfocus bestimmt werden soll

Beispiel:
```
focus := MaxFocusZR (reihe)
```

Beschreibung: Ermittelt den Zeitbereich, auf dem die Reihe (Real-)Daten enthält.

Um den MaxFocus der Textwerte zu bestimmen, benutzt man MaxTextFocusZR().

MaxQualitaet

Syntax: MaxQualitaet (ZR z, Intervall bereich) : Real
z:
bereich:

Beispiel: `MaxQualitaet (zr1, WWJ(1989))`

Beschreibung: Liefert die maximale Qualität der Zeitreihe z, die auf *bereich* vorliegt. Siehe auch MaxFocusZR().

MaxTextFocusZR

Syntax: MaxTextFocusZR (ZR reihe) : Intervall
reihe: Reihe, für die der MaxTextFocus bestimmt werden soll

Beispiel: `tfocus := MaxTextFocusZR (reihe)`

Beschreibung: Ermittelt den Zeitbereich, auf dem die Reihe Textwerte enthält.

Siehe auch MaxFocusZR().

MaxZP

Syntax: MaxZP (ZR reihe, Intervall i) : Zeitpunkt
reihe:
i: Berechnungszeitraum

Beispiel: `zp := MaxZP (reihe, breite)`

Beschreibung: Ermittelt den Zeitpunkt des Maximums der Zeitreihe *reihe* über dem Zeitraum i.

MD5Sum

Syntax: MD5Sum(String s) : String
s : Ein String

Beispiel: `pointy := MD5Sum (user+":"+password)`

Beschreibung: Rechnet die MD5-Summe von *s* nach RFC1321 aus und gibt diese als Base64-String aus. Der String hat immer 22 Zeichen.

Siehe auch Base64ToStr() und StrToBase64().

Median

Syntax: Median (ZR reihe, Intervall i) : Real
reihe:
i: Berechnungszeitraum

Beispiel: `med := Median (reihe, breite)`

Beschreibung: Ermittelt den Median (Zentralwert) der Zeitreihe *reihe* über dem Zeitraum *i*. Die Zeitreihe muss eine Intervallzeitreihe sein. Siehe auch Mittel().

Messgenau

Syntax: Messgenau (ZR z) : Real
z:

Beispiel: `genau := Messgenau (z)`

Beschreibung: Liefert das Attribut Messgenau der Zeitreihe, Messgenauigkeit.

Min

Syntax: Min(ZR reihe, Intervall i) : Real
reihe:
i: Berechnungszeitraum

Beispiel: `r := Min (reihe, breite)`

Beschreibung: Ermittelt das Minimum der Zeitreihe *reihe* über dem Zeitraum *i*.

Minute

Syntax: Minute (Zeitpunkt zp) : Real
zp:

Beispiel: `m := Minute (tmpzp)`

Beschreibung: Liefert die Minute eines Zeitpunkts.

Siehe auch Sekunde().

MinZP

Syntax: MinZP (ZR reihe, Intervall i) : Zeitpunkt
reihe:
i: Berechnungszeitraum

Beispiel: `zp := MinZP (reihe, breite)`

Beschreibung: Ermittelt den Zeitpunkt des Minimums der Zeitreihe *reihe* über dem Zeitraum *i*.

Mittel

Syntax: Mittel (ZR reihe, Intervall i) : Real
reihe:
i: Berechnungszeitraum

Beispiel: `r := Mittel (reihe, breite)`

Beschreibung: Ermittelt das arithmetische Mittel der Zeitreihe *reihe* über dem Zeitraum *i*. Siehe auch Median().

Mod

Syntax: Mod(Real r, Real d) : Real
r: Dividend
d: Divisor

Beispiel:
```
rest := Mod (wert, 10)
```

Beschreibung: Liefert den Rest der Division r/d.

Sowohl Dividend als auch Divisor müssen nicht notwendigerweise ganzzahlig sein. Z.B. gilt: $Mod(31.5, 3.14) = 0.1$

ModificationTable

Syntax: ModificationTable (ZRList zrl, Zeitpunkt abwann) : Relation
zrl: eine Zeitreihenliste
abwann: ab welchem Zeitpunkt sollen Änderungen berücksichtigt werden

Beispiel:
```
rel := ModificationTable (zrl, @"Gestern")
```

Beschreibung: Erzeugt eine Relation, die zu den Zeitreihen in *zrl* mit Änderungen ein Tupel mit Änderungsbereichen enthält.

Die Tupel enthalten zu allen Monaten bzw. Tagen, an denen seit *abwann* Änderungen stattgefunden haben, ein Bool-Feld. Diese Felder sind nach den Jahren, Monaten bzw. Tagen benannt (z.B. 2006 für das gesamte Jahr 2006, 200601 für den Januar 2006 oder 20060215 für den 15. Februar 2006). Wenn in der betreffenden Zeitreihe auf dem entsprechenden Bereich seit *abwann* eine Änderung stattgefunden hat, dann enthält das Feld True, sonst False. Für Bereiche, auf denen bei keiner Zeitreihe in *zrl* eine Änderung stattgefunden hat, werden keine Felder erzeugt.

Jedes Tupel enthält ein Zahlfeld namens ZRID, das die Position der jeweiligen Zeitreihe in der Zeitreihenliste angibt (siehe ZRNr().

Die Entscheidung ob Monats- oder Tagesfelder erzeugt werden, richtet sich nicht nach dem Zeitreihen, da diese verschiedene Endmonate (die in Tagen aufgelöst sind) haben können. Sie richtet sich nach dem jüngsten Endmonat aller Reihen. Liegt dieser in der Zukunft, wird stattdessen der Anfang des laufenden Monats herangezogen.

Das Ergebnis ist eine Memory-Relation. Die maximale Anzahl Felder einer Memory-Relation beträgt 255. Daher verdichtet die Funktion die Felder solange, bis höchstens 255 Felder übrig bleiben. Dabei wird folgendermaßen vorgegangen:

Grundsätzlich wird (auch ohne Verdichtung) für Daten, die in der Zukunft liegen (alle ab dem Ende des laufenden Monats), nur ein String-Feld namens Zukunft erzeugt, das das Intervall, auf dem in der Zukunft Änderungen erfolgt sind, als Text enthält (Beispiel: 20070201000000-20500101000000). Von Links aus werden solange Monate zu Jahren zusammengefasst, bis die maximale Feldanzahl unterschritten ist. Dies erfolgt jedoch nur, wenn mindestens zwei Monate in einem Jahr vorhanden sind.

Siehe auch DoModRecording(), ModifiedSince() und ModTabCompact().

ModifiedSince

Syntax: ModifiedSince (ZR z, Zeitpunkt abwann, Real womit) : QuantList
z: eine Zeitreihe
abwann: ab welchem Zeitpunkt sollen Änderungen berücksichtigt werden
womit: Ersatzwert

Beispiel:
```
qf := ModifiedSince(zr1, last, 0)
```

Beschreibung: Liefert die Bereiche von *z* zurück, die seit *abwann* geändert wurden.

Geliefert werden Lücke-Quanten für Bereiche, die vor *abwann* oder nie geändert wurden, und Quanten mit Y-Wert *womit*, für Bereiche, die seit *abwann* geändert wurden.

Siehe auch DoModRecording() und ModificationTable().

ModTabCompact

Syntax: ModTabCompact (Relation modtab) : Array
modtab: eine Modifikationstabelle

Beispiel:
```
A := ModTabCompact (mt)
```

Beschreibung: Erzeugt aus einer mit ModificationTable() erzeugten Relation ein Array(), das als Indexe Intervall-Strings der Form JJJJMMTTHHMMSS-JJJMMTTHHMMSS enthält und als Werte Strings, die mit Komma getrennt die ZRIds aus *modtab* enthalten.

Die Intervall-Strings geben die Bereiche an, auf denen die Zeireihen zu den ZRIds Änderungen haben. Soweit möglich sind dabei mehrere Felder aus *modtab* zu einem Intervall zusammengefasst worden.

Mit den ZRIds indiziert man mittels ZRNr() die Zeitreihen-Liste, auf der *modtab* basiert.

Siehe auch ModificationTable().

MomentanQuant

Syntax: MomentanQuant(Zeitpunkt xpunkt, Real wert) : Quant
xpunkt: Zeitpunkt (oder Realpunkt) des Quants
wert: y-Wert

Beispiel:
```
q := MomentanQuant( @"23.5.1949", 40 )
```

Beschreibung: Erzeugt ein MomentanQuant zum Punkt *xpunkt*, dessen y-Wert auf *text* gesetzt wird.

Siehe auch TextQuant() und IntervallQuant().

Monat

Syntax: Monat (Zeitpunkt zp) : Real
zp:

Beispiel:
```
m := Monat (tmpzp)
```

Beschreibung: Liefert den Monat eines Zeitpunkts.

Siehe auch Tag().

MulGroesse

Syntax: MulGroesse (ZR z, String sg, Intervall i, String sp, Bool b) : ZR
z:
sg: Größe
i: Berechnungszeitraum
sp: Parameter der Ergebniszeitreihe
b: Temporärflag

Beispiel:
```
mul := MulGroesse (zr1,"42.3km^2",MAXFOCUS,"Abfluss",TRUE)
```

Beschreibung: Multipliziert die Zeitreihe mit der Größe *sg*, welche eine einheitbehaftete Zahl ist. Die Ergebniszeitreihe erhält *sp* als Parameter. Im oben aufgeführten Beispiel könnte $zr1$ z.B. der Niederschlag in mm/h sein. Die Verknüpfung der Einheiten findet automatisch statt.

MultiBox

Syntax: MultiBox (Array fragen, Real breite, Bool mitabbruch) : Array
fragen: Fragen und Vorbelegungen, mit ; getrennt
breite: Breite der Eingabefelder in Pixeln
mitabbruch: Erzeugt einen Abbruch Button

Beispiel:
```
feld := MultiBox(fundvor, 100, True)
```

Beschreibung: Erzeugt ein neues Fenster, welches zu jedem Eintrag in $fragen$ einen Fragetext und ein Eingabefeld erzeugt. Der Fragetext und die Vorbelegung pro Zeile sind mit ; getrennt als Inhalt der Arrayeinträge abgelegt. Zusätzlich wird ein OK-Button erzeugt.

Das AGWindow, aus dem diese Input-Box gestartet wurde, ist solange inaktiv, bis der Benutzer den OK-Button gedrückt hat oder in einem Eingabefeld Return drückt. Der Programmablauf wird solange angehalten.

Zurückgegeben wird ein Array, das die gleichen Keys enthält wie $fragen$, als Inhalt jedoch jeweils die Inhalte der Eingabefelder enthält.

Beispiel:

```
funda["a-von"] := "von:;1.5.1999"
funda["b-bis"] := "bis:;1.6.1999"
feld := MultiBox (funda, 100)
svon := feld["a-von"]
sbis := feld["b-bis"]
```

Zu beachten ist, dass die Reihenfolge der Zeilen sich **nicht** nach der Reihenfolge der Belegung des Arrays richtet, sondern nach der alphabetischen Reihenfolge der **Keys**.

Wird ein Abbruch-Button mit *mitabbruch*=True erzeugt und dieser dann gedrückt, so ist der Rückgabewert **invalid**.

Siehe auch OkBox(), InputBox(), ElementBox(), SelectBox() und ZIBox().

MulZR

Syntax: MulZR (ZR z1, ZR z2, Intervall i, String param, Bool b) : ZR
z1:
z2:
i: Berechnungszeitraum
param: Parameter der Ergebniszeitreihe
b: [**?**]

Beispiel:

```
mul := MulZR (z1, z2, i, s, b)
```

Beschreibung: Multipliziert die Zeitreihe $z1$ mit der Zeitreihe $z2$. Die Multiplikation erfolgt auch bei kontinuierlichen Zeitreihen mathematisch korrekt (siehe auch [**?**]. Die Berechnung der Ergebniseinheit findet automatisch statt.

MySQLRelation

Syntax: MySQLRelation (String name, String datenbank, String host, String uspw) : Relation
name: Name der Relation
datenbank: Name der Datenbank
host: Host, auf dem MySQL läuft
uspw: Username / Passwort

Beispiel:
```
R := MySQLRelation ("stammdat", "eglv", "birke", "aqua/aqua")
```

Beschreibung: Öffnet die MySQL-Relation mit Namen *name*. Ist keine Relation dieses Namens vorhanden, oder der *name* ein Leerstring, dann wird eine ungültige Relation erzeugt (siehe IsValid()).

datenbank ist der Name der Datenbank (database), in der die Relation (table) gespeichert ist. *host* ist der Name des Rechners, auf dem der MySQL-Server läuft. *uspw* enhält mit / getrennt Username und Passwort oder nur den Usernamen, wenn es kein Passwort gibt.

Siehe auch NewMySQLRelation().

Name

Syntax: Name((...) p) : String
p: Parameter beliebigen Typs.

Beispiel:
```
s := Name (axoben)
```

Beschreibung: Liefert den Namen eines (komplexen) Azur-Typs, wie AxBox, Relation, Karte, Polygon, Layer, Typen, die keinen Namen besitzen, liefern einen Leerstring.

Siehe SetName().

NEreignisse

Syntax: NEreignisse(ZR zr, Intervall bereich, Real M, Distanz mindau, Bool temp) : ZR
zr: Abfluss- oder Wasserstands-Zeitreihe
bereich: Auswertungszeitraum
M: langjähriges Mittel (Q oder W)
mindau: Mindestdauer eines Niedrigwasserbereiches
temp: Temporärflag

Beispiel: `zr2 := NEreignisse(zr, bereich, 0.35, ~"7 Tage", false)`

Beschreibung: Erzeugt eine Momentanzeitreihe mit Niedrigwasser-Ereignissen. Diese Ereignisse werden aus einer Abfluss- oder einer Wasserstands-Zeitreihe *zr* gebildet. *zr* kann eine kontinuierliche oder eine Intervall-Zeitreihe sein. Nach [1] muss eine Tagesmittelwert-Zeitreihe benutzt werden.

Ein Ereignis wird als solches identifiziert, wenn es in einem Bereich liegt, der mindestens *mindau* kleiner ist als M, und wenn es vom nächsten und vorigen Ereignis durch einen Bereich getrennt ist, der größer als M ist.

Diese Funktion ist implizit in der Berechnung von partiellen und jährlichen Serien enthalten (siehe NPartSerie() und NJahrSerie()).

Der Parameter M kann z.B. mit `M := Mittel (zr, MaxFocusZR(zr))` bestimmt werden. Diese Auswertung kann jedoch lange dauern, weshalb dieser Parameter auch explizit vorgegeben werden kann.

Die Ergebniszeitreihe erbt alle Attribute der Ursprungszeitreihe, mit Ausnahme von Aussage, welche auf NER gesetzt wird.

NewAGWindow

Syntax: NewAGWindow(String name, String titel, GeoPoint lo, GeoPoint ru [, Bool nq [, Bool sofort [,Bool nostatus]]])
name: Name des AGWindows
titel: Titelzeile
lo: Pixelkoordinate der linken oberen Ecke
ru: Pixelkoordinate der rechten unteren Ecke
nq: optional NoQuit (Voreinstellung FALSE)
sofort: optional Window ist sofort sichtbar (Voreinstellung TRUE)
nostatus: optional Es wird keine Statuszeile erzeugt

Beispiel:
```
NewAGWindow("AGPop2", "Einstellungen", {100,150}, {300,250})
```

Beschreibung: Erzeugt ein neues AGWindow mit dem Namen *name*. Der Text *titel* erscheint in der Kopfzeile. *lo* und *ru* geben die Koordinaten des neuen Fensters an.

Ist *nq* TRUE, dann wird kein Quitbutton erzeugt.

sofort ist veraltet und hat keine Auswirkung.

Nach der Erzeugung eines Windows muss zwingend SetAGWindow() folgen. Siehe auch SetAGWindow(), GetAGWindow() und DelAGWindow().

NewAuswahl

Syntax: NewAuswahl(String name, Real xpos, Real ypos, Bool vert, String azurprg, String s1, ...)
name: Name der Auswahl
xpos: Koordinaten in Pixeln, relativ zur linken
ypos: oberen Ecke des Windows
vert: TRUE=die Auswahl verläuft vertikal
azurprg: Azurprogramm oder ""
s1,...: Liste der einzelnen Auswahlwerte

Beispiel:
```
NewAuswahl( "Farbe", 20,270, FALSE, "", "Schwarz", "Rot", "Blau")
```

Beschreibung: Eine Auswahl aller aufgeführten Strings wird an die Stelle *xpos*, *ypos* platziert. Die Labels werden entweder nebeneinander (*vert* =FALSE) oder untereinander (*vert* =TRUE) angeordnet. Das *azurprg* wird aufgerufen, wenn der Benutzer die Auswahl ändert. Falls hier ein Leerstring angegeben wird, findet kein Azuraufruf statt.

Falls man außer *s*1 keinen weiteren String mehr übergibt, wird *s*1 als Liste von Token aufgefasst, die durch Leerzeichen getrennt sind.

Eine Auswahl wird in einer Azurfunktion als String empfangen. Dieser enthält den aktuell gewählten Wert.

Mit ExportVar() kann man die Auswahl umsetzen. Der zu übergebende String muss einer der Auswahlwerte sein. Bei Übergabe eines Leerstrings wird die Auswahl so eingestellt, dass kein Wert gewählt ist.

NewAxBox

Syntax: NewAxBox (String id) : AxBox
id: Name der AxBox

Beispiel:
```
ax := NewAxBox ("Box")
```

Beschreibung: Erzeugt eine neue AxBox. Diese kann entweder an ein aqua_gramm übergeben werden (Plot()), oder auf eine Reportseite ausgegeben werden (AxBoxOnPage()).

NewBigPage

Syntax: NewBigPage (Real breite, Real hoehe, Real spalten) : Page
breite: Breite der Seite in cm
hoehe: Höhe der Seite in cm
spalten: Anzahl der Spalten

Beispiel:
```
page := NewBigPage (240, 91.4, 800)
```

Beschreibung: Initialisiert eine Page, deren Ausdehnung durch *breite* und *hoehe* explizit vorgegeben wird. Die Anzahl der Spalten muss angegeben werden, damit die Größe der Fonts berechnet werden kann.

Siehe auch NewPage().

NewButton

Syntax: NewButton(String name, Real xpos, Real ypos, String label, String azurprg [, Real breite [, Real hoehe]])
name: Name des Buttons
xpos: Koordinaten in Pixeln, relativ zur linken
ypos: oberen Ecke des Windows
label: Aufschrift des Buttons
azurprg: Azurfunktion, mit der der Button verknüpft ist
breite: optional Breite des Buttons in Pixeln
hoehe: optional Höhe des Buttons in Pixeln

Beispiel:
```
NewButton( "RepBut", 20,140, "Reports", "Preports" )
```

Beschreibung: Ein Button mit der Beschriftung *label* wird an der Position $xpos, ypos$ (links oben des Buttons) erzeugt. Drückt der Benutzer diesen Button, dann wird *azurprg* ausgeführt. *name* wird als Parameter übergeben. In der Azurfunktion kann auf *name* mittels des String-Parameters Command zugegriffen werden.

Optional kann die Breite des Buttons und dessen Höhe angegeben werden. Die Voreinstellung dieser Werte richtet sich nach der Länge der Beschriftung und dem gewählten Font.

Siehe auch NewMenuButton() und NewTrigger().

NewCanvas

Syntax: NewCanvas(Real lox, Real loy, Real rux, Real ruy, Real xsize, Real ysize)
lox: X-Koordinate in Pixeln der linken oberen Ecke
loy: Y-Koordinate in Pixeln der linken oberen Ecke
rux: X-Koordinate in Pixeln der rechten unteren Ecke
ruy: Y-Koordinate in Pixeln der rechten unteren Ecke
xsize: Breite des Canvas in cm
ysize: Höhe des Canvas in cm

Beispiel: `NewCanvas(10, 200, 500, 450, 30.0, 20.0)`

Beschreibung: Der Canvas ist der Hintergrund für alle grafischen Ausgaben von Zeitreihen in Achsenkreuzen. lox, loy sind die Ecke links oben, rux, ruy entsprechend rechts unten (Ursprung ist links oben). $xsize$ und $ysize$ legen das virtuelle Koordinatensystem des Canvas in cm fest.

Alle Angaben relativ zum Canvas erfolgen in cm. Die Größe eines Plots liegt damit auch fest. Werden für rox oder roy negative Werte angegeben, dann wird der Canvas auf das ganze Fenster ausgedehnt, wobei entsprechend rox bzw. roy viele Pixel Abstand gelassen werden.

Jedes AGWindow kann höchstens einen Canvas enthalten. Ist schon einer vorhanden, dann wird dieser vorher gelöscht.

Unter dem Canvas werden Scroll- und Zoom-Buttons, ein Button zum Focus-Rücksetzen, sowie Felder zum Darstellen der x- und y-Koordinate des Mauszeigers angelegt. Diese Felder sind als AGElement unter den Namen @scrlleft, @scrlright, @zoomin, @zoomout, @popfocus, @xpos und @ypos zu erreichen. (Siehe AGSetElemPos(), DelAGElement() und SetHandle().

Zum Löschen eines Canvas' steht die Prozedur DelCanvas() zur Verfügung.

NewCheckBox

Syntax: NewCheckBox(String name, Real xpos, Real ypos, String label, Bool b, String prg)
name: Name der CheckBox
xpos: Koordinaten in Pixeln, relativ zur linken
ypos: oberen Ecke des Windows
label: Label der CheckBox
b: Startzustand, TRUE=An, FALSE=Aus
prg: Azurprogramm oder ""

Beispiel: `NewCheckBox( "MitSum", 20,165, "Mit Summen", FALSE, "")`

Beschreibung: Eine CheckBox mit der Beschriftung *label* wird an der Position $xpos, ypos$ (links oben) erzeugt. b ist die Vorbelegung. Der Benutzer kann die CheckBox zwischen ihren zwei Zuständen umschalten. Wird *prg* angegeben, so wird dieses aufgerufen, wenn die CheckBox ihren Zustand wechselt.

Der Zustand der Checkbox kann mit ExportVar() gesetzt werden. Der String, der damit übergeben wird, muss entweder TRUEöder FALSEßein.

Das Label kann mit `ExportVar(name, "@neues Label")` umgesetzt werden.

NewCombo

Syntax: NewCombo (String name, Real xpos, Real ypos, Real len, String starttext, String prg , String s1 ...)
name: Name des Combofeldes
xpos: Koordinaten in Pixeln, relativ zur linken
ypos: oberen Ecke des Windows
len: Länge des Combofeldes in Pixeln
starttext: Starttext des Combofeldes
prg: Azurprogramm oder ""
s1 ...: beliebig viele Listeneinträge

Beispiel: `NewCombo("C1", 20,240, 120, "", "TuWas", "Rot", "Blau", "Gelb")`

Beschreibung: Ein Combofeld wird angelegt. Dieses besteht aus einem Eingabefeld (siehe NewEingabe()) und einer Auswahlliste (siehe NewListe()). Die Paramter bis *prg* entsprechen daher denen von NewEingabe. Aus den darauf folgenden Parametern wird die Liste aufgebaut. Die Liste ist immer geschlossen.

len gibt die Gesamtbreite des Combofeldes an. Das Eingabefeld ist etwas schmaler, da die Liste Platz beansprucht.

NewDatenbank

Syntax: NewDatenbank() : Datenbank

Beispiel: `db := NewDatenbank()`

Beschreibung: Erzeugt eine neue Datenbank.

Siehe auch DBRead() und DBWrite().

NewDBGrid

Syntax: NewDBGrid(String name, Real xpos, Real ypos, Real breite, Real zeilen, Relation relation, String updhandle, String updfieldhandle, String selhandle, String selfieldhandle, Bool retflag, String dispfeld1, ... | Array A)
name: Name des Aquagramm-Elmentes
xpos, ypos: Position der linken oberen Ecke in Pixeln
breite: Breite in Pixeln
zeilen: Höhe in Textzeilen
relation: MemRelation
updhandle: Azurfunktion bei Update eines Tupels
updfieldhandle: Azurfunktion bei Update eines Feldes
selhandle: Azurfunktion bei Select eines Tupels
selfieldhandle: Azurfunktion bei Select eines Feldes
retflag: Welche Funktion soll der Return-Taste zugeordnet werden
dispfeld1 bis dispfeldn: darzustellende Felder
oder A: Array mit darzustellenden Feldern

Beispiel:
```
NewDBGrid("Auswahl", 20,270, 100, 10, wrel,"Aendern",
          "", "Sel", "", False, "Nr","Name")
```

Beschreibung: Erzeugt auf dem aktuellen AGWindow ein DBGrid mit Namen *name* an der Position x, y.

Ein DBGrid ermöglicht das Editieren einer Relation in Tabellenform. Zur Zeit werden nur MemRelationen (siehe dazu NewMemRelation() unterstützt.

Die Tupel werden zeilenweise dargestellt. Jede Zeile wird entsprechend der Tupelstruktur in Felder unterteilt. Dadurch ergibt sich eine Tabellenform. Der Benutzer kann die Felder direkt editieren.

Wenn mittels CreateIndex() eine Sortierreihenfolge auf der Relation festgelegt ist, werden die Tupel entsprechend sortiert angezeigt. Alle Tupel der Relation werden verarbeitet. Falls die Menge eingeschränkt werden soll, muss dies vorher geschehen (DBFilter().

Parameter

Mit $dispfelder$ kann eine Auswahl der darzustellenden Felder (also Spalten) ausgewählt werden. Die Felder können editierbar oder zum Editieren gesperrt sein. Das Format des $dispfelder$-Parameter ist unten beschrieben. Wird kein Feld angegeben, so werden alle Felder dargestellt.

Die Größe der Tabelle wird mit *breite* (Breite in Pixeln) und *zeilen* (Anzahl der gleichzeitig darzustellenden Tupel) festgelegt. Ist *breite* -1, dann wird das DBGrid automatisch an die Breite des AGWindows angepasst. Ein vertikaler und/oder horizontaler Scroll-Balken wird automatisch erzeugt, wenn nicht alle Felder bzw. Tupel in die so festgelegte Tabelle passen.

Der Parameter $retflag$ legt die Funktion der Return-Taste beim Beenden einer Eingabe fest. Ist $retflag$ TRUE, dann wird bei Beenden einer Eingabe mit Return der *updatehandle* aufgerufen und darauf der Cursor in die nächste Zeile gesetzt. Ist die aktuelle Zeile bereits die letzte, dann wird der Parameter Bool name_Ret auf TRUE gesetzt.

Ist $retflag$ FALSE, wird bei Beenden einer Eingabe mit Return der *updfieldhandle* aufgerufen und der Cursor nicht verändert. Der Parameter Bool name_Ret gibt in diesem Fall an, ob der Benutzer das Editieren des Feldes mit der Return-Taste abgeschlossen hat.

Die Parameter $dispfeld1$ bis $dispfeldn$ enthalten eine Liste von Display-Feldern, also Felder des Tupels, die im DBGrid angezeigt werden sollen. Alternativ kann diese Liste auch als Array übergeben werden.

Das Displayfeld ist der Name eines Feldes der Relation (z.B. `Ort`). In der betreffenden Spalte werden dann pro Zeile die Inhalte dieses Feldes dargestellt.

Codes

Manchmal sind die Inhalte eines Feldes Codes, die nicht direkt dargestellt werden, sondern mittels einer Code-Relation umgewandelt werden sollen. Dies hat den Vorteil, dass der Benutzer statt der internen Codes Klartext sieht. Er kann die Inhalte dann nicht direkt editieren, sondern wählt sie aus einer aufklappbaren Liste aus. Die Liste wird aus einer Code-Relation erzeugt. Das Displayfeld hat dann die Form:

`feldname|coderel|codefield+dispfeld1+...+dispfeldn`.

`feldname` ist der Name des Code-Felds (z.B. `gemeinde`), `codrel` ist eine dbf-Relation (z.B. `gemeinde`) oder eine MemRelation (z.B. `Str(memgemeinde)`), `codefield` ist der Name des Feldes in `coderel`, das den Code enthält (z.B. `gebcode`). Hinter dem `codefield` folgt ein `+`, darauf ein oder mehrere Feldnamen für die Klartexte der Codes (z.B. `LANGTEXT`).

Spaltenformatierung

Folgt dem Displayfeld direkt ein &-Zeichen, so wird der darauf folgende String als Spaltenüberschrift verwendet, ansonsten der Feldname. Die Breite einer Spalte ergibt sich aus dem Maximum der Feldlänge und der Länge der Überschrift. Durch Anhängen eines weiteren & kann die Breite in Pixeln jedoch explizit gesetzt werden.

Die Breite der Tabelle ergibt sich aus der Summe aller Spaltenbreiten. Ein vorangestelltes - vor den Displayfeldern bewirkt, dass diese nicht editiert werden können.

Bedienung

Durch Mausklick, Betätigen des Scroll-Balkens oder durch Cursortasten kann der Benutzer ein Tupel auswählen (Cursor), die entsprechende Zeile wird markiert. Dies ist eine **Selektion**. Die Azurfunktion *selhandle* wird aufgerufen (wenn gesetzt). Die Azurfunktion $selfieldhandle$ wird aufgerufen, wenn der Benutzer eine neue Zelle in der Tabelle selektiert hat.

Hat der Benutzer ein Feld eines Tupels geändert, so wird entweder der $updfieldhandle$ oder der *updatehandle* aufgerufen. Der *updatehandle* wird immer aufgerufen, wenn mit dem Beenden der Eingabe eine andere Zeile angewählt wurde oder, bei gesetztem $retflag$ dazu die Return-Taste betätigt wurde. Der $updfieldhandle$ wird aufgerufen, wenn bei Beenden der Eingabe nicht die Zeile verlassen wird.

Falls ein Feld zum Editieren gesperrt ist (also grün ist), wird der $updfieldhandle$ aufgerufen, wenn der Benutzer auf dem Feld einen Doppelklick ausgelöst hat.

Den Handle-Funktionen werden die Relation, das aktuelle Tupel und die aktuelle Spalte als Parameter übergeben. Wenn das DBGrid name heißt, dann hat die Relation den Namen name_Rel, das Tupel heißt name_Tup und die Spalte name_Feld. Diese Variablen sind keine Kopien, sondern verweisen direkt auf den entsprechenden Speicherbereich. name_Feld ist leer, wenn keine Spalte angewählt wurde. Der Parameter Bool name_Ret wird übergeben (s.o.).

Funktion von Bool-Feldern:

Ein Bool-Feld kann drei Zustände annehmen: Ja, Nein und Ungesetzt. Falls dem Bool-Feld nicht explizit ein Wert zugewiesen wurde, steht es auf Ungesetzt (leer). Durch Betätigen der Leertaste oder der Enter-Taste in einem Bool-Feld wird es umgeschaltet von Ungesetzt nach Ja nach Nein nach Ungesetzt.

Möchte man die Reihenfolge auf Ungesetzt , Nein, Ja, Ungesetzt ändern, so fügt man vor dem Displayfeld ein `!`. Fügt man ein `?` vor das Displayfeld, bedeutet das, dass der Zustand Ungesetzt entfällt, man also nur zwischen Ja und Nein wechseln kann.

Felder zum Editieren sperren:

Es können ganze Zeilen oder ganze Spalten zum Editieren gesperrt werden.

Spalten sperrt man, indem man bei der Angabe der Displayfelder dem Feldnamen des Feldes ein Minus voranstellt. Beispiel: `-Name`.

Zeilen, also Tupel, sperrt man, indem die Tupel um ein Bool-Feld `READONLY` erweitert werden. Ist der Inhalt dieses Feldes eines Tupels auf `True` gesetzt, dann wird es grün hinterlegt und ist zum Editieren gesperrt.

Felder ungewissen Inhalts:

Felder, die einen ungewissen Inhalt haben (siehe SetUntrusted()), werden in rot ausgegeben. Sie können editiert werden.

Tupel selektieren:

Wenn der Benutzer mit der rechten Maustaste in das Quadrat links neben ein Tupel klickt, wird das Tupel selektiert. Dies wird angezeigt, indem das Quadrat blau eingefärbt wird. Durch abermaliges Klicken wird das Tupel wieder deselektiert, das Feld wieder grau.

Die selektierten Tupel können z.B. mit der Funktion CollectAll() abgefragt werden.

Kommunikation mit azur :

Änderungen, die der Benutzer in dem DBGrid vornimmt, werden sofort in die Relation übernommen. Änderungen an der Relation, die von Azur aus erfolgen, müssen dem DBGrid jedoch mitgeteilt werden. Dazu wird die Funktion ExportVar() benutzt (das DBGrid heißt in den Beispielen immer `Auswahl`):

Wenn die azurseitige Änderung eines Tupels sich sofort auf der Oberfläche niederschlagen soll, muss ein `ExportVar("Auswahl", "@updatetupel "+Str(tup))` erfolgen.

Um neue Tupel im DBGrid aufzunehmen, gibt es drei Möglichkeiten:

a Dem DBGrid wird als Inhalt der Text `@insert` übergeben. Es erzeugt unterhalb des aktuellen Tupels (oder, wenn keine Zeile selektiert ist, oberhalb der ersten Zeile) eine leere Zeile und verknüpft diese mit einem leeren, neuen Tupel, dem *Inserttupel*. Dieses Tupel ist **nicht** in der Relation enthalten.

b Wird hinter `@insert` zusätzlich ein Tupel an ExportVar übergeben (also z.B. `ExportVar("Auswahl", "@insert "+Str(tup))`, wird dieses Tupel in die neue Zeile eingetragen. Dieses Tupel **darf nicht** in der Relation enthalten sein.

c Ein neues Tupel wird mit AppTupel() in die Relation aufgenommen. Danach wird mit `ExportVar("Auswahl", "@update")` das DBGrid an die Relation angeglichen, das neue Tupel also ins DBGrid übernommen.

Mittels [b] und [c] können Vorbelegungen realisiert werden.

Das neu eingefügte Tupel wird bei [a] und [b] nicht in die Relation aufgenommen. Es wird jedoch bei der Select- bzw. Update-Funktion als name_Tup übergeben und kann dann durch Azur in die Relation eingefügt werden (mit AppTupel()). Danach muss dem DBGrid ein `"@update"` geschickt werden. Dadurch erscheint das Tupel wie jedes andere Tupel der Relation im DBGrid. Ein Inserttupel gibt es nach einem `@update` nicht mehr. Um in der Select- bzw. Update-Funktion zu erkennen, ob es sich um das Inserttupel handelt, ist es (in Fall [b]) ratsam, sich das Tupel in einer statischen Variablen zu merken.

Wenn nach einem `@insert` das Tupel nicht in die Relation übernommen wird und ein weiteres `@insert` erfolgt, wird das erste Inserttupel verworfen.

Soll das aktuelle Tupel gelöscht werden, so geschieht dies in zwei Schritten: das Tupel wird durch Azur aus der Relation gelöscht (DelTupel()). Darauf wird dem DBGrid ein `@update` geschickt. Die Zeile ist nun gelöscht.

Mit `"@select "+Str(tup)` kann ein Tupel direkt selektiert werden. Mit `"@selectfield "`+feldname kann ein Feld des aktuell selektierten Tupels selektiert werden.

Mit `"@replace "+Str(rel)` kann eine neue Relation an das DBGrid übergeben werden.

Die Position des Cursors kann auch relativ verändert werden mit

`@left`	Bewegt den Cursor eine Zelle nach links
`@right`	Bewegt den Cursor eine Zelle nach rechts
`@up`	Bewegt den Cursor eine Zeile nach oben
`@down`	Bewegt den Cursor eine Zeile nach unten
`@relpos x y`	Bewegt den Cursor x Zellen nach rechts und y Zeilen nach unten (bei negativen Werten entsprechend in die andere Richtung)
`@home`	Bewegt den Cursor in die erste Zeile
`@end`	Bewegt den Cursor in die letzte Zeile

Mittels `"@startedit` bzw. `"@finishedit` kann das Editieren des Feldes, auf dem sich der Cursor befindet, explizit ein- oder ausgeschaltet werden.

Das DBGrid prüft selbstständig, ob Key-Felder ausgefüllt sind und der Key eines Tupels eindeutig ist. Ist dies nicht der Fall, wird das Beenden der Eingabe nicht erlaubt und der Benutzer über den Grund in der Statuszeile informiert.

NewDBListe

Syntax: NewDBListe(S name, Real xpos, Real ypos, Real breite, Real hoehe, Bool offen, S azurprg, S relation, S keyfeld, S dispfeld [, Bool ohneleerzeile])
name: Name der Liste
xpos: Koordinaten in Pixeln, relativ zur linken
ypos: oberen Ecke des Windows
breite: Breite der Liste in Pixeln
hoehe: Höhe der offenen Liste in Pixeln
offen: TRUE=die Liste ist geöffnet
azurprg: Azurprogramm oder ""
relation: Mem-Relation
keyfeld: Schlüsselfeld
dispfeld: darzustellende(s) Feld(er)
ohneleerzeile: optional: erzeuge keine Leerzeile am Ende der Liste

Beispiel:
```
NewDBListe("Farbe", 20,270, 100, 200, FALSE, "", Str(memrel), "Nummer", "
```

Beschreibung: Erzeugt auf dem aktuellen AGWindow eine neue DBListe.

Eine DBListe wird wie in NewListe() erzeugt. Die Einträge werden jedoch nicht direkt angegeben, sondern aus einer Relation ausgelesen. Dargestellt werden die Felder mit Namen *dispfeld*. *dispfeld* kann auch den Namen mehrerer Felder enthalten, die durch + getrennt sind. Angezeigt werden dann pro Zeile der Inhalt dieser Felder durch ein Leerzeichen getrennt.

Als Inhalt der Liste wird an ein Azurprogram der Inhalt des Feldes *keyfeld* des ausgewählten Tupels übergeben.

relation ist der Str() der Mem-Relation (siehe Beispiel).

```
NewDBListe("Farbe", 20,270, 100, 200, FALSE, "", \
Str(memrel), "Nummer", "Name")
```

eine Liste erzeugt.

Wenn das erste Zeichen von *dispfeld* ein @ ist, so wird der Rest des Strings als Name einer Azurfunktion betrachtet. Diese Azurfunktion muss als Parameter ein Tupel bekommen und als Ergebnis einen String liefern. Beim Aufbau der Liste wird dann für jedes Tupel diese Funktion aufgerufen und deren jeweiliger Ergebnisstring als Listeninhalt gesetzt.

Beispiel:

```
NewDBListe ("Farbe", 20,270, 100, 200, FALSE, "", \
            Str(memrel), "Nummer", "@MacheInhalt")

#-----------------------------------

MacheInhalt (Tupel tup) : String
   RETURN (GetText(tup,"Name"))
END
```

Sortierreihenfolge

Standardmäßig wird die Liste nach dem dargestellten Inhalt sortiert. Soll die Liste nicht sortiert werden, die Tupel also in der Reihenfolge durchlaufen werden, in der sie in die Mem-Relation gelangt sind, muss man den Str() der Mem-Relation in Kleinbuchstaben wandeln (mit `LowCase(Str(memrel))`).

Die Liste kann auch nach dem Schlüsselfeld sortiert dargestellt werden. Dies wird erreicht, indem man dem Schlüsselfeld ein `!` voranstellt (z.B. `"!Nummer"`).

Siehe auch NewListe().

NewEditfeld

Syntax: NewEditfeld (String name, Real xpos, Real ypos, Real breite, Real hoehe, String inhalt[, Real max])
name: Name des Buttons
xpos: Koordinaten in Pixeln, relativ zur linken
ypos: oberen Ecke des Windows
breite: Breite des Feldes
hoehe: Höhe des Feldes
inhalt: Vorbelegung des Feldes
max: optional maximale Anzahl von Zeichen inkl. ¡RETURN¿

Beispiel:
```
NewEditfeld( "Bems", 20,140, 200, 100, "" )
```

Beschreibung: Erzeugt ein Editierfeld mit beliebig vielen Zeilen. *breite* und *hoehe* geben die Größe des Feldes in Pixeln an. Enthält der Text mehr Zeilen als darstellbar sind, dann ermöglicht eine Scrollleiste das vertikale Scrollen. *text* ist optional und gibt die Vorbelegung des EditFeldes an.

Siehe auch NewEingabe().

Mittels ExportVar() kann man den Inhalt des Editfelds setzen. Beginnt der Text mit `@app` , so wird der Resttext an den bestehenden angehängt, `@push` bewirkt, dass er vorne eingefügt wird. Beispiel:

```
ExportVar ("Bems", "@app hinten dran")
```

```
ExportVar ("Bems", "@push vorne dran")
```

NewEingabe

Syntax: NewEingabe(String name, Real xpos, Real ypos, Real len, String starttext, String prg [, Bool geheim [, Real anz]])
name: Name der Eingabe
xpos: Koordinaten in Pixeln, relativ zur linken
ypos: oberen Ecke des Windows
len: Länge des Eingabefeldes in Pixeln
starttext: Starttext der Eingabe
prg: Azurprogramm oder ""
geheim: optional TRUE = die Eingabe soll verdeckt erfolgen
anz: optional maximale Anzahl Zeichen

Beispiel:
```
NewEingabe( "Komment", 20,240, 120, "", "TuWas")
```

Beschreibung: Ein Text-Eingabefeld wird an der Position *xpos*, *ypos* (links oben) mit der Länge *len* erzeugt. *starttext* ist die Vorbelegung dieses Feldes, welche leer sein kann (""). Als letzter Parameter kann eine Azurfunktion angegeben werden, die aufgerufen wird, wenn der Benutzer die Return-Taste innerhalb dieses Feldes betätigt (siehe NewButton()). Der optionale letzte Parameter legt fest, ob die Eingabe in das Feld verdeckt erfolgt oder offen. Die Vorbelegung dieses Parameters ist False, also offen.

Optional kann auch die maximale Anzahl Zeichen festgelegt werden, die der Benutzer im Eingabefeld eingeben kann.

Der Inhalt des Eingabefeldes kann als Parameter in einer Azurfunktion empfangen werden. Beispiel

```
TuWas (String Komment)
   print (Komment)
END
```

Siehe auch NewEditfeld().

NewGeoCanvas

Syntax: NewGeoCanvas(String name, Real x1,Real y1, Real x2,Real y2, String azrprg)
name: Name des GeoCanvas'
x1,y1: Pixelposition links oben
x2,y2: Pixelposition rechts unten
azrprg: Azur-Funktion bei mittlerer Maustaste

Beispiel:
```
NewGeoCanvas ("GeoC", 10, 100, 500, 500, "")
```

Beschreibung: Es wird ein Platz für eine Karte (ein Canvas) erzeugt. $x1, y1$ geben die linke obere Ecke des GeoCanvas auf dem Fenster in Pixeln an, $x2, y2$ entsprechend dessen rechte untere Ecke.

Die Azur-Funktion *azrprg* wird aufgerufen, wenn die mittlere Maustaste betätigt wird (siehe dazu FindPoly()). Hier kann auch, wie im Beispiel, ein Leerstring stehen, was diese Funktionalität abschaltet.

Unter dem GeoCanvas werden Felder zum Darstellen der x- und y-Koordinate des Mauszeigers angelegt. Diese Felder sind als AGElement unter den Namen @xpos und @ypos zu erreichen. (Siehe AGSetElemPos(), DelAGElement() und SetHandle().

Siehe auch PlotKarte().

NewLabel

Syntax: NewLabel(String name, Real x, Real y, String text, Real size, String farbe, Real winkel)
name: Name des Labels
x: Koordinaten in cm, relativ zur linken
y: unteren Ecke des Canvas
text: Text des Labels
size: Textgröße in cm
farbe: Farbe des Labels, siehe ZRInAxBox()
winkel: Textwinkel in Grad

Beispiel: `NewLabel( "lab1", 10.3, 20.5, "Karamax", 0.25, "Blau", 0)`

Beschreibung: Erzeugt auf dem Canvas des aqua_gramms ein neues Label. x und y geben die Position in cm an (relativ zum Canvas-Ursprung links unten). $size$ gibt die Größe des Textes in cm an. $farbe$ dessen Farbe (siehe ZRInAxBox()) und $winkel$ den Winkel in Grad (waagerecht 0, senkrecht 90).

NewListe

Syntax: NewListe(String name, Real xpos, Real ypos, Real breite, Real hoehe, Bool offen, String azurprg, String s1, ...)
name: Name der Liste
xpos: Koordinaten in Pixeln, relativ zur linken
ypos: oberen Ecke des Windows
breite: Breite der Liste in Pixeln
hoehe: Höhe der offenen Liste in Pixeln
offen: TRUE=die Liste ist geöffnet
azurprg: Azurprogramm oder ""
s1,...: Liste der einzelnen Auswahlwerte

Beispiel: `NewListe( "Farbe", 20,270, 100, 200, FALSE, "", "Schwarz", "Rot", "Blau")`

Beschreibung: Eine Liste wird an die Stelle $xpos, ypos$ platziert. Die Liste wird aus den Einträgen $s1$ bis sn zusammengesetzt. Die Liste ist entweder offen oder geschlossen. Im offenen Zustand ist sie *hoehe* Pixel tief, im geschlossenen wird nur eine Zeile dargestellt. Diese Zeile kann durch Mausdruck geöffnet werden, so dass mehrere Zeilen gleichzeitig dargestellt werden. Diese Zeile enthält immer den aktuell gewählten Eintrag.

Falls die Liste von Strings nur einen String enthält, wird dieser als Liste von Strings (siehe Token()) aufgefasst, die durch Leerzeichen getrennt sind.

azurprg wird aufgerufen, wenn ein Listeneintrag selektiert wird. Besteht *azurprg* aus zwei durch Komma getrennten Teilen (also z.B. `SelMe,KlickMe`, dann wird bei einem Doppelklick das zweite, hier also `KlickMe` aufgerufen.

Jedes Listenelement hat einen Inhalt, welcher angezeigt wird, und einen Key. Der Key wird automatisch erzeugt. Dem ersten Listenelement wird der Key 0 zugewiesen, den weiteren entsprechend die folgenden Zahlen. Wenn man eigene Keys verteilen will, muss man eine leere Liste anlegen und dann jedes Element mit ExportVar und `@App` (siehe unten) explizit anhängen.

Steuerung mit ExportVar (liste, s)

Setzen: s=key oder s=inhalt

Beispiel: `ExportVar("liste1", "1")`
oder: `ExportVar("liste1", "Ahausen")`

Beide Beispiele setzen die Selektion der Liste *liste*1 auf *Ahausen* (unter der Voraussetzung, dass *Ahausen* den Key 1 besitzt).

Ändern: s=@Set inhalt—key

Setzt das Listenelement mit dem Key *key* auf *inhalt*

Löschen: s=@Del key oder @Del inhalt

Löscht das Listenelement mit dem Key *key* oder dem Inhalt *inhalt*.

Alles löschen: s=@DelAll

Löscht alle Einträge in der Liste.

Erweitern: s=@App inhalt—key

Beispiel: `ExportVar("liste1", "@App Behausen|2")`

Hängt eine neues Listenelement an die Liste *liste*1 an. In der Liste erscheint *Behausen*. Dieses Element wird mit dem Key 2 angesprochen.

Falls kein Key zu einem Element bekannt ist, kann auch, wie bei @Del, der Inhalt angegeben werden. Die Eineindeutigkeit der Keys wird nicht vom System überwacht. Die Liste ist grundsätzlich nicht sortiert, weder nach Inhalt, noch nach Key. Die Reihenfolge richtet sich nach der Reihenfolge des Anfügens.

Abfragen der Liste

Eine Liste kann in der Parameterliste eines Handles als Array oder String empfangen werden. Im Falle eines Arrays wird die gesamte Liste in eine Array-Variable gewandelt. Das selektierte Element wird zum aktiven Arrayeintrag. Ist der Parameter hingegen ein String, enhält dieser den Key des selektierten Listenelements.

Wenn der Klick auf eine Liste einen Handle auslöst, dann kann zusätzlich durch den Parameter Label der String abgefragt werden, der in der Liste gerade selektiert ist.

ImportVar() ist eine weitere Möglichkeit, die Liste abzufragen. Der String, der von dieser Funktion zurückgeliefert wird, wird mittels StrToArray() in ein Array gewandelt. Dieses übernimmt wie oben das selektierte Listenelement als aktiven Arrayeintrag.

Auf die Übernahme einer Liste als Array sollte jedoch verzichtet werden, wenn die Liste viele Elemente enthält, da die Umwandlung in ein Array lange dauern kann.

Beispiel:

```
MeinHandle (Array liste1)
  aktueller_inhalt := liste1[CURRVAL]
  aktueller_key    := liste1[CURRKEY]
END
```

oder:

```
MeinHandle (String liste1)
  aktueller_key := liste1
END
```

oder:

```
MeinHandle ()
  arr := StrToArray(ImportVar("liste1"))
  aktueller_inhalt := arr[CURRVAL]
  aktueller_key    := arr[CURRKEY]
END
```

NewMemRelation

Syntax: NewMemRelation (String aufbau [, String name]) : Relation
aufbau: Struktur der Relation
name: optional Name der Relation

Beispiel:
```
R := NewMemRelation ("Name#10s,Absatz#7.2n")
```

Beschreibung: Erzeugt eine neue Relation mit der Struktur $aufbau$. Die Syntax des Strings $aufbau$ ist im Anhang wiedergegeben.

Die Relation wird nur im Hauptspeicher verwaltet, es wird keine Datei angelegt (siehe NewRelation()). Verliert die zurückgegebene Variable ihre Gültigkeit, wird die Relation wieder gelöscht.

Es besteht die Möglichkeit, eine ungültige Relation anzulegen. Dazu übergibt man einen Leerstring als $aufbau$.

Es besteht die Möglichkeit, den Namen der Relation zu setzen. Diesen übergibt man als zweiten Parameter. Ohne Angabe des Namens wird er zu Leerstring gesetzt.

Siehe auch DBFilter().

NewMenu

Syntax: NewMenu (String name, Real xpos, Real ypos, String label [,String supermenu])
name: Name des Menus
xpos: Koordinaten in Pixeln, relativ zur linken
ypos: oberen Ecke des Windows
label: Aufschrift
supermenu: optional Name des Menus, in dem dies enthalten sein soll

Beispiel:
```
NewMenu( "aktion", 100, 80, "Welche Aktion")
```

Beschreibung: Ein Menu besteht aus einen Button mit der Beschriftung *label* an der Position $xpos, ypos$ (links oben). Ein kleiner Pfeil nach unten macht ihn als Menu kenntlich. Drückt der Benutzer diesen Button, dann werden die diesem Menu zugeordneten MenuButtons dargestellt (siehe NewMenuButton()).

Falls *supermenu* angegeben ist, wird das Menu als Sub-Menu zu *supermenu* erzeugt. *xpos*, *ypos* sind in diesem Falle bedeutungslos. *supermenu* kann auch eine MenuBar (siehe NewMenuBar() sein.

NewMenuBar

Syntax: NewMenuBar (String name)
name: Name der MenuBar

Beispiel:
```
NewMenuBar ( "mainmenu")
```

Beschreibung: Eine Menubar ist eine waagerechte Leiste im obersten Teil des Windows, deren Position automatisch berechnet wird. Sie enthält Menus und Buttons.

Siehe auch NewButton() und NewMenu().

Achtung: MenuBars sind nur unter OpenLook implementiert. Auf anderen Systemen, wie z.B. MS-Windows, werden sie nicht unterstützt.

NewMenuButton

Syntax: NewMenuButton(String name, String label, String menu, String azurprg)
name: Name des MenuButtons
label: Aufschrift
menu: Name des zugehörigen Menus
azurprg: Azurfunktion (siehe NewButton())

Beispiel:
```
NewMenuButton( "simul", "Simulation", "aktion", "Psimul")
```

Beschreibung: Ein Button mit der Beschriftung *label* wird in das Menu *menu* angehängt. Drückt der Benutzer diesen Button, dann wird die Azurfunktion *azurprg* ausgeführt. *name* wird als Parameter übergeben, so kann ein Azurprogramm an mehreren Buttons hängen. *menu* kann auch eine MenuBar sein (siehe NewMenuBar().

NewMySQLRelation

Syntax: NewMySQLRelation(String name, String aufbau, String datenbank, String host, String uspw) : Relation
name: Name der Relation
aufbau: Struktur der Relation
datenbank: Name der Datenbank
host: Host, auf dem MySQL läuft
uspw: Username / Passwort

Beispiel:
```
R := NewMySQLRelation("umsatz", "Name#10s,Absatz#7.2n", "eglv", "birke",
```

Beschreibung: Erzeugt eine neue MySQL-Relation mit der Struktur *aufbau*. Die Syntax des Strings *aufbau* ist im Anhang wiedergegeben.

Die weiteren Parameter sind in MySQLRelation() erklärt.

NewOraRelation

Syntax: NewOraRelation(String name, String aufbau, String instanz , String userpasswd [, String tablespace]) : Relation
name: Name der Relation
aufbau: Struktur der Relation
instanz: siehe OraRelation()
userpasswd: Username / Passwort
tablespace: optional

Beispiel:
```
R := NewOraRelation("umsatz", "Name#10s,Absatz#7.2n", "AQUA", "basti/bast
```

Beschreibung: Erzeugt eine neue Oracle-Relation mit der Struktur *aufbau*. Die Syntax des Strings *aufbau* ist im Anhang wiedergegeben.

Die weiteren Parameter sind in OraRelation() erklärt.

NewPage

Syntax: NewPage (Real spalten, Real groesse, Real orientierung) : Page
spalten: Anzahl der Spalten
groesse: Eine der folgenden Konstanten : DINA0, DINA1, DINA2, DINA3, DINA4, DINA5
orientierung : PORTRAIT für Hochformat, LANDSCAPE für Querformat

Beispiel:
```
page := NewPage (80, DINA4, PORTRAIT)
```

Beschreibung: Initialisiert eine Page, wobei sich die Anzahl der Zeilen aus den Parametern ergibt. Die Page kann durch weitere Azur-Funktionen mit Linien und Texten gefüllt und dann in ein Plotfile geschrieben werden.

Siehe auch NewBigPage().

NewQuantenfolge

Syntax: NewQuantenfolge() : QuantList

Beispiel:
```
qf := NewQuantenfolge()
```

Beschreibung: Erzeugt eine neue, leere Quantenfolge (QuantList).

NewRahmen

Syntax: NewRahmen(String titel, Real xlo, Real ylo, Real xru, Real yru)
titel: Rahmen-Text
xlo: Koordinaten der linken
ylo: oberen Ecke in Pixeln, relativ zur linken oberen Ecke des Windows
xru: Koordinaten der rechten
yru: unteren Ecke

Beispiel:
```
NewRahmen("Eingabe", 10,200, 190, 400)
```

Beschreibung: Erzeugt einen Rahmen mit der linken oberen Ecke xlo, ylo und der rechten unteren Ecke xru, yru. Der Rahmen wird links oben mit dem Text *titel* versehen.

Das AG-Element erhält den Namen R_*titel*. Der Titel kann mit der Funktion ExportVar() gesetzt werden; der Name verändert sich dadurch nicht.

Wenn *titel* mit zwei Unterstrichen (__) beginnt, wird ein leeres Label erzeugt (der Rahmen wird also vollständig geschlossen). Dies dient der Unterscheidbarkeit von Rahmen-Elementen, wenn man sie später über ihren Namen (siehe oben) ansprechen will.

NewRelation

Syntax: NewRelation(String name, String aufbau [, Bool burst]) : Relation
name: Name der Relation
aufbau: Struktur der Relation
burst: optional Beschleunigung beim Schreiben

Beispiel:
```
R := NewRelation("logbuch", "Name#10s,Absatz#7.2n")
```

Beschreibung: Erzeugt eine neue Relation mit Namen *name* und der Struktur *aufbau*. Die Syntax des Strings *aufbau* ist im Anhang wiedergegeben.

Ist *burst* `TRUE`, dann werden spätere Schreibzugriffe beschleunigt. Dies geht jedoch auf Kosten der Lesezugriff-Geschwindigkeit. Dieser Parameter sollte daher nur benutzt werden, wenn in der Hauptsache Schreibzugriffe erfolgen.

Siehe auch Relation() und NewMemRelation().

NewReport

Syntax: NewReport () : Report

Beispiel:
```
rep := NewReport ()
```

Beschreibung: Legt eine Variable vom Typ Report an. Siehe auch NewPage().

NewSlider

Syntax: NewSlider(String name,Real x,Real y,Real len,Real von,Real bis,Bool vert,Real wert)
name: Name des Sliders
x: Koordinaten in Pixeln, relativ zur linken
y: oberen Ecke des Windows
len: Länge des Sliders in Pixeln
von: Minimalwert des Sliders
bis: Maximalwert des Sliders
vert: TRUE=Slider ist vertikal, FALSE=Slider ist horizontal
wert: Startwert des Sliders

Beispiel: `NewSlider( "TnWert", 20,200, 100, 5, 50, FALSE, 10 )`

Beschreibung: Ein Slider ist ein Regler, mit dem ein numerischer Wert eingestellt werden kann. Der Slider wird an Position x, y dargestellt und hat die Länge *len*. Ist *vert* TRUE, dann läuft der Slider von unten nach oben, sonst von links nach rechts. *von* und *bis* geben den Bereich an, den der Sliderwert annehmen kann. *wert* ist der Startwert des Sliders.

NewText

Syntax: NewText(String name, Real x, Real y, String text, Bool hidden [, String ausricht])
name: Name des Textes
x: Koordinaten in Pixeln, relativ zur linken
y: oberen Ecke des Windows
text: Inhalt des Textes
hidden: TRUE=Der Text soll verborgen sein
ausricht: optional: Ausrichtung des Textes

Beispiel: `NewText( "text1", 80,120, "Berechnung erfolgt.", FALSE)`

Beschreibung: Ein statischer Text mit der Beschriftung *text* wird an der Position *xpos*, *ypos* (links oben) erzeugt. Ist *hidden* TRUE, dann wird der Text nicht dargestellt, sondern dient lediglich als verborgene Variable.

ausricht ist ein optionaler Parameter. Mit ihm kann die Ausrichtung des Textes angegeben werden. Als Voreinstellung wird der Text so ausgegeben, dass er links bündig an die Position x, y anschließt. Wenn *ausricht* RECHTS ist, wird der Text so ausgegeben, dass sein Ende rechts bündig an x, y anstößt. Mit ZENTRIERT wird der Text um x, y zentriert.

Der Text kann mit ExportVar() geändert werden.

NewTrigger

Syntax: NewTrigger(String name,Real anz,Real x,Real y,String label,String azurprg [, String point [,Real breite [,Real hoehe]]])
name: Name des Triggers
anz: Anzahl der Klicks
xpos: Koordinaten in Pixeln, relativ zur linken
ypos: oberen Ecke des Windows
label: Aufschrift des Buttons
azurprg: Azurprogramm, mit dem der Triggerbutton verknüpft ist
point: optional Name eines Pointer-Typs
breite: optional Breite des Buttons in Pixeln
hoehe: optional Höhe des Buttons in Pixeln

Beispiel:
```
NewTrigger( "Schnitt", 4, 80,20, "Ausschneiden", "Pschnitt")
```

Beschreibung: Ein Trigger ist ein Button mit der Zusatzfunktion, dass *azurprg* erst aufgerufen wird, wenn der Benutzer *anz* mal mit der Maus in eine AxBox geklickt hat. Durch jeden Klick werden X- und Y-Wert eines Punktes definiert, die in den (automatisch angelegten) Variablen *name_x_*1, *name_y_*1 bis *name_x_anzahl*, *name_y_anzahl* abgelegt werden.

point bezeichnet optional den Namen eines Pointers, der statt des üblichen Pfeils während des Triggerns benutzt werden soll. Der Standardpointer wird durch Angabe eines Leerstrings angefordert.

Optional kann die Breite des Buttons und dessen Höhe angegeben werden. Die Voreinstellung dieser Werte richtet sich nach der Länge der Beschriftung und dem gewählten Font.

Siehe auch NewButton() und SetPointerType().

NewUVSRelation

Syntax: NewUVSRelation(String name, String aufbau, String dsn, String url) : Relation
name: Name der Relation
aufbau: Struktur der Relation
dsn: DSN-String der Datenbank
url: host:port

Beispiel:
```
R := NewUVSRelation("umsatz", "Name#10s,Absatz#7.2n", "suedwind test", "h
```

Beschreibung: Erzeugt eine neue Relation mit der Struktur *aufbau*. Die Syntax des Strings *aufbau* ist im Anhang wiedergegeben.

Die weiteren Parameter sind in UVSRelation() erklärt.

NextLuecke

Syntax: NextLuecke (ZR zr, XPunkt start) : Intervall
zr: Reihe
start: Punkt, ab dem eine Lücke gesucht werden soll

Beispiel:
```
naechste := NextLuecke (zr, Rechts(letzte)+~"1 Min")
```

Beschreibung: Sucht ab *start* die nächste Lücke in *zr*.

Ist keine weitere Lücke vorhanden, so wird ein ungültiges Intervall zurückgeliefert (siehe IsValid()).

Siehe FindeEreignis(), ExtrahiereEreignis() und SynthetisiereEreignis().

NextTupel

Syntax: NextTupel (Relation R, Tupel tup [, String idxfeld]) : Tupel
R: eine Mem-Relation oder eine UVS-Relation
tup: Tupel aus R
optional: idxfeld: maßgebliches Indexfeld

Beispiel:
```
t := NextTupel(Stamm, tup, "ORT")
```

Beschreibung: Liefert das Tupel aus der Relation R, das auf das Tupel *tup* folgt. Maßgeblich für die Sortierung idst $idxfeld$. Ist *tup* das letzte Tupel in R, dann wird ein ungültiges Tupel zurückgeliefert.

Ohne Angabe eines Indexfelds wird der Standardindex benutzt (siehe CreateIndex()) oder, wenn dieser nicht erstellt wurde, die unsortierte Reihenfolge.

tup muss ein Tupel aus R sein. Ist dies nicht der Fall, ist das Ergebnis undefiniert.

Siehe auch PrevTupel().

NJahrSerie

Syntax: NJahrSerie(ZR zr, Intervall bereich, Real M, Distanz mindau, Bool temp) : ZR
zr: Abfluss- oder Wasserstands-Zeitreihe
bereich: Auswertungszeitraum
M: langjähriges Mittel (Q oder W)
mindau: Mindestdauer eines Niedrigwasserbereiches
temp: Temporärflag

Beispiel:
```
zr2 := NJahrSerie(zr, bereich, 0.35, ~"7 Tage", FALSE)
```

Beschreibung: Erzeugt eine Momentanzeitreihe mit dem niedrigsten Niedrigwasser-Ereignis pro Jahr. Siehe hierzu NEreignisse().

Falls in einem Jahr kein Ereignis vorhanden ist, wird ein Wert in der Mitte des Jahres mit Luecke erzeugt. Dies sollte jedoch nur in extrem seltensten Fällen vorkommen.

Die Ergebniszeitreihe erbt alle Attribute der Ursprungszeitreihe, mit Ausnahme von Aussage, welche auf jSN gesetzt wird.

Siehe auch NPartSerie().

KSTest

Syntax: KSTest(ZR zre, ZR zrv, Real alpha) : Tupel
zre: beobachtete (empirische) Verteilung
zrv: theoretisch erwartete Verteilung
alpha: Signifikanzniveau (0,1)

Beispiel: `tup := KSTest(zrem, gumbelzr, 0.05)`

Beschreibung: Testet die Anpassung einer beobachteten an eine theoretisch erwartete Verteilung mittels des *Kolmogoroff-Smirnoff*-Tests. (Siehe [6]). Getestet wird, ob die theoretisch erwartete Verteilung als ungeeignet verworfen werden muss.

Das Ergebnis des Tests ist ein Tupel, das zwei Felder enthält: Das Boolfeld abgelehnt, das anzeigt, ob die Verteilung statistisch signifikant abgelehnt werden muss (`TRUE`), oder nicht abgelehnt werden kann (`FALSE`), und das Zahlfeld Dmax, das die maximale Abweichung enthält.

alpha (zwischen 0 und 1) legt das Niveau der Signifikanz des Tests und damit eine Grenze für Dmax fest. Je kleiner *alpha* ist, desto sicherer ist die Aussage des Tests.

Für ausreichend große Stichproben (mehr als 35 Werte), ist *alpha* frei wählbar (sinnvolle Werte liegen zwischen 0.001 und 0.2). Ist die Stichprobe kleiner, ist die zugrunde liegende Funktion zur Berechnung der Grenze für Dmax nicht anwendbar, die Werte werden dann einer Tabelle entnommen. *alpha* darf in diesem Fall nur die Werte 0.2, 0.1, 0.05, 0.02 und 0.01 annehmen. Für Tafelwerte und die Bestimmung von Dmax siehe [7].

Da der Ursprung der theoretisch erwarteten Verteilung *zrv* eine partielle Serie sein kann, ist die Größe der zugrunde liegenden Stichprobe nicht aus den Daten ermittelbar. Sie muss deshalb im Attribut Messgenau abgelegt sein. Die Anzahl der Jahre ergibt sich bei beiden Reihen aus den Attributen GueltVon und GueltBis.

Siehe auch Verteilung().

NPartSerie

Syntax: NPartSerie(ZR zr, Intervall bereich, Real M, Distanz mindau, Real fak, Bool temp) : ZR
zr: Abfluss- oder Wasserstands-Zeitreihe
bereich: Auswertungszeitraum
M: langjähriges Mittel (Q oder W)
mindau: Mindestdauer eines Niedrigwasserbereiches
fak: Es werden *jahre**fak Werte erzeugt
temp: Temporärflag

Beispiel:
```
zr2 := NPartSerie(zr, bereich, 0.35, ~"7 Tage", 1, FALSE)
```

Beschreibung: Erzeugt eine Momentanzeitreihe mit den $jahre * fak$ niedrigsten Niedrigwasser-Ereignissen in *bereich*. Siehe hierzu NEreignisse().

Falls in einem Jahr kein Ereignis vorhanden ist, wird ein Wert in der Mitte des Jahres mit Luecke erzeugt. Dies sollte jedoch nur in extrem seltensten Fällen vorkommen.

Die Ergebniszeitreihe erbt alle Attribute der Ursprungszeitreihe, mit Ausnahme von Aussage, welche auf pSN gesetzt wird.

Siehe auch NJahrSerie().

NWGrenze

Syntax: NWGrenze (ZR z) : Real
z:

Beispiel:
```
grenze := NWGrenze( z )
```

Beschreibung: Liefert das Attribut Nachweisgrenze (NWGrenze) der Zeitreihe.

OkBox

Syntax: OkBox(String frage) : Bool
frage: beliebiger Text

Beispiel: `IF (OkBox("Wirklich entfernen?"))`

Beschreibung: Erzeugt ein neues Fenster, welches den Text *frage* und die Buttons OK und Abbruch enthält. Das AGWindow, aus dem diese Ok-Box gestartet wurde, ist solange inaktiv, bis der Benutzer einen der beiden Button gedrückt hat. Der Programmablauf wird solange angehalten.

Der Rückgabewert ist TRUE, wenn der Benutzer den OK-Button gedrückt hat, sonst FALSE.

Siehe auch SelectBox() und InputBox().

OnOff

Syntax: OnOff(String s) : Bool
s : Ein String

Beispiel: `IF (OnOff(wort))`

Beschreibung: Testet, ob *s* einen Wert enthält, der *wahr* oder *falsch* ausdrückt. Wahr sind An, Ja, On, Yes, 1, True, T und Oui. Alle anderen Wörter gelten als *falsch*. Groß-Klein-Schreibung wird nicht beachtet.

OpenRel

Syntax: OpenRel(String name) : Relation
name: Name der DBF-Relation

Beispiel: `R := OpenRel ("betreiber")`

Beschreibung: Öffnet eine DBF-Relation und liefert davon eine Kopie als Memory-Relation zurück. Änderungen an Tupeln werden nicht automatisch in die DBF-Relation übernommen. Dies kann jedoch mit WriteDBF() erreicht werden.

In Programmen, die mit ADBInit() arbeiten, sollte man stattdessen die Funktion ADBOpenRel() benutzen.

OpenZR

Syntax: OpenZR (Tupel *rt*, Bool anlegen) : ZR
rt: ein ReiheTupel
anlegen: optional Zeitreihe anlegen? (Vorbelegung: TRUE)

Beispiel:
```
neu := OpenZR(t)
```

Beschreibung: Öffnet eine bestehende Zeitreihe oder legt eine neue mit den Attributen in *rt* an. Mit dem optionalen Parameter *anlegen* kann man das Anlegen von neuen Zeitreihen unterdrücken. Wenn *anlegen* `FALSE` ist und die Zeitreihe nicht existiert, wird eine ungültige Zeitreihe zurückgegeben (siehe IsValid()).

Siehe auch GetZR() und ReiheTupel().

OpenZRId

Syntax: OpenZRId (String id) : ZR
t: eine die Reihe beschreibende Id (Dateiname)

Beispiel:
```
neu := OpenZR (dieid)
```

Beschreibung: Öffnet eine bestehende Zeitreihe mit der Id *id*. Ist keine Zeitreihe mit passender Id vorhanden, dann wird eine ungültige Zeitreihe (siehe IsValid()) zurückgeliefert.

Die Id einer Zeitreihe entspricht (im Kontext eines Datenpools) dem Dateinamen der Datei, die die Zeitreihendaten enthält. Sie kann über das Attribut WerteDatei (siehe Attribute()) ermittelt werden.

Siehe auch GetZR() und OpenZR().

OraRelation

Syntax: OraRelation (String name, String instanz [, String userpasswd]) : Relation
name: Name der Relation
instanz: auf dem Client etablierte Instanz zum Zugriff auf den Server
userpasswd: optional User/Passwort, Voreinstellung: `"scott/tiger"`

Beispiel:
```
R := OraRelation ("stammdat", "AQUA")
```

Beschreibung: Öffnet eine Oracle-Relation.

Instanz ist der Name einer Client-Instanz, über die der Zugriff auf einen Oracle-Server realisiert ist. Diese Instanz muss mit dem Programm Net8 von Oracle etabliert worden sein. *userpasswd* enthält den User und dessen Passwort mit / getrennt.

Wenn *name* ein Leerstring ist, dann wird eine ungültige Relation erzeugt.

Siehe auch NewOraRelation().

OraSQLDirekt

Syntax: OraSQLDirekt(String instanz , String userpasswd , String command) : Relation
instanz: siehe OraRelation()
userpasswd: Username / Passwort
command: SQL-Befehl

Beispiel:
```
OraSQLDirekt("AQUA", "basti/basti", sql_befehl)
```

Beschreibung: Führt einen SQL-Befehl direkt aus. Der SQL-Server muss Oracle sein.

Ort

Syntax: Ort (ZR z) : String
z:

Beispiel:
```
wo := Ort (z)
```

Beschreibung: Liefert das Attribut *Ort* der Zeitreihe.

OrtQuery

Syntax: OrtQuery (String s) : ZRList
s:

Beispiel: `liste := OrtQuery ("29004005")`

Beschreibung: Liefert alle Zeitreihen, die den Ort s haben. Siehe auch Query() und ReiheTupel().

Owunda

Syntax: Owunda(ZR serie, Intervall bereich, Bool temp) : ZR
serie: partielle oder jaehrliche Serie
bereich: Auswertungszeitraum
temp : Temporärflag

Beispiel: `vp := Owunda (pserie, dekade, true)`

Beschreibung: Berechnet die Parameter u und w der Verteilungsfunktionen aller Dauerstufen aus *serie*. Das Vorgehen ist in Verteilungsparameter() beschrieben. Owunda unterscheidet sich in der Anpassung von u und w an einen Funktionsverlauf. Statt die Dauerstufen in drei Bereiche einzuteilen und in diesen einen linearen Funktionsverlauf zu bestimmen (siehe [3]), wird hier das in OWUNDA [10] beschriebene Verfahren angewendet.

Ziel dieses Verfahrens ist es, aus einer Menge von Funktionen, diejenige auszuwählen, die die geringste Abweichung zwischen Originalwerten und angepassten Werten aufweist.

Folgende Funktionen werden untersucht:

a) $y = a + bx$

b) $y = ae^{bx}$

c) $y = a + b \ln x$

d) $y = ax^b$

e) $y = a + bx + cx^2$

f) $y = a + bx + cx^2 + dx^3$

g) $y = a + bx + cx^2 + dx^3 + ex^4$

h) $y = a + bx + cx^2 + dx^3 + ex^4 + fx^5$

i) $y = \frac{x}{b+ax}$

j) $y = axe^{bx}$

k) $y = axe^{bx^2}$

Dazu wird in zwei Stufen vorgegangen.

1. Stufe: Es wird die Funktion gesucht, mit der u am besten angepasst werden kann. Für alle Funktionen werden die Formen $u = f(D)$, $u = f(\ln D)$ und $u = e^{f(\ln D)}$ betrachtet. Eine Funktion wird verworfen, wenn sie nicht über die Dauerstufen streng monton wächst. Die so ermittelte Funktion geht nun in die zweite Stufe ein.

2. Stufe: Es wird die Funktion gesucht, mit der w am besten angepasst werden kann. Auch hier werden jeweils die drei Formen betrachtet. Eine Funktion wird verworfen, wenn $h_D = u + w \ln T$ über D nicht monton wächst (untersucht wird bei T=100).

Die Ermittlung der Koeffizienten der jeweiligen Funktion a) bis d) und i) bis k) erfolgt nach dem *Prinzip der kleinsten Quadrate*. Die übrigen Funktionen e) bis h) sind Polynome, deren Parameter nach der *Tschebyscheff'schen Methode* approximiert werden.

Zur weiteren Verarbeitung dient die Funktion Regenhoehenlinie.

PackHeader

Syntax: PackHeader (String datei) : Tupel
datei: Packdatei

Beispiel: `headertup := PackHeader("pack.zpa")`

Beschreibung: Liefert den Header einer Packdatei. Das Tupel hat den Aufbau Reihenart#1S,von#14S,bis#14s,anzahl#7N,Kommentar#200s. Diese Information kann dem Benutzer als grobe Übersicht präsentiert werden. Der Kommentar sollte vom Ersteller möglichst sinnvoll und sprechend ausgefüllt werden.

Siehe auch PackRead(), PackWrite() und PackInfo().

PackInfo

Syntax: PackInfo (String datei) : Relation
datei: Packdatei

Beispiel: `rel := PackInfo("pack.zpa")`

Beschreibung: Liefert die Header aller Packs in *datei* als MemRelation. Zu jedem Pack gibt es ein Tupel, das den gesamten Attributesatz der Zeitreihe und zusätzlich die Felder Nummer#7N, AnzQuals#2N, Laenge#8N, Select#B und OrtSynchro#B enthält. Diese Tupel können als Parameter der Funktion OpenZR() verwendet werden, um die entsprechende (lokale) Zeitreihe zu öffnen. Das Feld Select wird für alle Tupel auf TRUE, OrtSynchro auf FALSE gesetzt. Diese beiden Felder werden für PackRead() benötigt.

Das Ergebnis kann als Parameter für PackRead() benutzt werden. Wenn alle Packs einer Packdatei ohne vorherige Auswahl importiert werden sollen, geschieht das mittels:

```
PackRead("pack.zpa", PackInfo("pack.zpa"), TRUE, "pack.log")
```

Siehe auch PackRead(), PackWrite() und PackHeader().

PackRead

Syntax: PackRead (String datei, Relation rel, Bool overwrite, Bool merge, Bool aneu, String logdatei, Bool mitstatus)
datei: zu importierende Datei
rel: Information, welche Packs importiert werden sollen
overwrite: steuert, ob vorhandene Daten überschrieben werden
merge: steuert, ob Werte mit vorhandenen zusammengemischt werden
aneu: steuert, welche Interpretationsattribute sich durchsetzen
logdatei: Name der Log-Datei
mitstatus: sollen Ablaufmeldungen ausgegeben werden?

Beispiel: `PackRead("pack.zpa", rel, True, False, True, "pack.log", False)`

Beschreibung: Liest die Packs aus einer Packdatei und schreibt sie in die entsprechenden Zeitreihen. Es werden dabei nur die Packs bearbeitet, deren Tupel in *rel* ein gesetztes Feld Select besitzen (siehe PackInfo()). Das Feld Nummer eines Tupels bestimmt, zu welchem Pack es gehört. Packs, zu denen kein Tupel gehört, werden nicht importiert. Der Benutzer kann also vorher wählen, welche Zeitreihen importiert werden sollen.

Die Attribute der importierten Zeitreihen richten sich nach dem jeweiligen Tupel in *rel* und **nicht** nach den in *datei* gespeicherten Attributen. Es ist also möglich, die Attribute (z.B. den Ort()) beim Import zu beeinflussen, indem man sie zwischen PackInfo() und PackRead umsetzt.

Das Feld OrtSynchro der Tupel in *rel* gilt nur für Reihen-Reihen, also Reihen, die als Y-Werte andere Reihen referenzieren. Ist OrtSynchro False (Voreinstellung nach PackInfo()), so richtet sich der Ort der referenzierten Reihe nach dem Pack in *datei*. Ist OrtSynchro True, dann erben die referenzierten Reihen stattdessen den Ort von der Reihen-Reihe.

Enthält die Packdatei mehrere Tausend Packs, ist es nicht sinnvoll, den Benutzer eine explizite Auswahl treffen zu lassen. Der Parameter *overwrite* steuert dann lediglich, ob bereits vorhandene Daten überschrieben werden sollen, oder nicht.

Der Parameter *merge* legt fest, ob die Werte in die vorhandenen hineingemischt werden, oder diese ersetzen. Das Standardverhalten von Zeitreihen ist immer, die vorhandenen zu ersetzen. Hineinmischen ist sinnvoll, wenn die Werte als einzelne Messwerte betrachtet werden, von denen keiner verloren gehen soll. Dies ist nur selten der Fall.

Mit *aneu* legt man fest, ob die Interpretationsattribute der Zeitreihe im Pack die der vorhandenen Zeitreihe ersetzen sollen (TRUE) oder nicht (FALSE).

In jedem Fall wird eine Logdatei geschrieben, die pro Zeile Auskunft darüber gibt, ob ein Pack importiert wurde und ob Daten überschrieben wurden. Die Logdatei wird nach jedem Schreiben einer Zeile geflusht, sodass auch nach einem eventuellen Absturz die Information verfügbar ist, was importiert wurde.

Ist *mitstatus* True, so wird für jede eingelesene Zeitreihe eine Meldung in der Statuszeile (oder auf der Konsole) ausgegeben.

Zeitreihen, die lokal nicht vorhanden sind, werden lokal neu angelegt. Dies wird in der log-Datei vermerkt.

Siehe auch PackWrite(), PackHeader() und PackInfo().

PackWrite

Syntax: PackWrite (ZRList zrl, String datei, String kommentar, String mitstatus)
zrl: Liste der zu exportierenden Zeitreihen
datei: Ausgabedatei
kommentar: Kommentar des Pack-Headers
mitstatus: sollen Ablaufmeldungen ausgegeben werden?

Beispiel:
```
PackWrite(zrl, "pack.zpa", "Austauschdatei, i.A. Stempelmann", False)
```

Beschreibung: Erstellt eine Packdatei. *zrl* enthält die Liste der Zeitreihen, die in die Packdatei geschrieben werden, sowie den Fokus, über den exportiert werden soll. *datei* ist frei wählbar. Ist diese Datei schon vorhanden, wird sie überschrieben. Der *kommentar* kann bis zu 200 Zeichen enthalten, alle weiteren Zeichen werden abgeschnitten.

Sind in *zrl* sehr viele Zeitreihen enthalten, kann die Ausführung **sehr** lange dauern. Es werden daher Statusmeldungen auf der Statuszeile ausgegeben (falls *mitstatus* True ist).

Es werden immer **alle Qualitäten** exportiert. Möchte man einzelne Qualitäten exportieren, ist das AQZ-Ascii-Format zu empfehlen (siehe Export()).

Siehe auch PackRead(), PackHeader() und PackInfo().

PageMaxPos

Syntax: PageMaxPos (Page seite) : GeoPoint
seite: eine Reportseite

Beispiel:
```
maxp := PageMaxPos (seite)
```

Beschreibung: Liefert die Koordinaten der rechten, oberen Ecke der Page *seite* in cm. Siehe auch PagePos().

PageOnReport

Syntax: PageOnReport (Report rep, Page page)
rep:
page: eine Reportseite

Beispiel:
```
PageOnReport (rep, page)
```

Beschreibung: Fügt die Seite *page* dem Report *rep* zu. Die Seiten werden in der Reihenfolge ausgegeben, in der sie hinzugefügt wurden.

Siehe PrintReport() und NewPage().

PagePos

Syntax: PagePos (Page seite, GeoPoint p) : GeoPoint
seite: eine Reportseite
p: Punkt auf der Seite in Spalten und Zeilen

Beispiel:
```
pp := PagePos (seite, {0,78})
```

Beschreibung: Wandelt Koordinatenangaben in Zeilen und Spalten in cm-Werte um. Das Ergebnis kann z.B. dazu benutzt werden, um AxBoxen in den laufenden Text einfließend auf einer Page auszugeben. Siehe auch PageMaxPos(), AxBoxOnPage() und SetAxLage().

Parameter

Syntax: Parameter (ZR z) : String
z:

Beispiel: `par := Parameter (z)`

Beschreibung: Liefert das Attribut *Parameter* der Zeitreihe.

ParamQuery

Syntax: ParamQuery (String s) : ZRList
s:

Beispiel: `liste := ParamQuery ("Temperatur")`

Beschreibung: Liefert alle Zeitreihen, die den Parameter *s* haben. Siehe auch Query() und ReiheTupel().

PartielleSerie

Syntax: PartielleSerie(ZR zr, ZR tagesw, Intervall bereich, Bool temp, [Real anz]) : ZR
zr: Zeitreihe (kontinuierlich oder 5-Minuten-Summen)
tagesw: Zeitreihe mit Tageswerten
bereich: Auswertungszeitraum
temp: Temporärflag
anz: optional: Voreinstellung 2.5

Beispiel: `pserie := PartielleSerie(zr, zrt, dekade, false)`

Beschreibung: Erzeugt die partielle Serie der jeweiligen Niederschlags-Zeitreihe auf dem Auswertungszeitraum *bereich*. Alle bestehenden Werte in der Serie werden gelöscht. Ist die partielle Serie nicht vorhanden, dann wird sie angelegt. Das Attribut Ort ergibt sich aus dem Ort der Zeitreihe *zr*. Das Attribut Aussage wird auf pSe gesetzt.

Die Partielle Serie ist eine Momentan-Zeitreihe mit 21 Qualitätsschichten. Jede Schicht enthält die Menge der Extremwerte mit Zeitbezug zu einer Dauerstufe. Die Dauerstufe ist als Klartext zum ersten Zeitpunkt abgelegt. Dauerstufen sind fest: 5, 10, 15, 20, 30, 45, 60 und 90 Minuten, 2, 3, 4, 6, 9, 12 und 18 Stunden und 1, 2, 3, 4, 5 und 6 Tage.

Ist *zr* eine kontinuierliche Zeitreihe, so werden die benötigten 5-Minuten-Summen stückweise berechnet. Es ist sinnvoll, ausschließlich mit kontinuierlichen Zeitreihen zu arbeiten, da das Vorhalten von 5-Minuten-Summen über lange Zeiträume unverhältnismäßig viel Speicherplatz beansprucht.

Der optionale Parameter *anz* legt fest, wieviele Werte die partielle Serie enthalten soll. Die Anzahl Werte ist *anz*· Anzahl Jahre, die nicht vollständig Lücke sind, in *bereich*. Die Anzahl der Jahre, die nicht vollständig Lücke sind, wird im Attribut NWGrenze() abgelegt.

Nach [3] wird, um die statistische Unabhängigkeit zu gewährleisten, pro *Tag* nur ein Eintrag in die partielle Serie zugelassen. Da die Dauerstufen größer 1 Tag aus den Tagessummen des Messers berechnet werden, werden die Grenzen eines *Tages* entsprechend der Blockung der Tageswert-Zeitreihe gelegt (typischerweise 7:30).

Obwohl es nach obiger DVWK-Richtlinie nicht zwingend vorgeschrieben ist, wird folgende (hydrologisch sinnvolle) Randbedingung berücksichtigt: *Jede Tagessumme geht höchstens einmal in die partielle Serie ein, auch wenn sie durch die überlappende Berechnung mehrfach zum Zuge kommen würde.*

Zur Berechnung siehe [3]. Für die weitere Bearbeitung siehe Verteilungsparameter.

Plot

Syntax: Plot(AxBox ax)
ax:

Beispiel:
```
Plot ( axmitte)
```

Beschreibung: Zeichnet *ax* in das aqua_gramm. Die Ausdehnung der X-Achse muss vorher mit SetAxXBereich() gesetzt werden.

Falls die AxBox sich schon auf dem Canvas befindet, dann wird sie lediglich angepasst und, wenn möglich, nicht neu gezeichnet.

Siehe auch ClearCanvas().

PlotKarte

Syntax: PlotKarte(Karte K)
K: eine Karte

Beispiel: `PlotKarte (Map)`

Beschreibung: Zeichnet die Karte auf den GeoCanvas des aktuellen AGWindows. Ist kein GeoCanvas definiert, wird eine Fehlermeldung erzeugt. Die Karte wird auf die volle Größe des GeoCanvas' ausgedehnt. Passt das Seitenverhältnis des GeoCanvas' nicht zum Seitenverhältnis der Karte (siehe KarteSetBereich()), wird ein entsprechend kleinerer Ausschnitt der Karte dargestellt.

Wenn die Karte nicht aus einer Szenerie-Datei geladen wurde (siehe ReadKarte()), die den darzustellenden Ausschnitt vorgibt, so muss der Ausschnitt vorher mittels KarteSetBereich() oder KarteSetVoll() gesetzt werden.

Siehe auch Karte() und KarteOnPage().

PlotTextOnPage

Syntax: PlotTextOnPage(Page page, GeoPoint wo, String text, Real size, [Real style, [Real winkel | String ausricht]])
page:
wo: Koordinate in cm
text: beliebiger Text
size: Texthöhe in cm
style: NORMAL, BOLD oder ITALIC
winkel: optional. In Grad, 0 bedeutet waagerecht. oder:
ausricht: siehe dazu TextOnPage()

Beispiel:
```
PlotTextOnPage (page, {3.5,7.89}, "Haupttabelle", 0.2, NORMAL)
```

Beschreibung: Zeichnet einen Text an der angegebenen Position auf die Seite. Die linke untere Ecke hat die Koordinate (0,0). Mit *size* wird explizit die Fontgröße gesetzt.

Siehe auch TextOnPage(), DrawLineOnPage() und PlotTextOnPage()

PolyAdjust

Syntax: PolyAdjust(Polygon p1, Polygon p2, Real ftol)
p1: ein Polygon
p2: ein benachbartes Polygon
ftol: Fehlertoleranz in m

Beispiel:
```
PolyAdjust (poly1, poly2, 10)
```

Beschreibung: Falls *poly*1 und *poly*2 Segmente besitzen, deren Start- und Endpunkt nicht weiter als $ftol$ m entfernt liegen, dann werden die Polygone angepasst. Die beiden Segmente werden zu einem verschmolzen. Die ursprünglichen Segmente werden dann durch dieses neue Segment ersetzt, welches sich die beiden Poylgone teilen.

Siehe auch PolyGlue().

PolyArea

Syntax: PolyArea (Polygon p) : Real
p: ein Polygon

Beispiel: `flaeche := PolyArea (poly)`

Beschreibung: Berechnet die Fläche des Polygons p in Quadratmetern. Ist das Polygon nicht geschlossen oder degeneriert (weniger als 4 Punkte oder alle Punkte kolinear), dann wird 0 zurückgeliefert.

Besitzt das Polygon Inseln, so wird deren Fläche von der Gesamtfläche abgezogen.

Siehe auch PolyLen() und PolyDist().

PolyAttr

Syntax: PolyAttr(Polygon P, String attr) : String
P: Ein Polygon
attr: Name des Attributs

Beispiel: `farbe := PolyAttr (poly, "Farbe")`

Beschreibung: Liefert das Attribut *attr* des Polygons P. Ist *attr* kein Attribut eines Polygons, dann wird ein Leerstring zurückgeliefert.

Ist das Attribut eine Zahl, kann man diese mittels StrToReal() aus dem Rückgabestring gewinnen.

Ist das Attribut ein Bool, dann werden die Strings `"TRUE"` bzw. `"FALSE"` geliefert.

Attribute sind:

- Name (siehe auch Name() und SetName())
- Attribut2
- NumAttribut ein Zahlwert
- Label das Label beim Zeichnen (normalerweise Name)
- Filled `"TRUE"` = wenn das Polygon gefüllt ist
- SymbolTyp Symbolnummer für unselektierte Einpunktpolygone
- SelSymbolTyp Symbolnummer für selektierte Einpunktpolygone
- SymbolWinkel Winkel des Symbols in Grad
- LabelSize Größe des Labels in cm
- LabelUnter Label unter das Polygon zeichnen (bei Einpunkt-Polygonen)
- Farbe
- FillFarbe (normalerweise Farbe)
- PunkteAn `"TRUE"` = es werden die Stützpunkte mitgezeichnet
- ZeichneLabel `"FALSE"` = die Label werden nicht mitgezeichnet
- LineStyle `Solid, Dashed, Dotted, LongDashed, DashDotted, DashDotDotted`

Siehe auch PolySetAttr() und LayerAttr().

PolyCenter

Syntax: PolyCenter(Polygon p) :GeoPoint
p: ein Polygon

Beispiel:
```
mitte := PolyCenter (poly)
```

Beschreibung: Berechnet den Mittelpunkt von p unter der Annahme, p sei eine Fläche. Der Mittelpunkt ist der Flächenschwerpunkt der Fläche. Inseln werden bei der Berechnung berücksichtigt.

Siehe auch PolyLen() und PolyArea().

PolyDist

Syntax: PolyDist(Polygon p, GeoPoint gp) : Real
p: ein Polygon
gp: ein Punkt

Beispiel: `abstand := PolyDist (poly, einzelpunkt)`

Beschreibung: Berechnet den Abstand des Punktes *gp* vom Polygon *p*. Dieser ergibt sich aus dem Minimum aller Abstände von *gp* zu allen Stützpunken von *p*.

Siehe auch PolyArea(), PolyLen() und PolyInside().

PolyEntkolin

Syntax: PolyEntkolin(Polygon p, Real ftoleranz)
p: ein Polygon
ftoleranz: Fehlertoleranz in m

Beispiel: `PolyEntkolin (poly, 10)`

Beschreibung: Entkollinearisiert *p*. Entkollinearisieren bedeutet, dass alle Punkte, die kollinear (oder annähernd kollinear) sind, entfernt werden.

Dazu werden über alle Punkte jeweils drei aufeinanderfolgende betrachtet. Ist die Entfernung des mittleren zu der Verbindung der beiden äußeren kleiner als $ftoleranz$, dann wird dieser entfernt.

Siehe auch PolyLen() und PolyArea().

PolyGlue

Syntax: PolyGlue(Polygon p1, Polygon p2, Real ftol) : Polygon
p1: ein Polygon
p2: ein benachbartes Polygon
ftol: Fehlertoleranz in m

Beispiel: `gluedpoly := PolyGlue (poly1, poly2, 10)`

Beschreibung: Verschmilzt die Polygone *p*1 und *p*2 zu einem neuen Polygon und liefert dieses zurück. Ist ein Verschmelzen nicht möglich, so wird ein ungültiges Polygon zurückgeliefert.

Verschmolzen werden können zum einen Polygone, die sich Segmente teilen (dasselbe Segment referenzieren). Falls die Polygone sich mehr als ein Segment teilen, können Inseln entstehen. Diese Inseln sind reine Ausschlussflächen, die zwar Inseln des Polygons, jedoch keine selbständigen Polygone sind.

Zum anderen können Polygone verschmolzen werden, die an gegenüberliegenden Seiten eines, ungewünschten Randes liegen. Der Rand durchschneidet gleichsam ein Polygon in zwei Hälften, PolyGlue macht dies wieder rückgängig. Die Randstücke müssen nicht identisch sein. Als Verschmelzungspunkte werden die Mittel der jeweils passenden Endpunkte benutzt. Verschmolzen werden können Punkte, die nicht mehr als $ftol$ m entfernt liegen.

Siehe auch PolyAdjust()

Polygon

Syntax: Polygon(String name, [GeoPoint p1, ...]) : Polygon
name: Name des Polygons

Beispiel:
```
poly := Polygon ("Strasse", {10,10}, {20,23}, {50,34})
```

Beschreibung: Erzeugt ein neues Polygon und gibt dieses zurück. Es kann eine beliebige (auch leere) Folge von Punkten angegeben werden, aus denen das Polygon zusammengesetzt ist.

Punkte können später auch mit dem +-Operator angehängt werden, z.B.:

```
poly := poly + {75,67}
```

Siehe auch Layer().

PolyInside

Syntax: PolyInside (Polygon poly, GeoPoint gp) : Bool
poly: ein Polygon
gp: ein Punkt

Beispiel:
```
innen := PolyInside (poly, einzelpunkt)
```

Beschreibung: Gibt zurück, ob der Punkt *gp* im Polygon *poly* enthalten ist.

Siehe auch PolyArea() und PolyLen().

PolyLen

Syntax: PolyLen(Polygon p) : Real
p: ein Polygon

Beispiel: `laenge := PolyLen (poly)`

Beschreibung: Berechnet die Länge des Polygons p in Metern.

Möglicherweise vorhandene Inseln werden nicht berücksichtigt, das heißt es wird immer die Länge des Umringes berechnet.

Siehe auch PolyArea() und PolyDist().

PolyProjection

Syntax: PolyProjection(Polygon poly, GeoPoint p) : GeoPoint
poly: ein Polygon
p : ein beliebiger Punkt

Beispiel: `drauf := PolyProjection (poly, p)`

Beschreibung: Projiziert den Punkt p auf das Polygon *poly* und liefert den projizierten Punkt zurück.

Gelingt keine Projektion, weil das Lot von p aus nicht auf dem Polygon liegt, so wird der Stützpunkt geliefert, der p am nächsten ist.

PolySchnitt

Syntax: PolySchnitt (Polygon poly1, Polygon poly2) : Layer
poly1: ein Polygon
poly2: ein weiteres Polygon

Beispiel: `PolySchnitt (poly1, poly2)`

Beschreibung: Berechnet die Schnittfläche(n) der Polygone *poly*1 und *poly*2, die geschlossen sein müssen. Es wird ein Layer() zurückgeliefert, der alle Schnittflächen als Polygone enthält.

PolySection

Syntax: PolySection(Polygon p, GeoPoint p1, GeoPoint p2) : Polygon
p: ein Polygon
p1, p2 : zwei Punkte auf dem Polygon

Beispiel: `abschnitt := PolySection (p, p1, p2)`

Beschreibung: Liefert einen Ausschnitt aus p zurück, der von $p1$ bis $p2$ läuft. $p1$ und $p2$ müssen auf dem Polygon liegen, sie brauchen jedoch keine Stützstellen sein.

Falls $p1$ oder $p2$ nicht genau auf dem Polygon liegen, wird ein leeres Polygon zurückgeliefert.

Siehe auch PolyProjection()

PolySelect

Syntax: PolySelect(Polygon P, Bool onoff)
P: Ein Polygon
onoff: `TRUE` oder `FALSE`

Beispiel: `PolySelect (poly, TRUE)`

Beschreibung: Ist $onoff$ `TRUE`, dann wird das Polygon P selektiert, sonst wird es deselektiert. Wird das Polygon in einer Karte in einem GeoCanvas dargestellt (siehe NewGeoCanvas()), so wird die Änderung des Select-Zustandes erst sichtbar, wenn die Karte mittels PlotKarte() erneut dargestellt wird.

Soll die Änderung unmittelbar sichtbar werden, muss man die Prozedur SelectPolys() benutzen.

Siehe auch Selection() und KarteSelect().

PolySetAttr

Syntax: PolySetAttr(Polygon P, String attr, String wert)
P: Ein Polygon
attr: Name des Attributs
wert: neuer Wert des Attributs

Beispiel: `PolySetAttr (poly, "Farbe", "Blau")`

Beschreibung: Setzt das Attribut *attr* des Polygons *P* auf den Wert *wert*. Ist *attr* kein Attribut eines Polygons, dann hat diese Funktion keine Wirkung.

Ist das Attribut eine Zahl, muss diese mittels Str() in einen String gewandelt werden.

Ist das Attribut ein Bool, dann werden die Strings `"TRUE"` bzw. `"FALSE"` benutzt.

Zu den Attributen siehe PolyAttr().

Siehe auch LayerSetAttr().

PolySplit

Syntax: PolySplit(Polygon poly, GeoPoint p)
poly: ein Polygon
p: ein Punkt

Beispiel: `PolySplit (poly1, pickpoint)`

Beschreibung: Der Punkt *p* wird auf das Polygon *poly* projiziert. Das Segment s, welches den projizierten Punkt enthält, wird dort in zwei Teile geteilt. Alle weiteren Polygone, die ebenfalls das Segment s enthalten, werden entsprechend angepasst.

Pos

Syntax: Pos (String s1, String s2) : Real
s1:
s2:

Beispiel:
```
a := Pos ("AQUAPLAN", "QUAP")
```

Beschreibung: Gibt die erste Position des Strings $s2$ im String $s1$ an. Ist $s2$ nicht in $s1$ enthalten, wird -1 zurückgegeben. Positionen in Strings beginnen bei 0. Im Beispiel erhält `a` den Wert `1`.

Siehe auch RevPos().

PrependQuant

Syntax: PrependQuant(QuantList ql, Quant q)
ql: eine Quantenliste
q: ein Quant

Beispiel:
```
AppendQuant(ql, lq)
```

Beschreibung: Fügt das Quant lq vorne an die Quantenfolge ql an. Hinten anhängen mit AppendQuant().

Siehe auch DelQuant().

PrevTupel

Syntax: PrevTupel(Relation R, Tupel tup [, String idxfeld]) : Tupel
R: eine Mem-Relation oder eine UVS-Relation
tup: Tupel aus R
optional: idxfeld: maßgebliches Indexfeld

Beispiel:
```
t := PrevTupel(Stamm, tup, "ORT")
```

Beschreibung: Liefert das Tupel aus der Relation R, das dem Tupel tup vorangeht. Maßgeblich für die Sortierung idst $idxfeld$. Ist tup das erste Tupel in R, dann wird ein ungültiges Tupel zurückgeliefert.

Ohne Angabe eines Indexfelds wird der Standardindex benutzt (siehe CreateIndex()) oder, wenn dieser nicht erstellt wurde, die unsortierte Reihenfolge.

tup muss ein Tupel aus R sein. Ist dies nicht der Fall, ist das Ergebnis undefiniert.

Siehe auch NextTupel().

Print

Syntax: Print([p1[, p2[, p3[, ...]]]])
p1, p2, p3, ...: Beliebig viele Parameter beliebigen Typs.

Beispiel:
```
name := "Welt"
Print ("Hallo ", name, ".")
```
$\Longrightarrow$ `Hallo Welt.`

Beschreibung: Schreibt alle Parameter in der angegebenen Reihenfolge auf `stdout` (Bildschirm) und führt einen Zeilenvorschub aus. Die Anzahl der Parameter ist hierbei beliebig; ein Aufruf ohne Parameter ergibt eine Leerzeile.

Siehe auch PrintParam().

PrintFile

Syntax: PrintFile (String filename, Bool append, [p1[, p2[, p3[, ...]]]])
filename: Name der Ausgabe-Datei
append: TRUE=Datei wird erweitert, FALSE=Datei wird neu angelegt
p1, p2, p3, ...: Beliebig viele Parameter beliebigen Typs.

Beispiel:
```
name := "Welt"
PrintFile ("output.dat", FALSE, "Hallo ", name, ".")
```
$\Longrightarrow$ `Hallo Welt.` (in der Datei `output.dat`)

Beschreibung: Schreibt alle Parameter in der angegebenen Reihenfolge auf die Datei *filename* und führt dort einen Zeilenvorschub aus. Die Anzahl der Parameter ist hierbei beliebig. Ein Aufruf ohne Parameter ergibt eine Leerzeile in der Datei.

siehe auch WriteFile().

PrintPage

Syntax: PrintPage(Page page, String filename, String format)
page:
filename: Name der Ausgabedatei
format: Ausgabeformat, einer der folgenden Strings : `"HPGL"`, `"PS"`, `"PDF"`, `"SVG"`, `"FIG"`, `"DXF"`, `"WMF"`, `"PREVIEW"`, `"DRUCKER"`, `"ASCII"`, `"ASCII-APP"`

Beispiel:
```
PrintPage( page, "reportxy.plt", "HPGL" )
```

Beschreibung: Ausgabe der Page in eine Datei mit dem angegebenen Format. Die Formate `PREVIEW` und `DRUCKER` benötigen unter Unix die Kommandos lpr und gv. `PREVIEW` öffnet ein neues Fenster, in dem eine Vorabansicht der Grafik erscheint, die auch gescrollt und gezoomed werden kann. `DRUCKER` schickt die Ausgabe auf den aktuell gewählten Standarddrucker. `ASCII` gibt die Seite (ohne Grafik) im ASCII-Format (Klartext) aus, das Format `ASCII-APP` hängt die Seite im ASCII-Format an eine Datei an.

`WMF` steht für Windows Meta File. Dateien diesen Typs können unter einem MS-Windows-Betriebssytem einfach von anderen Programmen eingelesen werden.

Siehe auch NewPage() und PageOnReport()

PrintParam

Syntax: PrintParam([p1[, p2[, p3[, ...]]]])
p1, p2, p3, ...: Beliebig viele Parameter beliebigen Typs.

Beispiel:
```
name := "Welt"
welt := "Hallo"
PrintParam (name, welt)
```
$\Longrightarrow$ `name=Welt, welt=Hallo`

Beschreibung: Schreibt alle Parameter inkl. ihres Variablennamens in der angegebenen Reihenfolge auf `stdout` (Bildschirm) und führt einen Zeilenvorschub aus. Die Anzahl der Parameter ist hierbei beliebig; ein Aufruf ohne Parameter ergibt eine Leerzeile.

Siehe auch Print()

PrintReport

Syntax: PrintReport (Report rep, String filename, String format)
rep:
filename: Name der Ausgabedatei
format: Ausgabeformat, siehe PrintPage()

Beispiel:
```
PrintReport( rep, "reportxy.plt", "HPGL" )
```

Beschreibung: Ausgabe des Reports in eine Datei mit dem angegebenen Format.

Zu den Formaten siehe PrintPage(). Siehe auch PageOnReport().

PrintStatus

Syntax: PrintStatus([p1[, p2[, p3[, ...]]]])
p1, p2, p3, ...: Beliebig viele Parameter beliebigen Typs.

Beispiel:
```
name := "Welt"
PrintStatus( "Hallo ", name, "." )
```

Beschreibung: Arbeitet wie Print(), nur dass die Ausgabe, wenn die Azurfunktion als Teil einer AzurLib aus einem Aquagramm heraus aufgerufen wurde, auf der Statuszeile dieses Aquagramms erfolgt.

PrioZR

Syntax: PrioZR (ZRList zrl, Intervall i, String p,String o, Bool b [, Bool innen]) : ZR
zrl: Liste mit Zeitreihen
i: Berechnungszeitraum
p: Parameter der Ergebniszeitreihe
o: Ort der Ergebniszeitreihe
b: Temporärflag
innen: optional: nicht interpolieren, Voreinstellung: False

Beispiel:
```
zr1 := PrioZR (liste, bereich, "Abfluss", "1200", FALSE)
```

Beschreibung: Erzeugt oder erweitert eine Zeitreihe auf dem Intervall i, die sich aus den Zeitreihen in *zrl* ergibt. Die Reihenfolge in dieser Liste gibt an, mit welcher Priorität die Zeitreihen eingehen. Im Ergebnis stehen Daten der ZR mit der jeweils **höchsten Priorität, die dort nicht Lücke ist.**

Die Ergebniszeitreihe erbt die Attribute von der ersten Zeitreihe in der Liste. Die Herkunft wird auf A = abgeleitet gesetzt.

Mit *innen* kann man optional festlegen, dass die Reihe, mit der eine Lücke aufgefüllt wird, an den Lückengrenzen nicht interpoliert wird, sondern Verbindungen zum nächsten bzw. letzten inneren Punkt der Reihe erzeugt werden. Diese Option ist nur sinnvoll, wenn interpolierte Knickpunkte vermieden werden sollen, was nur in Ausnahmefällen wichtig ist.

Prompt

Syntax: Prompt([p1[, p2[, p3[, ...]]]])
p1, p2, p3, ...: Beliebig viele Parameter beliebigen Typs.

Beispiel:
```
Prompt("Bitte Ort eingeben: ")
ort := ReadLine()
```

Beschreibung: Arbeitet wie Print(), nur dass kein abschließender Zeilenumbruch erzeugt wird. Diese Prozedur kann beispielsweise dazu benutzt werden, um den Benutzer auf eine Eingabe hinzuweisen (zu „prompten“).

Siehe auch ReadLine().

Publiziert

Syntax: Publiziert (ZR zr) : Bool
zr: Eine Reihe

Beispiel: `ispub := Publiziert (kurvezr)`

Beschreibung: Liefert das Attribut Publiziert der Reihe. Wurde das Attribut der Reihe noch nicht gesetzt, ist es False. Siehe auch SetPubliziert().

Qualitaetsfolge

Syntax: Qualitaetsfolge (ZR z, Intervall i) : QuantList
z:
i: Berechnungszeitraum

Beispiel: `ql := Qualitaetsfolge (zr1, bereich)`

Beschreibung: Liefert die Folge der Qualitäten als QuantList. Die Quanten sind Intervall-Quanten, deren Höhe die Qualität auf ihrem XBereich angibt.

Quantenfolge

Syntax: Quantenfolge (ZR z, Intervall i, [Real qualy [, String readmode]]) : Quant-List
z:
i: Berechnungszeitraum
qualy: optional Qualität
readmode: optional Interpoliermodus

Beispiel: `ql := Quantenfolge(zr1, bereich)`

Beschreibung: Liefert die Folge der Quanten, aus denen sich die Zeitreihe zusammensetzt.

Wird *qualy* nicht angegeben, dann wird die höchste Qualität benutzt. Wenn man *readmode* angeben möchte, jedoch keine *qualy*, dann setzt man *qualy* auf -1, da die Parameter nur von rechts aus weggelassen werden können.

readmode gibt an, ob die Quantenfolge auf die Grenzen interpoliert werden soll (`INTERPOLIER`), ob sie auf die im Fokus liegenden Stützstellen beschränkt werden soll (`INNEN`) oder ob sie um den vorherigen und folgenden Wert erweitert werden soll, wenn die Grenzen nicht genau auf zwei Stützstellen fallen (`AUSSEN`). Die Voreinstellung ist (`INTERPOLIER`).

Die explizite Angabe des *readmodes* sollte nur erfolgen, wenn man sich über die Konsequenzen des Verlassens des allgemeinen Zeitreihenmodells im Klaren ist.

QuantNr

Syntax: QuantNr (QuantList ql, Real num) : Quant
ql: eine Quantenliste
num: Die Nummer des Quants (0..)

Beispiel:
```
q := QuantNr( ql, 10 )
```

Beschreibung: Liefert das Quant mit der laufenden Nummer *num*. Die Quanten einer QuantList sind von 0 bis AnzQuanten()-1 durchnummeriert. Liegt *num* außerhalb dieses Bereichs, dann wird ein ungültiges Quant geliefert.

Quelle

Syntax: Quelle (ZR z) : String
z:

Beispiel:
```
von := Quelle( z )
```

Beschreibung: Liefert das Attribut Quelle der Zeitreihe. Siehe Anhang.

Query

Syntax: Query(Tupel t) : ZRList
t: ein ReiheTupel

Beispiel: `liste := Query(t)`

Beschreibung: Liefert alle Zeitreihen, die auf das Muster-Tupel t passen. t muss ein ReiheTupel sein. Ein ReiheTupel kann mit den Funktionen ReiheTupel() oder Attribute() erzeugt werden. Siehe auch Tupel(), SetText() oder SetZahl().

QvonW

Syntax: QvonW (ZR wzr, Intervall i, String quelle, Bool b) : ZR
wzr: Wasserstands-Zeitreihe
i: Überarbeitungsbereich
quelle: Attribut-Quelle der Gültigkeits-ZR
b: [**?**]

Beispiel: `zrq := QvonW (zrq, focus, "", b)`

Beschreibung: Diese Funktion setzt Wasserstand (W) in Abfluss (Q) um.

quelle gibt an, welche Quelle die bei der Berechnung heranzuziehenden Zeitreihen der Gültigkeiten, der Stauwerte bzw. der Etawerte haben sollen. Wird nicht mit mehreren Quellen gearbeitet, übergibt man hier einen Leerstring.

Alle für die Berechnung nötigen Zeitreihen werden automatisch geöffnet. Dies sind

Die Gültigkeiten-Zeitreihe

Die Gültigkeiten-Zeitreihe gibt an, wann welche Abflusskurve herangezogen und wann welches Verfahren (Eta oder Stau) angewendet werden soll.

Die Real-Werte der Zeitreihe enthalten die Nummer der Abflusskurve(n). Das Verfahren ist als Intervalltext abgelegt. Bereich ohne Überdeckung mit einem Intervalltext werden als unbekannt betrachtet, im Ergebnis wird eine Lücke erzeugt.

Der Übergang von einer Abflusskurve in die nächste kann abrupt erfolgen (steiles Übergangsquant der Länge 10s) oder fließend sein. Bei einem fließenden Übergang werden die Abfluss-Werte aus beiden Abflusskurven linear interpoliert.

Zusätzlich zur Angabe des Verfahrens wird in den Intervalltexten, mit Komma getrennt, festgelegt, ob Veränderungswerte (Stauwerte oder Etawerte) benutzt werden sollen oder ob bei deren Fehlen ersatzweise 0 (frei) angenommen wird. Mögliche Texte sind also: `Eta,Mit` oder `Eta,Ohne` oder `Eta,Frei` oder `Stau,Mit` oder `Stau,Ohne` oder `Stau,Frei`

Attribute der Gültigkeiten-Zeitreihe:

Ort	wie *wzr*
SubOrt	*wzr*
Parameter	Abflusskurven
DefArt	K
Aussage	(leer)
Herkunft	O
Quelle	*quelle*
Reihenart	Z
Version	0

Abflusskurven

Abflusskurven sind Real-Reihen. Ihr Definitionsbereich (X) ist Wasserstand, ihr Wertebereich (Y) Abfluss. Die Einheit des Wasserstands (XEinheit) muss der Einheit der Wasserstands-Zeitreihe entsprechen, ist also nicht auf cm festgelegt. Die Einheit der Abflusskurve muss kompatibel zu der Einheit der Ergebniszeitreihe sein.

Abflusskurven werden sowohl für das Stauwerte-Verfahren, als auch für das Eta-Verfahren benutzt. Im Stauwerte-Verfahren gibt es (pro Gültigkeitsbereich) nur eine Abflusskurve, die das mittlere Abflussverhalten pro Wasserstand widerspiegelt, und die daher das Attribut `Mit` hat. Beim Eta-Verfahren werden zwei Abflusskurven eingesetzt, die pro Wasserstand eine Abfluss-Spanne definieren. Die Abflusskurve mit den kleineren Werten (Q_Z) hat die Aussage `Min`, die mit den höheren (Q_0) `Max`.

Jede Abflusskurve hat eine Nummer, die sie eindeutig von den anderen Abflusskurven zu der selben Station unterscheidet. Zu jeder Min-Abflusskurve gibt es auch eine Max-Abflusskurve mit der selben Nummer.

Die Hauptzeitreihe ist die Abflusskurven-Zeitreihe (Gültigkeiten-ZR).

Die Attribute einer Abflusskurve:

Ort	wie *wzr*
SubOrt	*wzr*
Parameter	Abflusskurve
DefArt	K
Aussage	Mit, Min oder Max
Herkunft	O
Quelle	(leer)
Reihenart	R
Version	Abflusskurven-Nummer

Stauwerte

Stauwerte sind eine Zeitreihe, die über die Zeit angibt, welche Korrektur am Wasserstand vorgenommen wird (um wieviel der als gestaut angenommen Wasserstand verringert werden soll).

Existiert keine Stauwerte-Zeitreihe oder enthält diese auf einem Bereich Lücke, so wird folgender Wert angenommen:

Stau,Mit : Lücke
Stau,Ohne : 0
Stau,Frei : 0

Bei Stau,Ohne wird immer 0 angenommen, auch wenn die Stauwerte-ZR existiert.

Ihre Attribute:

Ort	wie *wzr*
SubOrt	*wzr*
Parameter	Stauwert
DefArt	K
Aussage	(leer)
Herkunft	O
Quelle	*quelle*
Reihenart	Z
Version	0

Etawerte

Etawerte geben über die Zeit an, welcher Faktor η zwischen Q_Z und Q_0 angesetzt werden soll.

Für Etawerte gilt der Abschnitt Stauwerte analog. Der Parameter ist Etawert.

Random

Syntax: Random() : Real

Beispiel: `zahl := Random ()`

Beschreibung: Liefert eine Zufallszahl zwischen 0 und 1.

Rastern

Syntax: Rastern (QL ql, Distanz raster) : QL
ql: Ausgangsquantenliste
raster: Rasterbreite

Beispiel: `qf2 := Rastern(qf2, ~"1 Minute")`

Beschreibung: Quanten, die kürzer als *raster* sind, werden mit ihrem Vorgänger verschmolzen (Linker Wert ¡ Rechter Wert) oder gelöscht (Links ¿= Rechts). Wenn das Vorgänger-Quant zu lang ist, wird es so zerlegt, dass rechts ein Quant der Länge *raster* entsteht. So werden langgezogene Übergänge vermieden.

RawToReal

Syntax: RawToReal (String s, Bool lebo) : Real
s : Eine Bytefolge
lebo: True=Low Endian Byte Order

Beispiel: `zahl := RawToReal (block, False)`

Beschreibung: Wandelt eine Bytefolge, die die binäre Form einer Zahl darstellt, in diese Zahl um. Die Bytefolge kann entweder vier oder acht Bytes enthalten, jenachdem, ob es sich um eine Zahl mit einfacher oder doppelter Genauigkeit handelt.

Wenn *lebo* True ist, sind die Bytes der Bytefolge vertauscht.

Siehe auch RealToRaw().

ReadFile

Syntax: ReadFile (String filename) : String
filename: name der Datei

Beispiel:
```
alles := ReadFile ("daten.txt")
```

Beschreibung: Liest die gesamte Datei *filename* und gibt den Inhalt in einem String zurück.

Siehe auch WriteFile(), ReadLine() und FileCopy().

ReadKarte

Syntax: ReadKarte (String szenerie) : Karte
szenerie:

Beispiel:
```
K := ReadKarte ("karte.szn")
```

Beschreibung: Erzeugt eine neue Karte aus der Datei *szenerie*, welche den Aufbau der Karte (Layer, Farben, Ausschnitt) definiert.

Siehe auch WriteKarte(), PlotKarte(), KarteOnPage() und Karte().

Eine ausführliche Beschreibung des Aufbaus einer Szenerie-Datei findet sich im Handbuch [**?**].

ReadLayer

Syntax: ReadLayer (String filename [, String info]) : Layer
filename:
info: optional spezielle Steuerinformation für einige Formate

Beispiel:
```
L := ReadLayer ("isos.ai")
```

Beschreibung: Erzeugt einen neuen Layer aus der Datei *filename* und liefert diesen zurück. Das Format der Datei wird automatisch erkannt.

Siehe auch WriteLayer(), Layer() und Karte().

Eine ausführliche Beschreibung der Geo-Formate findet sich im Handbuch [?].

ReadLine

Syntax: ReadLine ([String filename]) : String
filename: name der Datei

Beispiel:
```
zeile := ReadLine ("daten.txt")
```

Beschreibung: Liest die nächste Zeile der Datei *filename*. Beim ersten Aufruf dieser Funktion wird *filename* geöffnet, alle weiteren Aufrufe greifen auf die geöffnete Datei zu. Wird ReadLine ohne Parameter aufgerufen, so wird die Zeile von der Standardeingabe (Tastatur) gelesen. Siehe auch EndOfFile(), Rewind(), FileForward() und ReadFile().

ReadPalmDB

Syntax: ReadPalmDB (String datei, String format) : Relation
datei: Dateiname der PDB-Datei
format: Tupelstruktur des Ergebnisses

Beispiel:
```
R := ReadPalmDB ("tour1.pdb", "Name#13S,WerteDa#B,Wert#10.2N")
```

Beschreibung: Liest eine Relation aus einer PDB-Datei (PalmOS Data Base).

Der Formatstring (Tupelstruktur) hat von dbf-Dateien abweichende Besonderheiten: - keine Unterstützung von Memofeldern (z.B. MEMO#S) - Zahlen dürfen nicht größer als (4.000.000.000 / Nachkommstellen) sein

Siehe auch WritePalmDB().

RealFormat

Syntax: RealFormat(String s)
s:

Beispiel: `RealFormat( "7.1f" )`

Beschreibung: Setzt das Format der Realzahlen bei der Ausgabe. Der Aufbau des Formats entspricht dem Aufbau in der Programmiersprache C.

RealToRaw

Syntax: RealToRaw (Real r, Bool lebo, Bool single) : String
r : Eine Zahl
lebo: True=Low Endian Byte Order
single: Zahl in einfacher Genauigkeit

Beispiel: `block := RealToRaw (13.5, False, False)`

Beschreibung: Liefert die interne Bytefolge der Zahl. Falls *lebo* True ist, werden die Bytes vertauscht. *single* gibt an, ob es sich um eine Zahl einfacher Genauigkeit handelt, für die nur vier Bytes ausgegeben werden. Ist *single* False werden die acht Bytes der doppelten Genauigkeit ausgegeben.

Siehe auch RawToReal().

Rechts

Syntax: Rechts (Intervall i) : Zeitpunkt
i:

Beispiel: `zp := Rechts (bereich)`

Beschreibung: Liefert die rechte Seite des (Zeit-)Intervalls i.

RechtsReal

Syntax: RechtsReal (Intervall i) : Real
i:

Beispiel:
```
zp := RechtsReal (bereich)
```

Beschreibung: Liefert die rechte Seite des (Real-)Intervalls *i*.

RecvAQTP

Syntax: RecvAQTP(String datei) : String
datei: Name der Datei auf Client-Seite

Beispiel:
```
neudatei := RecvAQTP("datei.zpa"))
```

Beschreibung: Holt die Datei *datei* vom Client und legt sie unter neuem Namen lokal im Arbeitsverzeichnis des Servers ab. Der neue Name wird zurück geliefert. Die lokale Kopie wird nach Beendigung des Programms gelöscht.

Für Anwendungen ohne AquaWeb ist diese Funktion bedeutungslos. Es wird *datei* zurückgeliefert.

Siehe auch SendAQTP().

RedrawCanvas

Syntax: RedrawCanvas()

Beispiel:
```
RedrawCanvas()
```

Beschreibung: Zeichnet, wenn vorhanden, den gesamten Canvas und den GeoCanvas neu. Siehe auch UpdateCanvas() und Plot().

Regenhoehenlinie

Syntax: Regenhoehenlinie(ZR verteilp, Intervall bereich, Real wieder, Bool temp) : ZR
verteilp: Verteilungsparameter-Zeitreihe
bereich: Auswertungszeitraum
temp: Temporärflag
wieder: Wiederkehrintervall (Jaehrlichkeit)

Beispiel:
```
rhl := Regenhoehenlinie( verteilp, [@"1980,@"1990"], 30, false )
```

Beschreibung: Berechnet eine Regenhoehenlinie mit Hilfe der Verteilungsparameter in *verteilp* und der dort beschriebenen Funktionen (siehe auch [3]) für das angegebene Wiederkehrintervall *wieder*. Die Werte der Parameter u und w werden in *verteilp* am Beginn von *bereich* gesucht (siehe Verteilungsparameter. Sind dort keine Werte abgelegt (falscher *bereich*), dann werden Lückenwerte erzeugt.

Das Ergebnis ist eine kontinuierliche Zeitreihe, die X-Werte sind relativ zum Beginn von *bereich* angelegt. Die Aussage ist auf pRL bzw. jRL gesetzt, je nachdem ob sie aus Verteilungparametern einer partiellen oder jährlichen Serie hervorgegangen ist.

RegExpMatch

Syntax: RegExpMatch(String s, String muster) : Bool
s: ein String
muster: Musterstring

Beispiel:
```
IF (RegExpMatch(name, "A*B*")
```

Beschreibung: Prüft, ob der String *name* auf den Regulären Ausdruck *muster* passt. Die Definition von Regulären Ausdrücken unterscheidet sich von der Benutzung von Wildcards (wie bei dir oder ls).

Wenn im Beispiel *name* = `"AAAABB"` ist, dann wird TRUE zurückgeliefert. `"AABBAA"` führt jedoch zu einem FALSE. `"AAAA"` führt zu einem TRUE.

Siehe auch WildcardMatch().

ReiheIDTupel

Syntax: ReiheIDTupel() : Tupel

Beispiel: `rt := ReiheIDTupel()`

Beschreibung: Liefert ein Tupel, das alle Identifikationsattribute von Reihen enthält.

Falls man damit eine Reihe öffnen möchte, muss man es vorher mittels TupPrjct() auf ein ReiheTupel() projizieren.

Reihenart

Syntax: Reihenart (ZR z) : String
z:

Beispiel: `rart := Reihenart (niederzr )`

Beschreibung: Liefert das Attribut Reihenart der Zeitreihe. „Zeitreihen“ in Azur können auch Realreihen sein, die als Reihenart R haben.

ReiheTupel

Syntax: ReiheTupel() : Tupel

Beispiel: `rt := ReiheTupel()`

Beschreibung: Liefert ein ReiheTupel. Alle Felder sind auf Leerstring bzw. 0 gesetzt. Dieses Tupel kann verändert werden und als z.B. Muster für Query() benutzt werden.

Siehe auch ReiheIDTupel().

Relation

Syntax: Relation(String name [, Bool burst]) : Relation
name: Name der Relation
burst: optional, Beschleunigung beim Schreiben

Beispiel: `R := Relation ("betreiber")`

Beschreibung: Öffnet eine Relation zum Lesen und Schreiben.

Ist *burst* `TRUE`, dann werden spätere Schreibzugriffe beschleunigt. Dies geht jedoch auf Kosten der Lesezugriff-Geschwindigkeit. Dieser Parameter sollte daher nur benutzt werden, wenn in der Hauptsache Schreibzugriffe erfolgen.

Es besteht die Möglichkeit, eine ungültige Relation anzulegen. Dazu übergibt man einen Leerstring als *name*.

Siehe auch NewRelation().

Um Relationen nur zum Lesen zu öffnen, empfiehlt sich die Funktion OpenRel(), die die Relation als MemoryRelation zurückliefert.

RelChanged

Syntax: RelChanged (Relation rel) : Bool
rel: eine Relation

Beispiel:
```
IF (RelChanged(rel))
```

Beschreibung: Gibt an, ob sich die Relation extern *rel* seit ihrem Öffnen (siehe z.B. Relation()) verändert hat.

Kann nicht auf MemRelationen angewendet werden.

Siehe auch RelModified().

RelClearModify

Syntax: RelClearModify (Relation rel) : Bool
rel: eine Mem-Relation

Beispiel:
```
RelClearModify(rel)
```

Beschreibung: Löscht den Modify-Zustand einer Relation.

Kann nur auf MemRelationen angewendet werden. Siehe auch RelModified().

RelDateMatch

Syntax: RelDateMatch (Relation quelle, S key, S dat, ZP datum) : Relation
quelle: Memory-Relation mit Zeit-Bezug
key: Name des Schlüsselfelds
dat: Name des Datumfelds
datum: maßgebliches Datum

Beispiel:
```
substrat := RelDateMatch (zeit_geber, "ORT", "BEGINN", @"Heute")
```

Beschreibung: *quelle* kann mehrere Tupel pro Key (z.B. Ort) enthalten, die sich im Datum unterscheiden. *quelle* muss eine Memory-Relation sein (siehe OpenRel()).

Zurückgegeben wird eine Relation, die pro Key das Tupel enthält, das das jüngste Datum hat, welches nicht jünger als *datum* ist. Es ist möglich, dass für Keys kein Tupel ausgewählt wird.

Siehe auch SearchAll() und WerteMenge().

RelJoin

Syntax: RelJoin (Relation rel1, Relation rel2, String keyfeld) : Relation
rel1: eine Relation
rel2: eine weitere Relation
keyfeld: Name des Schlüsselfelds

Beispiel:
```
beides := RelJoin (kerndaten, erweiterung, "ORT")
```

Beschreibung: Erzeugt eine Relation, die alle Felder aus *rel*1 und alle Felder aus *rel*2 enthält (doppelte Felder werden nur einmal genommen). RelJoin ist demnach ein **Left Outer Join**.

Für jedes Tupel aus *rel*1 gibt es im Ergebnis genau einen Eintrag. Zu jedem Tupel in *rel*1 wird das Tupel mit gleichem *key*-Feld in *rel*2 gesucht und verschmolzen. Wir kein passendes Tupel in *rel*2 gefunden, werden entsprechend der Struktur von *rel*2 leere Felder erzeugt.

Tupel aus *rel*2 können im Ergebnis mehrfach auftauchen, Tupel aus *rel* jedoch nur einmal.

Der interne Name des Ergebnisses entspricht dem von *rel*1.

Siehe auch RelPrjct(), RelSchnitt() und TupJoin().

RelKey

Syntax: RelKey () : String
rel: eine Relation

Beispiel:
```
keyfeld := RelKey (rello)
```

Beschreibung: Falls mit CreateIndex() auf *rel* ein Index erzeugt wurde, wird der Name des Keyfelds (oder die Namen der Keyfelder) geliefert, sonst ein Leerstring.

Siehe auch TupKey().

RelModified

Syntax: RelModified (Relation rel) : Bool
rel: eine Mem-Relation

Beispiel:
```
IF (RelModified(rel))
```

Beschreibung: Gibt an, ob die Relation verändert wurde. Dieser Zustand kann gelöscht werden mit der Funktion RelClearModify().

Kann nur auf MemRelationen angewendet werden.

Siehe auch IsModified() und FieldModified().

RelPrjct

Syntax: RelPrjct (Relation quelle, Relation ziel)
quelle: Quell-Relation
ziel: Ziel-Relation

Beispiel:
```
RelPrjct(oldrel, neurel)
```

Beschreibung: Projiziert alle Tupel aus *quelle* auf neue Tupel, die in *ziel* abgelegt werden. Doppelte Tupel werden gelöscht.

ziel sollte eine Memory-Relation sein, da es andernfalls zu schwerwiegenden Geschwindigkeitsproblemen kommen kann.

Siehe auch TupPrjct().

RelRewrite

Syntax: RelRewrite (Relation memrel, Relation permrel)
memrel: eine Memory-Relation
permrel: eine permanente Relation

Beispiel:
```
RelRewrite (rel, diskrel)
```

Beschreibung: *memrel* ist aus *permrel* hervorgegangen (z.B. durch DBFilter()). Dann wurde *memrel* geändert, es wurden also Tupel verändert, neue Tupel hinzugefügt oder Tupel gelöscht. Die Prozedur RelRewrite schreibt diese Änderungen zurück in die ursprüngliche, permanente Relation.

Dieses Abgleichen wird vermerkt, indem geänderte Tupel auf nicht-geändert gesetzt werden (siehe ClearModify()), die interne Liste gelöschter Tupel gelöscht wird und neue Tupel mit der, neu entstandenen, Recordnummer versehen werden (siehe TuprecNum()).

Siehe auch NewDatenbank().

RelSchnitt

Syntax: RelSchnitt (Relation rel1, Relation rel2, String keyfeld) : Relation
rel1: eine Relation
rel2: eine weitere Relation
keyfeld: Name des Schlüsselfelds (der Schlüsselfelder)

Beispiel:
```
beides := RelSchnitt (kerndaten, erweiterung, "ORT")
```

Beschreibung: Erzeugt die Schnittmenge der Relationen $rel1$ und $rel2$.

Um ein Tupel aus $rel1$ mit einem aus $rel2$ zu vergleichen, wird $keyfeld$ herangezogen. Der Feldname (oder die mit + verbundenen Feldnamen) in $keyfeld$ müssen in beiden Relationen vorhanden sein.

Die Ergebnisrelation enthält genau die Felder, die in $keyfeld$ angegeben sind, und alle Tupel, die sowohl in $rel1$ als auch in $rel2$ vorhanden sind.

Siehe auch RelJoin() und RelUnion().

RelSchreibbar

Syntax: RelSchreibbar (Relation rel) : Bool
rel: eine Relation

Beispiel:
```
IF (RelSchreibbar(rel))
```

Beschreibung: Gibt an, ob die Relation *rel* schreibbar ist.

Siehe auch ZRSchreibbar(), ExistsFile(), FileSchreibbar() und FileLesbar().

RelUnion

Syntax: RelUnion (Relation rel1, Relation rel2) : Relation
rel1: eine Relation
rel2: eine weitere Relation

Beispiel:
```
beides := RelUnion (kerndaten, erweiterung)
```

Beschreibung: Erzeugt die Vereinigungsmenge aus $rel1$ und $rel2$ in der neuen Relation *beides*. $rel1$ und $rel2$ müssen die gleiche Struktur besitzen.

Es sind danach keine doppelten Tupel in *beides* enthalten. Dies ist auch der Fall, wenn in einer der beiden Startrelationen doppelte Tupel vorhanden waren.

$rel1$ und $rel2$ müssen den selben Index besitzen. Wenn kein Index erzeugt wurde, wird das erste Feld als Index genommen.

Der interne Name des Ergebnisses lautet `union`.

Siehe auch RelJoin(), RelPrjct() und TupJoin().

Remove

Syntax: Remove (string filename)
filename:

Beispiel: `Remove("mytemp.dat")`

Beschreibung: Löscht die Datei *mytemp.dat*. Es ist darauf zu achten, dass ausreichende Zugriffsrechte vorhanden sind.

Siehe auch RemoveDir().

RemoveDir

Syntax: RemoveDir (String dirname)
filename:

Beispiel: `RemoveDir("sicherung")`

Beschreibung: Löscht das Verzeichnis *dirname*. Es ist darauf zu achten, dass ausreichende Zugriffsrechte vorhanden sind.

Siehe auch Remove().

RemoveHandle

Syntax: RemoveHandle (String elementname, String azurfunction)
elementname: Name des AGElements
azurfunktion: Name einer Azur-Funktion

Beispiel: `RemoveHandle ("gobut", "NochEinAufruf")`

Beschreibung: Meldet *azurfunktion* als Handle für das Element *elementname* ab.

Sie auch AddHandle() und SetHandle(), wo auch die Sonderfunktionen beschrieben sind.

RemoveZR

Syntax: RemoveZR (ZR z)
z:

Beispiel: `RemoveZR(zrtemp)`

Beschreibung: Löscht die Zeitreihe *zrtemp*. Diese Aktion kann nicht rückgängig gemacht werden und alle Daten der Zeitreihe gehen verloren. Es ist darauf zu achten, dass ausreichende Zugriffsrechte vorhanden sind.

Rename

Syntax: Rename (string oldname, String newname)
oldname:
newname:

Beispiel: `Rename("namen.dat", "neu.dat")`

Beschreibung: Benennt die Datei *oldname* um in *newname*.

Siehe auch Remove().

Reorganize

Syntax: Reorganize (ZR z [, Real qual)
z: eine Zeitreihe
qual: optional: eine Qualitätsstufe

Beispiel: `Reorganize(zrtemp)`

Beschreibung: Reorganisiert die Zeitreihe *z*. Es werden alle nur teilweise belegten Blöcke aufgefüllt, sodass die Datei, in der die Zeitreihe gespeichert ist, in der Regel kleiner wird. Fehlerhafte Daten, wie z.B. rückläufige Zeitpunkte, werden ebenfalls bereinigt.

Wird als zweiter Parameter eine Qualitätsstufe angegeben, so findet zusätzlich ein Löschen von Qualitätsstufen statt. Alle Qualitäten, die größer als *qual* sind, werden „heruntergedrückt“ auf *qual*.
Beispiel: `Reorganize(zrtemp,0)` schreibt die höchste Qualität in Qualität 0 und löscht alle weiteren Daten.

Replace

Syntax: Replace (string welcher, string was, string womit) : string
welcher: der String, in dem etwas ersetzt wird
was: der Teilstring, der ersetzt werden soll
womit: der Teilstring, mit dem was ersetzt wird

Beispiel:
```
s2 := Replace (sk, ",", " ")
```

Beschreibung: Erzeugt einen neuen String, der durch das Ersetzen aller Vorkommen von *was* in *welcher* durch *womit* entsteht. Wenn im Beispiel `sk` den Wert `"Gestern,Heute,Morgen"` hat, dann ist das Ergebnis `"Gestern Heute Morgen"`. Siehe auch SubStr() und Pos().

RevPos

Syntax: RevPos (String s1, String s2) : Real
s1: Gesamtstring
s2: Teilstring

Beispiel:
```
a := RevPos ("Der Derwisch", "Der")
```

Beschreibung: Gibt die letzte Position des Strings $s2$ im String $s1$ an. Ist $s2$ nicht in $s1$ enthalten, wird -1 zurückgegeben. Positionen in Strings beginnen bei 0. Im Beispiel erhält `a` den Wert `4`.

Siehe auch Pos().

Rewind

Syntax: Rewind(String filename)
filename: Name der Datei

Beispiel:
```
Rewind( "daten.txt" )
```

Beschreibung: Setzt die Datei *filename* auf die erste Zeile zurück. Siehe auch ReadLine() und EndOfFile().

Um die Datei hinter die letzte Zeile zu setzen, benutzt man die Prozedur FileForward().

Rewrite

Syntax: Rewrite(Relation R, Tupel t)
R: dbf-Relation
t: Tupel

Beispiel:
```
Rewrite (Stamm, t)
```

Beschreibung: Das Tupel t wird in die dbf-Relation R zurückgeschrieben. Wenn t aus der Relation stammt (siehe Search() oder DBFilter()), enthält es bereits die korrekte Record-Nummer. Ist t anders entstanden (z.B. durch Tupel()), dann wird es an das Ende der Relation angehängt.

Bei Mem-Relationen (siehe NewMemRelation()) darf diese Prozedur nicht verwendet werden. Tupel, die aus einer Mem-Relation stammen, müssen nicht zurückgeschrieben werden, da sie die Zugehörigkeit zu der Mem-Relation nicht verlieren können. Neue Tupel müssen mit AppTupel() eingefügt werden.

RMax

Syntax: RMax (Real r1, Real r2, ...) : Real
r1,r2,...: beliebig viele Werte

Beispiel:
```
rm := RMax (a, b, c, d, 0.5)
```

Beschreibung: Liefert das Maximum aller übergebenen Zahlen (Real).

Siehe auch RMin().

RMin

Syntax: RMin (Real r1, Real r2, ...) : Real
r1,r2,...: beliebig viele Werte

Beispiel: `rm := RMin (a, b, c, d, 0.5)`

Beschreibung: Liefert das Minimum aller übergebenen Zahlen (Real).

Siehe auch RMax().

RStr

Syntax: RStr(Real r, String f [,Bool komma [,Bool ttrenner]]) : String
r: eine Realzahl
f: das Format entsprechend RealFormat()
komma: optional: Dezimalkomma verwenden, Voreinstellung: False
ttrenner: optional: Tausender-Trenner verwenden, Voreinstellung: False

Beispiel: `s := RStr( 3.1415, "3.2f" )`

Beschreibung: Wandelt die Zahl r gemäß f in einen String um.

Ist *komma* True, so wird statt eines Dezimalpunkts ein Dezimalkomma verwendet.

Ist *ttrenner* True, so werden die Tausenderblöcke mit einem Trennzeichen getrennt. Wenn *komma* True ist, wird dazu ein Punkt benutzt, sonst ein Komma.

Siehe auch Str(), GStr() und ZPStr().

ScanRel

Syntax: ScanRel(Relation R) : Array
R: Relation, aus der Werte importiert werden sollen

Beispiel: `arr := ScanRel(rel)`

Beschreibung: Bereitet das Importieren von Wertepaaren aus R in Zeitreihen vor.

Die Relation R hat folgenden Aufbau

1. Feld: String: Stationsname
2. Feld: Datum
3. Feld: String/Zeit: Zeit
4. Feld: Real/String: Wert
5. Feld: String: Textwert

.

Die Namen der Felder sind frei.

Alle Stationsnamen, die in R enthalten sind, werden in einem Array zusammen mit dem kleinsten und größten Zeitpunkt zurückgeliefert. Der Index eines Array-Eintrags ist dabei der Stationsname, der Inhalt enthält, mit Leerzeichen getrennt, die beiden Zeitpunkte als String im Format `#Y#m#d#H#M#S`.

Auf R sollte ein Index über die ersten drei Felder erstellt worden sein. Ist dies nicht der Fall, erstellt ScanRel diesen Index. Keinesfalls darf ein anderer Index erstellt worden sein. Aus Performancegründen ist es sinnvoll, Memory-Relation zu benutzen.

Beispiel der Benutzung:

```
# dBase-Relation oeffnen
db := Relation ("werte")

# in Memory-Relation wandeln
R := DBFilter (db, Tupel(db))

# Scannen
arr := ScanRel (R)

# Schleife ueber alle Eintraege
FORALL _key IN arr
   station := _key
   von := @Token(arr[_key], 1)
   bis := @Token(arr[_key], 2)
   bereich := [von, bis]

   print (station,": ",bereich)
ENDFOR
```

Siehe auch ImportRel().

SchwellenMax

Syntax: SchwellenMax(ZR reihe, Intervall b, Real sch, Real mind, Bool tmp) : ZR
reihe: Ausgangsreihe
b: Auswertungszeitraum
sch: Schwellwert
mind: Mindestwert für Maximalwerte
tmp: Temporärflag

Beispiel: `smzr := SchwellenMax(fuellzr, bereich, 0, FALSE)`

Beschreibung: Erzeugt eine Momentan-Reihe mit Maximalwerten (Ereignissen). Die Berechnung erfolgt abschnittsweise auf Bereichen, die größer als *sch* sind. Für jeden dieser Bereiche wird genau ein Maximum bestimmt. Ist dieses Maximum kleiner als *mind*, wird es verworfen.

Siehe auch FuellenZR().

SchwellenZR

Syntax: SchwellenZR(ZR reihe, Real sch, Intervall b, Bool oben , Bool tmp) : ZR
reihe: Ausgangsreihe
sch: Schwellwert
b: Auswertungszeitraum
oben: TRUE=oben (OS), FALSE=unten (US)
tmp: Temporärflag

Beispiel:
```
szr := SchwellenZR(sammlerzr, 0.8, bereich, FALSE, FALSE)
```

Beschreibung: Erzeugt eine kontinuierliche Reihe, die abschnittsweise Lücke oder *sch* enthält. Die Attribute werden von *reihe* geerbt, die Aussage auf US oder OS gesetzt. US bedeutet, dass die Ergebnisreihe dort den Wert *sch* enthält, wo *reihe sch* überschreitet, sonst Lücke. Für OS verhält es sich genau anders herum. Hat *reihe* auf einem Intervall genau den Wert *sch*, so zählt dieses Intervall bei US als Lücke, bei OS zählt es als *sch*.

ScreenSize

Syntax: ScreenSize() : GeoPoint

Beispiel:
```
p := ScreenSize()
```

Beschreibung: Liefert die Größe des Bildschirms in Pixeln.

Search

Syntax: Search(Relation R, String s [, String sortindex]) : Tupel
R: Relation, die durchsucht werden soll
s: Muster, nach dem gesucht werden soll
sortindex: optional der zu benutzende Sortierindex (nicht für dBase)

Beispiel:
```
t := Search(stamm, "Bestadt")
```

Beschreibung: Sucht das Tupel, das auf das Muster *s* passt. Dazu wird als Voreinstellung der Schlüsselindex herangezogen, der vorher erstellt wurde (siehe CreateIndex()). Sind mehrere passende Tupel vorhanden, dann wird eines (aber nicht ein bestimmtes) geliefert. Ist kein passendes Tupel vorhanden, so wird ein invalid Tupel zurückgegeben.

Da *key* ein String ist, muss bei Nicht-String-Feldern auf das Format geachtet werden. Für numerische Felder muss *key* dem Format des Feldes entsprechend formatiert sein, also z.B. `13.70`, falls nach der Zahl 13.7 gesucht werden soll und das Format des Feldes 5.2N ist. Für Datumsfelder ist das Format JJJJMMTT und für Zeitfelder das Format HH:MM:SS zu verwenden. Siehe dazu die Funktionen RStr() und ZPStr().

Statt nach dem Schlüsselindex kann auch nach einem beliebigen Sortierindex gesucht werden. Der Feldname (oder die Feldnamen) wird dazu als Parameter *sortindex* übergeben.

Wenn sich der Schlüssel über mehrere Felder erstreckt, dann werden die einzelnen Felder im Muster mit + getrennt. Beispiel:

```
tup := Search (R, "3001005+19902306", "DBMSNR+DATVON")
```

Wenn R eine Memory-Relation ist (siehe NewMemRelation()), so liefert Search unmittelbar das Tupel aus R (also keine Kopie). Änderungen auf diesem Tupel wirken sich also direkt auch auf R aus.

Siehe auch SearchNum() und SearchFirst().

SearchAll

Syntax: SearchAll(Relation R, String key [, String sortindex]) : Relation
R: Relation, die durchsucht werden soll
key: String, nach dem gesucht werden soll
sortindex: optional der zu benutzende Sortierindex (nicht für dBase)

Beispiel:
```
mrel := SearchAll(stamm, "RRB_Frk")
```

Beschreibung: Sucht alle Tupel aus der Relation R heraus, die auf *key* passen, und speichert sie in einer MemRelation (siehe NewMemRelation()). Die Ergebnisrelation erbt den Namen der Ausgangsrelation.

Wenn kein *sortindex* ahgegeben wird, muss auf R muss mittels CreateIndex() ein Index erzeugt worden sein.

Die Tupel in der Ergebnisrelation sind **Kopien** der Tupel aus R. D.h., Änderungen auf ihnen betreffen nicht die Tupel aus R.

Siehe auch SelectAll(), CollectAll(), SearchFirst() und DBFilter().

SearchFirst

Syntax: SearchFirst(Relation R, String key [, String sortindex]) : Real
R: Relation, die durchsucht werden soll
key: Muster, nach dem gesucht werden soll
sortindex: optional der zu benutzende Sortierindex (nicht für dBase)

Beispiel:
```
i := SearchFirst(stamm, "RRB_Frk")
```

Beschreibung: Sucht die Position des ersten Tupels in R, das auf *key* passt. Wenn kein *sortindex* angegeben wird, muss ein Index auf R erzeugt worden sein (CreateIndex()). Wird kein passendes Tupel gefunden, dann wird -1 zurückgeliefert. Die Funktion SearchNum() liefert das Tupel an dieser Position.

Mit dieser Funktion ist es einfach möglich, alle Tupel zu finden, die auf *key* passen. Die an SearchNum() übergebene Position wird dazu solange um eins erhöht, bis sich das entsprechende *key*-Feld ändert.

Siehe auch Search() und SearchAll().

SearchNum

Syntax: SearchNum(Relation R, Real rank [, String sortindex]) : Tupel
R: Relation, die durchsucht werden soll
rank:
sortindex: optional der zu benutzende Sortierindex (nicht für dBase)

Beispiel:
```
t := SearchNum(stamm, 112)
```

Beschreibung: Sucht das Tupel, das im Schlüsselindex die Reihenfolgenummer *rank* hat. So kann man die Relation sortiert durcharbeiten. Im Allgemeinen ist die Benutzung dieser Funktion zu vermeiden, da sie darauf hinweist, dass der Algorithmus einer unsauberen Analyse entsprungen ist (Array-Denken). Sehr hilfreich ist sie jedoch, falls der Schlüssel nicht eindeutig ist. Ist $rank < 0$ oder $rank >$ Anzahl Tupel, so wird ein ungültiges Tupel (siehe IsValid()) zurückgegeben.

Statt nach dem Schlüsselindex kann auch nach einem beliebigen Sortierindex gesucht werden. Der Feldname (oder die Feldnamen) wird dazu als Parameter *sortindex* übergeben.

Wenn R eine Memory-Relation ist (siehe NewMemRelation()), so liefert SearchNum unmittelbar das Tupel aus R (also keine Kopie). Änderungen auf diesem Tupel wirken sich also direkt auch auf R aus.

Siehe auch SearchFirst() und SearchAll().

Sekunde

Syntax: Sekunde (Zeitpunkt zp) : Real
zp:

Beispiel:
```
s := Sekunde (tmpzp)
```

Beschreibung: Liefert die Sekunde eines Zeitpunkts. Sekunden eines Zeitpunkts werden mit einer Genauigkeit von 5 Sekunden gespeichert.

Siehe auch Jahr().

SelectAll

Syntax: SelectAll (Relation R, Bool sel, String key, String index)
R: Memory-Relation
sel: True=selektiere, False=deselektiere
key: String, nach dem gesucht werden soll
index: Feld, das nach String durchsucht wird

Beispiel:
```
SelectAll(stamm, True, "Abchausen", "ORT")
```

Beschreibung: Selektiert bzw. deselektiert alle Tupel aus R, die im Feld *index* den Eintrag *key* haben.

Mit SelectAll kann man im Zusammenspiel mit CollectAll() die Funktionalität von SearchAll() nachbilden.

Siehe auch TupSelect() und TupIsSelected().

SelectBox

Syntax: SelectBox(String frage, String buttons) : Real
frage: beliebiger Text
buttons: Mit Space getrennte Liste von Button-Labeln

Beispiel:
```
ret := SelectBox("Was machen?", "Alles Etwas Nichts")
```

Beschreibung: Erzeugt ein neues Fenster, welches den Text $frage$ und Buttons entsprechend *buttons* enthält. Das AGWindow, aus dem diese Select-Box gestartet wurde, ist solange inaktiv, bis der Benutzer einen der Buttons gedrückt hat. Der Programmablauf wird solange angehalten.

Der Rückgabewert ist die Nummer des gedrückten Buttons. Der erste Button hat die Nummer 0.

Siehe auch OkBox() und InputBox().

Selection

Syntax: Selection(Karte K) : Layer
K: eine Karte

Beispiel: `Lorte := Selection (Map)`

Beschreibung: Liefert die Polygone der Karte zurück, die (in AquaGramm) selektiert wurden. Diese können dann z.B. mit FORALL durchlaufen werden.

Siehe auch Polygon(), GeoPoint() und WriteLayer().

SelectPolys

Syntax: SelectPolys (GeoPoint lu, GeoPoint ro, Bool unselect)
lu: Gauß-Krüger-Koordinaten der linken unteren Ecke
ro: Gauß-Krüger-Koordinaten der oberen rechten Ecke
unselect: `TRUE`=die Polygone sollen deselektiert werden

Beispiel: `SelectPolys (lu, ro, \verb`FALSE?)?

Beschreibung: Selektiert oder deselektiert alle Polygone, die innerhalb des Rechtecks (lu, ro) liegen auf dem GeoCanvas des aktuellen AGWindows (siehe NewGeoCanvas() und SetAGWindow()). Die Aktion wird sofort ausgeführt und benötigt deshalb keinen Aufruf von PlotKarte().

Es werden nur aktive Layer berücksichtigt (siehe LayerSetAttr()).

Wenn ein einzelnes, bekanntes Polygon selektiert werden soll, dann lautet der Aufruf `SelectPolys (poly.Center(), poly.Center(), FALSE)`.

Siehe auch PolySelect() und SetGeoParam().

SelYAx

Syntax: SelYAx (AxBox ax, R yaxnr)
ax: eine AxBox
yaxnr: Nummer der Y-Achse (0-3)

Beispiel: `SelYAx (axbox1, 1)`

Beschreibung: Eine AxBox kann bis zu vier Y-Achsen besitzen. Im Normalfall gibt es jedoch nur eine Y-Achse. Mit dieser Funktion kann die Y-Achse gewählt werden, auf die sich die weiteren Funktionen (z.B. ZrInAxbox() oder AxSetYGitter()) beziehen.

Ohne weitere Angaben (SetAxYRechts()) wird die erste Y-Achse links mit Beschriftung außen gezeichnet. Die zweite Y-Achse erscheint rechts mit Beschriftung außen, die dritte Y-Achse links innen und die vierte rechts innen.

Siehe auch NewAxBox().

SendAQTP

Syntax: SendAQTP(String datei, Bool istext, String remotename)
datei: Name der Datei
istext: ist es eine Textdatei?
remotename: Namen, den die Datei auf der Clientseite erhalten soll

Beispiel: `SendAQTP("datei.zpa", FALSE, "datei.zpa"))`

Beschreibung: Versendet die Datei *datei* an den Client. *datei* wird nach erfolgreichem Versand auf dem Server gelöscht. *istext* legt fest, ob die Datei *datei* eine Textdatei (`TRUE`) oder eine Binärdatei (`FALSE`) ist. Textdateien werden bei Bedarf so umgewandelt, dass sie zum Betriebssystem des Clients passen.

Wenn das Programm nicht unter einem AquaWeb-Server läuft, dann wird die Datei weder verschickt noch gelöscht.

Siehe auch RecvAQTP().

SendMail

Syntax: SendMail (S smtphost, S addr, S from, S subj, S msg [, S datei, ...]) : S
smtphost: Hostname oder Hostname:Port des Mailhosts
addr: Mailadresse des Empfängers
from: Absender
subj: Betreff
msg: Anschreiben ohne Anhang
datei: optional eine oder mehrere Dateien als Anhang

Beispiel:
```
err := SendMail ("mail.ab.de", "user@ab.de", "abo@aquaplan.de", "Abo", ms
```

Beschreibung: Versendet eine Mail an *addr* per SMTP über *smtphost*. *subj* ist das Betreff, *msg* das Anschreiben. Darauf können optional ein oder mehrere Namen von Dateien folgen, die der Mail als Anhang angehängt werden.

War das Senden erfolgreich, wird ein Leerstring zurückgeliefert, sonst die Fehlermeldung.

Wenn mehrere Dateien angehängt werden sollen, können die Namen auch mit Leerzeichen getrennt in einem String übergeben werden.

Beispiel: `err := SendMail (host, user, from, "Abo", msg, "abo.pdf abo.asc zw`

statt `err := SendMail (host, user, from, "Abo", msg, "abo.pdf", "abo.asc",`

Siehe auch FTPPut().

SendSignal

Syntax: SendSignal (Real pid, Real sig) : Bool
pid: Id des Prozesses, an den das Signal gesendet werden soll
sig: Signalnummer

Beispiel:
```
SendSignal(prozess, 15)
```

Beschreibung: Schickt das Signal *sig* an den Prozess *pid*.

Der Rückgabewert gibt an, ob das Versenden erfolgreich war.

Siehe auch CatchSignal() und GetPID().

SerBinRead

Syntax: SerBinRead (Real port, Real timeout, Real nbytes [, Bool skipnull]) : S
port: Portnummer der seriellen Schnittstelle, 1-100.
timeout: Timeout in s.
nbytes: maximale Anzahl zu lesender Zeichen, oder 0.
optional: skipnull: Char(0) überlesen

Beispiel:
```
zeile := SerBinRead (2, 5, 0)
```

Beschreibung: Liest binäre Daten aus der Schnittstelle *port* aus.

Falls $nbytes > 0$ ist, wird versucht, *nbytes* zu lesen. Wenn dabei auf das nächste Zeichen mehr als *timeout* Sekunden gewartet wird, bricht das Lesen ab.

Falls $nbytes = 0$ ist, werden alle innerhalb der festen Zeit *timeout* eintreffenden Zeichen gelesen.

Falls die Schnittstelle im Blocking-Modus geöffnet ist (Synchronbetrieb), hat das *timeout* keine Bedeutung, da immer *nbytes* eingelesen werden müssen, unabhängig davon, wie lange das dauert. Im Blocking-Modus **muss** $nbytes > 0$ sein.

Mit *skipnull* kann man festlegen, ob Nullen überlesen werden sollen. Dies kann sinnvoll sein, wenn eigentlich kein Nullen auftauchen würden, die Leitung aber so schlecht ist, dass versehentlich Nullen eingestreut werden. Die Voreinstellung ist, dass Nullen gelesen werden.

Falls keine Daten anliegen, wird ein Leerstring zurückgegeben.

Die Schnittstelle muss mit SerOpen() geöffnet worden sein.

Siehe auch SerRead() und SerGet().

SerClose

Syntax: SerClose (Real port)
port: Portnummer der seriellen Schnittstelle, 1-100.

Beispiel:
```
SerClose (3)
```

Beschreibung: Beendet die Kommunikation mit einer seriellen Schnittstelle und gibt diese Resource frei.

Siehe auch SerOpen().

SerGet

Syntax: SerGet (Real port) : String
port: Portnummer der seriellen Schnittstelle, 1-100.

Beispiel: `zeichen := SerGet (1)`

Beschreibung: Liefert das nächste Zeichen, das auf der seriellen Schnittstelle anliegt.

Siehe auch SerRead() und SerBinRead().

SerOpen

Syntax: SerOpen (R port, R baud, S devparams, B blocking, B binary) : Bool
port: Portnummer der seriellen Schnittstelle, 1-100.
baud: Baudrate, z.B. 9600
devparams: Schnittstellenparameter, z.B. N,8,1
blocking: True=blocking, False=non-blocking
binary:

Beispiel: `SerOpen (1, 9600, "N,8,1", False, False)`

Beschreibung: Bereitet eine serielle Schnittstelle für die Kommunikation vor.

devparams enthält mit Kommas getrennt Parität (N=keine, E=gerade, O=ungerade), die Anzahl Bits pro Byte und die Anzahl der Stoppbits.

Hinter der Anzahl der Stoppbits kann optional mit `1` vereinbart werden, dass am Port ein Modem angeschlossen ist. Die Werte der Flusskontrolle werden dann entsprechend gesetzt. Beispiel: `SerOpen (1, 9600, "N,8,1,1")`.

Der Parameter *blocking* legt fest, ob der Lese- oder Schreibevorgang anhält, wenn die Schnittstelle keine Daten liefert (Synchronbetrieb). In diesem Fall wartet eine Lesevorgang ewig, im Non-Blocking-Modus wartet er höchstens bis zum Timeout.

binary legt fest, ob über die Verbindung Ascii- oder Binärdaten ausgetauscht werden sollen. Falls man mit SerBinRead() binäre Daten ausliest, muss man *binary* auf True setzen, damit das System CarriageReturn-Zeichen nicht in LineFeed-Zeichen umsetzt.

Siehe auch SerClose(), SerGet() und SerWrite().

Wenn das Öffnen erfolgreich war, wird True zurückgeliefert, sonst False.

SerRead

Syntax: SerRead (Real port, Real timeout, String endezeichen) : String
port: Portnummer der seriellen Schnittstelle, 1-100.
timeout: Timeout in s.
endezeichen: Zeichen, das die Zeile abschließt.

Beispiel:
```
zeile := SerRead (2, 10, Char(13))
```

Beschreibung: Liest eine Zeile (abgeschlossen mit *endezeichen*) ein. Wenn mehr als *tiemout* Sekunden verstreichen, bis das Lesen eines Zeichens abgeschlossen ist, wird die Funktion abgebrochen. In diesem Fall ist das Ergebnis Char(3).

Zeilenende, CR und LF werden nicht mitgeliefert.

Char(0) werden entfernt.

Die Schnittstelle muss mit SerOpen() geöfnet worden sein.

Siehe auch SerBinRead(), SerGet() und SerWrite().

SerReady

Syntax: SerReady (Real port) : Bool
port: Portnummer der seriellen Schnittstelle, 1-100.

Beispiel:
```
IF (SerReady (1))
```

Beschreibung: Prüft, ob Daten an der seriellen Schnittstelle zum Lesen anliegen.

Siehe auch SerRead() und SerWrite().

SerWrite

Syntax: SerWrite (Real port, String text [, Bool mitnull])
port: Portnummer der seriellen Schnittstelle, 1-100.
text: Text, der verschickt werden soll
optional: mitnull, soll ein Char(0) die Ausgabe abschließen

Beispiel:
```
SerWrite (2, "F0001\n")
```

Beschreibung: Schreibt den Text *text* auf die serielle Schnittstelle *port*. Wenn erforderlich, müssen Zeilenendezeichen mit angegeben werden (siehe Beispiel).

Wenn *mitnull* True ist, wird die Ausgabe mit einem Char(0) abgeschlossen. Die Voreinstellung ist False.

Siehe auch SerRead().

SetActive

Syntax: SetActive(String name, Bool anaus)
name: Name eines Window-Elements
anaus: Aktiv: true, Inaktiv: false

Beispiel:
```
SetActive( "eingabe1", False)
```

Beschreibung: Schaltet das Element *name* inaktiv (False) oder aktiv (True). Im inaktiven Zustand kann das Element keine Eingaben empfangen und deutet dies durch einen „Grauschleier“ an.

Siehe auch AGSetActive() und SetChangeable().

SetAGElemInfo

Syntax: SetAGElemInfo (String elename, String info)
elename: Name eines Elements
info: ein String

Beispiel:
```
SetAGElemInfo( "eingabe1", "ABC")
```

Beschreibung: Setzt den Info-String des Elements *elename* auf $info$. Der Info-String kann frei verwendet werden. Er hat keine Nebenwirkungen.

Existiert das Element nicht oder nicht auf dem aktuellen AGWindow, dann hat der Aufruf keine Wirkung.

Siehe auch AGElemInfo().

SetAGWindow

Syntax: SetAGWindow(String name)
name: Name des AGWindows

Beispiel: `SetAGWindow("AGPop2")`

Beschreibung: Setzt das aktuelle AGWindow auf *name*. Ein AGWindow ist ein Fenster des aqua_gramms, aus dem die gegenwärtige Azurfunktion aufgerufen wurde. Einfache aqua_gramme haben nur ein Fenster, welches den Namen AG0 hat. Jedes aqua_gramm hat mindestens ein Fenster mit diesem Namen. Es ist sozusagen das Hauptfenster.

Alle Ausgaben auf das Aqua_gramm (siehe z.B. Plot(), ExportVar() oder ClearCanvas()) gehen in das aktuelle Fenster.

Siehe auch GetAGWindow(), NewAGWindow() und DelAGWindow().

SetAussage

Syntax: SetAussage(ZR z, String s)
z:
s:

Beispiel: `SetAussage( zr1, "Max" )`

Beschreibung: Setzt das Attribut Aussage der Zeitreihe z auf s. **Die Benutzung dieser Funktion sollte vermieden werden.** Aussage ist ein Identifikationsattribut. Seine Änderung kann zu einem inkonsistenten Datenpool führen.

SetAutoScroll

Syntax: SetAutoScroll (Real offset, Real delay, String func)
offset: wieviel soll gescrollt werden (in Minuten)
delay: wie oft soll gescrollt werden (in Minuten)
func: Name einer Azurfunktion

Beispiel: `SetAutoScroll (15, 15, "")`

Beschreibung: Legt fest, dass alle AxBoxen alle *delay* Minuten um $offset$ Minuten in die Zukunft gescrollt werden. $func$ ist eine Azurfunktion, die nach jedem Scrollen aufgerufen wird. Diese Funktion kann auch mittels SetHandle(`"@autoscroll", func`)gesetzt werden.

Um das automatische Scrollen abzuschalten, ruft man die Funktion mit offset=0 auf.

SetAxClipping

Syntax: SetAxClipping (AxBox ax, Bool b)
ax:
b:

Beispiel: `SetAxClipping (axoben, FALSE)`

Beschreibung: Legt fest, ob Teile von Reihen, die außerhalb des Y-Achsen-Bereichs liegen, gezeichnet werden ($b = FALSE$) oder geclippt werden ($b = TRUE$). Die Voreinstellung ist $TRUE$, die Daten werden geclippt.

SetAxDist

Syntax: SetAxDist(AxBox ax, Bool anaus)
ax:
anaus: Distanz-Achse an oder aus

Beispiel: `SetAxDist(ax, True)`

Beschreibung: Legt fest, ob die X-Achse der AxBox ax als Absolut-Achse (Normalfall) oder als Distanz-Achse dargestellt werden soll. Distanz-Achsen stellen die Skalierungstexte in Distanzen relativ zum Ursprung dar. Dies ist z.B. sinnvoll für Dauerlinien.

SetAxFont

Syntax: SetAxFont (AxBox ax, String font)
ax:
font: Fontname

Beispiel: `SetAxFont(ax, "Helvetica")`

Beschreibung: Legt fest, welche Schriftart für die Darstellung der Texte einer Axbox benutzt wird. Dies betrifft die Achsenskalierungen, die Einheit und die Legenden. Die Voreinstellung ist Courier.

Siehe auch SetPageFont().

SetAxGedreht

Syntax: SetAxGedreht(AxBox ax, Bool b)
ax:
b:

Beispiel: `SetAxGedreht( axunten, TRUE )`

Beschreibung: Legt fest, ob das Achsenkreuz gedreht werden soll. Im gedrehten Zustand verläuft die X-Achse von unten nach oben und die Y-Achse von links nach rechts. Die Voreinstellung ist nicht gedreht. Siehe auch SetAxXOben, SetAxYRechts und SetTextVertical.

SetAxGitter

Syntax: SetAxGitter(AxBox ax, Bool b)
ax:
b:

Beispiel: `SetAxGitter( axbox1, TRUE )`

Beschreibung: Legt fest, ob in die AxBox an den Schnittpunkten der Skalierungsstriche Gitterkreuze gezeichnet werden. Standardmäßig ist diese Option ausgeschaltet.

Siehe auch AxSetXGitter() und AxSetYGitter()

SetAxHideBS

Syntax: SetAxHideBS (AxBox ax, Bool on)
ax: Eine AxBox
on: True oder False

Beispiel: `SetAxHideBS (ax, False)`

Beschreibung: Legt fest, ob Bereiche entsprechend ihrer Bearbeitungsstände farbig hinterlegt werden (Voreinstellung) oder weiß bleiben (*on*=True)

Siehe auch ZRInAxBox().

SetAxLage

Syntax: SetAxLage(AxBox ax, GeoPoint lu, GeoPoint ro)
ax: Eine AxBox
lu: Punkt links unten in cm
ro: Punkt rechts oben in cm

Beispiel: `SetAxLage (ax, {0,0}, {10,10})`

Beschreibung: Legt die Lage der AxBox auf der Page (siehe AxBoxOnPage()) oder dem aqua_gramm (siehe Plot()) fest. Die Angaben sind in cm. Der Ursprung des Achsenkreuzes liegt links unten (im ungedrehten Zustand).

Die Angabe der Lage und des X-Bereiches einer AxBox ist obligatorisch. Siehe auch PagePos(), SetAxXBereich(), AxBoxOnPage() und Plot().

Im aqua_gramm bezeichnet lu den Ursprung des Achsenkreuzes. Die Länge der Achsen berechnet sich als Differenz der Punkte ro und lu.

Beim Plotten auf eine Page geben lu und ro dagegen die Lage der das Achsenkreuz umrahmenden Box an. Das Achsenkreuz wird so positioniert, dass die Achsen samt Beschriftung möglichst vollständig in der Box untergebracht werden. Der Abstand des Ursprungs von lu beträgt $0.5+4\cdot textsize$ in X-Richtung und $0.5 + 2 \cdot textsize$ in Y-Richtung. Die Achsen sind jeweils 2 cm kürzer als die Box. $textsize$ ist standardmäßig 0.2 (siehe SetAxTextsize()), maßgebend ist die vor dem Aufruf von AxBoxOnPage gültige $textsize$.

Die Lage kann **nicht** gesetzt werden, wenn sich die AxBox im Vollbild-Modus befindet. Dies kann durch Anklicken in einem AquaGramm oder explizit durch SetAxVoll() veranlasst worden sein.

SetAxLegPos

Syntax: SetAxLegPos (AxBox ax, GeoPoint lo [, GeoPoint biglo])
ax: Die AxBox, zu der die Legendenposition gesetzt wird
lo: Der obere linke Punkt der Legende
biglo: optional lo, wenn die Achse großgeklickt wurde

Beispiel: `SetAxLegPos( axunten, {12,3} )`

Beschreibung: Setzt die Lage der Legende der AxBox *ax*. Der Punkt legt deren linke obere Ecke fest. Da die Anzahl der Spalten und Zeilen und die Breite jeder Spalte von den Daten abhängig ist, ist der Punkt oben links der einzige Fixpunkt.

Der zweite optionale Parameter *biglo* steuert die Lage der Legende, wenn die AxBox durch großklicken auf den ganzen Bildschirm ausgedehnt wurde. Standardmäßig ist das die linke obere Ecke der AxBox.

Soll die Legende nicht erscheinen, so setzt man *lo* auf `{-1,-1}` und *biglo* auf `{0,-1}`

SetAxLueckeModus

Syntax: SetAxLueckeModus(AxBox ax, Real lm)
ax: Eine AxBox
lm:

Beispiel: `SetAxLueckeModus (ax, 2)`

Beschreibung: Legt fest, wie Lücken von Zeitreihen in der AxBox *ax* auf der X-Achse dargestellt werden. Mögliche Werte sind: `0` = stelle keine Lücken auf der Achse dar, `1` = markiere die Vereinigungsmenge der Lückenbereiche aller Zeitreihen der AxBox mit einer gelben Linie auf der X-Achse, `2` = markiere mit einer schwarzen, gepunkteten Linie statt mit gelb.

SetAxScalDist

Syntax: SetAxScalDist (AxBox ax, Real xcm, Real ycm [, Bool nosmallticks])
ax:
xcm: Skalierungsabstand der X-Achse
ycm: Skalierungsabstand der Y-Achse
nosmallticks: optional: immer ohne kleinste unbeschriftete Skalierungsstriche, normal=False

Beispiel:
```
SetAxScalDist(ax, 1.5, 1.2)
```

Beschreibung: Legt fest, in welchem Abstand die Skalierungstexte der AxBox *ax* platziert werden sollen. Diese Vorgabe ist nur als Wunsch zu verstehen. Von den möglichen Abständen, die sich in der Berechnung der Achsen ergeben, wird der genommen, der dem Wunsch am nächsten kommt. Der Abstand kann also größer oder kleiner sein. Vorbelegung ist 1.5 und 1.2 cm.

Standardmäßig werden auf der Y-Achse bei Bedarf neben normalen und kleinen Skalierungsstrichen auch kleinste Skalierungsstriche gezeichnet. Wenn der Parameter *nosmallticks* auf `True` gesetzt wird, werden unbeschriftete nicht gezeichnet.

Siehe auch AxReplaceYTexts() und SetAxFont().

SetAxTextsize

Syntax: SetAxTextsize (AxBox ax, Real size)
ax:
size: Höhe in cm

Beispiel:
```
SetAxTextsize( axoben, 0.5 )
```

Beschreibung: Setzt die Größe der Texte, mit denen die Achsen der AxBox beschriftet werden. Die Länge der Skalierungsstriche und der Abstand dieser von den Texten wird entsprechend angepasst.

Die Voreinstellung ist 0,2 cm.

Die Größe aller anderen Texte wird mit der Prozedur AxSetTextsize() gesetzt.

SetAxVoll

Syntax: SetAxVoll(AxBox ax, Bool onoff)
ax: Eine AxBox
onoff: True=Vollbild an, False=Vollbild aus

Beispiel:
```
SetAxVoll (ax, False)
```

Beschreibung: Setzt die AxBox in den Vollbild-Modus oder daraus zurück. Ob sie im Vollbild-Modus ist, kann mittels AxInfo() abgefragt werden.

Diese Funktion ist besonders nützlich, wenn eine AxBox auf einem Aqua-Gramm vom Benutzer zum Vollbild geklickt wurde, und ihre Lage jetzt geändert werden soll. Die Funktion SetAxLage() hat nämlich keine Auswirkung, wenn die AxBox sich im Vollbild-Modus befindet.

Der Vollbild-Modus einer AxBox wird beim Plotten auf einen Canvas nicht mitübertragen.

SetAxXBereich

Syntax: SetAxXBereich(AxBox ax, Intervall bereich)
ax: Eine AxBox
bereich: X-Bereich der Achse (Zeitintervall oder Realintervall)

Beispiel:
```
SetAxXBereich (ax, [@"1.1.1980",@"1.1.1990"])
```

Beschreibung: Legt die Ausdehnung der X-Achse der AxBox fest. Legt über den Typ des Intervalls (Zeit oder Real) gleichzeitig fest, ob die X-Achse eine Zeitachse oder eine Realachse ist.

Die Angabe des XBereichs einer AxBox ist obligatorisch, bevor diese mittels AxBoxOnPage() auf eine Reportseite oder mittels Plot() auf ein aqua_gramm ausgegeben wird.

SetAxXInvers

Syntax: SetAxXInvers(AxBox ax, Bool b)
ax:
b:

Beispiel: `SetAxXInvers( axoben, TRUE )`

Beschreibung: Legt fest, ob die X-Achse der AxBox *ax* von links nach rechts (FALSE) oder von rechts nach links (TRUE) gezeichnet werden soll.

SetAxXLog

Syntax: SetAxXLog(AxBox ax, Bool b)
ax:
b:

Beispiel: `SetAxXLog( axoben, TRUE )`

Beschreibung: Legt fest, ob die X-Achse der AxBox *ax* logarithmisch gezeichnet werden soll. Dies gilt nicht für Zeitachsen.

SetAxXOben

Syntax: SetAxXOben(AxBox ax, Bool b)
ax:
b:

Beispiel: `SetAxXOben( axoben, TRUE )`

Beschreibung: Legt fest, ob die X-Achse unten (FALSE) oder oben gezeichnet werden soll.

SetAxXTexte

Syntax: SetAxXTexte(AxBox ax, Bool b)
ax:
b:

Beispiel: `SetAxXTexte( axoben, FALSE )`

Beschreibung: Legt fest, ob die X-Achse der AxBox *ax* beschriftet werden soll. Standardmäßig hat jede Achse eine Beschriftung. Diese auszuschalten kann sinnvoll sein, um Platz zu sparen, wenn mehrere AxBoxen übereinander gezeichnet werden sollen.

SetAxXUnit

Syntax: SetAxXUnit(AxBox ax, String s)
ax:
s:

Beispiel: `SetAxXUnit( axoben, "cm" )`

Beschreibung: Setzt die X-Einheit für die AxBox *ax* auf *s*. Das ist der String, der rechts unten an der Achse erscheinen soll. Üblicherweise läßt man die Einheit bei Zeitachsen weg.

SetAxYBereich

Syntax: SetAxYBereich(AxBox ax, Real ymin, Real ymax)
ax:
ymin:
ymax:

Beispiel: `SetAxYBereich( axoben, 0, 25 )`

Beschreibung: Setzt die Ausdehnung der Y-Achse der AxBox *ax*.

Standardmäßig richtet sich die Ausdehnung der AxBox nach den darin enthaltenen Zeitreihen und Konstanten. Nachdem mit SetAxYBereich die Ausdehnung explizit gesetzt worden ist, entfällt diese automatische Berechnung. Um sie wieder aktiv zu machen (und damit die explizite Einstellung der Ausdehnung wieder aufzuheben) benutzt man `SetAxYBereich(axoben, -1, -2)`.

Siehe auch SetAxYStart0().

SetAxYInvers

Syntax: SetAxYInvers(AxBox ax, Bool b)
ax:
b:

Beispiel: `SetAxYInvers( axoben, TRUE )`

Beschreibung: Legt fest, ob die Y-Achse der AxBox *ax* von unten nach oben (FALSE) oder von oben nach unten (TRUE) gezeichnet werden soll.

SetAxYLog

Syntax: SetAxYLog(AxBox ax, Bool b)
ax:
b:

Beispiel: `SetAxYLog( axoben, TRUE )`

Beschreibung: Legt fest, ob die Y-Achse der AxBox *ax* logarithmisch gezeichnet werden soll.

SetAxYRechts

Syntax: SetAxYRechts(AxBox ax, Bool b)
ax:
b:

Beispiel: `SetAxYRechts( axoben, TRUE )`

Beschreibung: Legt fest, ob die Y-Achse der AxBox *ax* auf der linken Seite (FALSE) oder der rechten Seite (TRUE) erscheinen soll.

SetAxYStart0

Syntax: SetAxYStart0 (AxBox ax, Bool anaus)
ax:
anaus: True=An, False=Aus

Beispiel: `SetAxYStart0( axoben, True )`

Beschreibung: Legt fest, ob die Y-Achse bei 0 beginnen soll. Es ist zwar möglich, den Y-Bereich explizit zu setzen (SetAxYBereich()), das setzt aber voraus, dass man den maximalen Y-Wert kennt oder zeitaufwendig berechnet.

SetAxYUnit

Syntax: SetAxYUnit(AxBox ax, String s)
ax:
s:

Beispiel: `SetAxYUnit( axoben, "m^3/s" )`

Beschreibung: Setzt die Y-Einheit für die AxBox ax auf s. Jeder Aufruf von ZRInAxBox() setzt die Y-Einheit um auf die Einheit der Zeitreihe.

SetBool

Syntax: SetBool(Tupel t, String feld, Bool b)
t: Tupel, der geändert wird
feld: Name des Feldes im Tupel
b: neuer Inhalt des feldes $feld$

Beispiel: `SetBool( t, "Eingang", True )`

Beschreibung: Setzt das Feld mit Namen $feld$ des Tupels t auf den Wert b. $feld$ muss ein Boolfeld sein.

Siehe auch SetText() und GetBool().

SetChangeable

Syntax: SetChangeable(String name, Bool anaus)
name: Name eines Window-Elements
anaus: Aktiv: true, Inaktiv: false

Beispiel:
```
SetChangeable( "eingabe1", False)
```

Beschreibung: Schaltet das Element *name* inaktiv (False) oder aktiv (True). Im inaktiven Zustand kann das Element keine Eingaben empfangen. Im Gegensatz zu SetActive() wird dies jedoch nicht durch einen „Grauschleier“ angedeutet.

SetCurrent

Syntax: SetCurrent (Array feld, ... key)
feld : ein Array
key: der Key des Eintrags, der gewählt werden soll.

Beispiel:
```
SetCurrent (feld, "B5")
```

Beschreibung: Setzt den Eintrag des Arrays *feld*, der als zurzeit aktiver ausgewählt sein soll. Dieser Eintrag kann dann mit `feld[CURRVAL]` abgefragt werden.

Der Typ des Parameters *key* ist beliebig. Wenn mit StrSplit() ein Array angelegt wurde, muss man Real benutzen.

Siehe auch Array() und StrToArray().

SetDatenpool

Syntax: SetDatenpool(String poolname)
poolname: Name des Pools

Beispiel:
```
SetDatenpool("ABT111")
```

Beschreibung: Setzt den Datenpool für Zeitreihen auf *poolname*. Dies ist der Name eines Verzeichnisses, in dem sich die Zeitreihen-Dateien befinden. Siehe auch GetDatenpool().

Warnung: Diese Funktion nicht benutzen, wenn implizite Beziehungen zwischen Zeitreihen existieren, z.B. bei ZRFolgen, oder wenn parallel mit ChangeDir gearbeitet wird. Empfehlung: diese Prozedur nicht verwenden.

SetDatum

Syntax: SetDatum (Tupel t, String feld, Zeitpunkt zp)
t: Tupel, das geändert wird
feld: Name des Feldes im Tupel
zp: neuer Inhalt des Feldes feld

Beispiel:
```
SetDatum(t, "wann", @"8.1.1996")
```

Beschreibung: Setzt das Feld mit Namen *feld* des Tupels *t* auf den Wert *zp*. *feld* muss ein Datumfeld sein. Die Stunden, Minuten und Sekunden des Zeitpunkts werden nicht berücksichtigt.

Siehe auch SetText(), SetZahl() und GetDatum().

SetDefArt

Syntax: SetDefArt(ZR z, String s)
z:
s:

Beispiel:
```
SetDefArt( zr1, "I" )
```

Beschreibung: Setzt das Attribut DefArt der Zeitreihe *z* auf *s*. **Die Benutzung dieser Funktion sollte vermieden werden.** DefArt ist ein Identifikationsattribut. Seine Änderung kann zu einem inkonsistenten Datenpool führen.

SetEditMode

Syntax: SetEditMode(String modus)
modus: Modus zum grafischen Editieren

Beispiel: `SetEditMode("CLICK")`

Beschreibung: Diese Funktion dient der Unterstützung des grafischen Editierens. Es gibt drei Modi

- `OFF`: das grafische Editieren wird ausgeschaltet.
- `CLICK`: die Funktion der linken Maustaste ändert sich auf CLICK. Bei jedem Klick in eine Axbox wird die Position in der AxBox ermittelt (XPunkt, Real) und diese an den Handle `@editmode` übergeben. Die Parameter heißen: em_xp, em_wert.
- `DRAG`: Ein Klick auf die linke Maustaste startet den Ziehmodus. Jede nennenswerte Bewegung mit der Maus bei gedrückter linker Maustaste erzeugt einen Aufruf des Handles `@editmode`. Beim Loslassen der Maustaste wird ebenfalls der Handle aufgerufen. Diesem wird die alte und neue Position des Mauszeigers (XPunkt, Real) übergeben. Die Parameter heißen: em_xp, em_wert, em_xp0, em_wert0. Der Parameter em_pressed zeigt an, ob die Maustaste gedrückt ist.

Ausführliches Beispiel:

```
SetHandle ("@editmode", "KlickFunc")
SetEditMode ("KLICK")
...

KlickFunc (Zeitpunkt em_xp, Real em_wert)
   ...
END
```

SetEinheit

Syntax: SetEinheit(ZR z, String s)
z:
s:

Beispiel: `SetEinheit( zr1, "mm/h" )`

Beschreibung: Nur in Ausnahmefällen zu verwenden. Um eine Einheit zu ändern, sollte die Prozedur Transpose benutzt werden. Eine sinnvolle Anwendung ist jedoch das Anpassen der Einheit nach einem Import eines Fremdformates. Einheiten in Azur folgen streng der SI-Norm, d.h. Groß- und Kleinschreibung ist relevant. Niederschlagsintensitäten haben z.B. die Einheit `mm/h` und nicht `MM/H`.

SetEnv

Syntax: SetEnv(String name, String wert)
name: Name der Umgebungsvariablen
wert: zuzuweisender Wert

Beispiel: `SetEnv( "Code", "A0" )`

Beschreibung: Setzt die Umgebungsvariable *name* auf den Wert *wert*.

Diese Funktion entspricht dem Betriebssytembefehl set bzw. setenv.

Siehe auch GetEnv().

SetFocus

Syntax: SetFocus(ZRList liste, Intervall bereich)
liste: ZRList, deren Focus gesetzt wird

Beispiel: `SetFocus( zrl, bereich )`

Beschreibung: Setzt den Bearbeitungszeitraum einer Reihenliste.

SetFToleranz

Syntax: SetFToleranz(ZR z, Real r)
z:
r:

Beispiel:
```
SetFToleranz( nzr, 0.01 )
```

Beschreibung: Setzt das Attribut FToleranz (Fehlertoleranz) der Zeitreihe z auf r. Fehlertoleranz ist ein wichtiges Attribut, es wird vor allem zum Entkollinearisieren und zum Verdichten von Stützstellen benötigt.

SetGeoParam

Syntax: SetGeoParam (String param)
param: Attribut2 der Polygone

Beispiel:
```
SetGeoParam ("01")
```

Beschreibung: Setzt den Typ der Polygone, die von einer Selektion erfasst werden sollen. Der Typ eines Polygons ist im Attribut2 (siehe PolyAttr()) abgelegt. Wenn alle Polygone erfasst werden sollen, muss ein Leerstring übergeben werden.

Diese Einschränkung auf ein bestimmtes Attribut2 wirkt sich sowohl auf die Polygone aus, die mit SelectPolys() aus einem Azurprogramm heraus selektiert wurden, als auch auf die Selektion, die der Benutzer direkt am Bildschirm veranlasst.

SetGlobal

Syntax: SetGlobal(String name, String wert)
name: Name der globalen String-Variablen
wert: zuzuweisender Wert

Beispiel:
```
SetGlobal( "Summe", Str(summe) )
```

Beschreibung: Setzt die globale Variable *name* auf den Wert *wert*. Ist die Variable noch nicht vorhanden, dann wird sie erzeugt.

Diese Funktionalität kann dazu benutzt werden, Variablen mit anderen Azurfunktionen, oder auch anderen AzurLibs eines AquaGramms, auszutauschen. Die Funktion Str() wandelt jeden Azurtyp in einen String um, falls die auszutauschende Variable kein String ist.

Siehe auch GetGlobal().

SetGueltBis

Syntax: SetGueltBis(ZR z, Zeitpunkt zp)
z:
zp:

Beispiel: `SetGueltBis( zr1, @"1.11.1992" )`

Beschreibung: Setzt das Ende des Gültigkeitszeitraums der Zeitreihe *z* auf *zp*. Ist die Gültigkeit der Zeitreihe noch nicht abgelaufen, dann steht das Attribut GueltBis auf Leerstring und ergibt so einen Zeitpunkt, der `Invalid` ist (siehe IsValid()). Dieser Gültigkeitszeitraum wird dazu benutzt, um die Betriebsdauer einer Messstation zu dokumentieren. Auf welchem Zeitraum die Zeitreihe wirklich Daten enthält, ist mit der Funktion MaxFocusZR() zu erfragen.

SetGueltVon

Syntax: SetGueltVon(ZR z, Zeitpunkt zp)
z:
zp:

Beispiel: `SetGueltBis( zr1, @"1.11.1954" )`

Beschreibung: Setzt den Beginn des Gültigkeitszeitraums der Zeitreihe *z* auf *zp*. Siehe auch SetGueltBis().

SetHandle

Syntax: SetHandle(String elementname, String azurfunction)
elementname: Name des AGElements
azurfunction: Name einer Azur-Funktion

Beispiel:
```
SetHandle ("gobut", "KlickAufruf")
```

Beschreibung: Normalerweise wird der Handle schon beim Erzeugen eines Elements gesetzt. Mit dieser Funktion lässt sich dieser Handle umsetzen. Das Setzen spezieller Handles (siehe unten) muss immer über diese Funktion erfolgen.

Ein Handle ist die Azurfunktion, die aufgerufen wird, wenn ein AGElement aktiviert wird. Aktivieren bedeutet z.B. Drücken eines Buttons oder einer CheckBox oder Betätigen der mittleren Maustaste in einer Karte.

Es gibt eine Reihe spezieller Handles, die nicht einem Element, sondern einer Aktion zugeordnet sind:

- Der Handle des QuitButtons wird mit `SetHandle ("@quit", func)` gesetzt. Die Funktion des Fensterschließens an sich kann jedoch so nicht beeinflusst werden.
- Der Handle, der aufgerufen wird, wenn der Focus einer AxBox oder der Ausschnitt einer Karte sich ändert, wird mit `SetHandle ("@focuschange", fu` gesetzt. Der Handle erhält den aktuellen Ausschnitt der Axbox als Spezialparameter Intervall Focus. Die betreffende AxBox wird dem handle als Spezialparameter AxBox AktAx übergeben.
- Grafisches Editieren: Der Handle, der aufgerufen werden soll, wenn der Benutzer in eine AxBox linksklickt, wird mit `SetHandle ("@editmode", func)` gesetzt. Zur Funktionsweise von editmode siehe SetEditMode().
- `SetHandle ("@canvaschange", func)` setzt den Handle, der aufgerufen wird, wenn sich die Lage von AxBoxen auf dem Canvas ändert oder die aktive AxBox wechselt. Die betreffende AxBox wird dem handle als Spezialparameter AxBox AktAx übergeben.
- Der Handle, der aufgerufen werden soll, wenn der Benutzer die Größe des Fensters ändert, wird mit `SetHandle ("@resize", func)` gesetzt.
- Der Handle, der aufgerufen werden soll, wenn der Benutzer die Selektion von Polygonen einer Karte ändert, wird mit `SetHandle ("@select", func)` gesetzt.

Ein DBGrid (siehe NewDBGrid()) hat vier verschiedene Handle. Sie werden durch Voranstellen der Handlenamen unterschieden. Die Namen der Handle sind Update, Updatefield, Select und Selectfield.

`SetHandle ("MyGrid", "Select=EineFunc")` setzt also den Handle, der beim Selektieren eines Tupels des DBGrids `MyGrid` aufgerufen wird, auf `EineFunc`. Falls man den Handle löschen will, gibt man an: `SetHandle ("MyGrid", "Sel`

Soll ein Handle gelöscht werden, so wird er auf Leerstring gesetzt.

Siehe auch AddHandle() und RemoveHandle().

SetHelpText

Syntax: SetHelpText (String name, String text)
name: Name des AG-Elements
text: Hilfetext

Beispiel: `SetHelpText ("wbutton", "Darstellung des Wasserstandes")`

Beschreibung: Setzt für das Aquagramm-Element *name* den Hilfetext auf *text*. Dieser Text wird in der Statuszeile (unter MS-Windows als Bubblehelp) angezeigt, wenn der Mauspfeil über dem Element steht.

SetHerkunft

Syntax: SetHerkunft(ZR z, String s)
z:
s:

Beispiel: `SetHerkunft( diezr, "0" )`

Beschreibung: Setzt das Attribut Herkunft der Zeitreihe z auf s. Da es sich bei der Herkunft um ein **Identifikationsattribut** handelt, sollte die Benutzung dieser Funktion **vermieden** werden.

SetHoehe

Syntax: SetHoehe(ZR zr, Real h)
zr: Eine Reihe
h: neue Höhe von *zr*

Beispiel: `SetHoehe( zr1, 154.0 )`

Beschreibung: Setzt das Attribut Hoehe der Reihe. Siehe auch Hoehe().

SetIntervallArt

Syntax: SetIntervallArt (AxBox ax, string art)
axbox:
art:

Beispiel: `SetIntervallArt (axoben, "BALKEN")`

Beschreibung: Legt fest, in welcher Darstellungsart **Intervall**-Zeitreihen in der AxBox *ax* dargestellt werden sollen. *art* kann sein: SKYLINE, BALKEN, GEFUELLT, SAEULEN, MULTISAEULEN oder STRICHE. BALKEN füllt den Bereich unter dem Quant bis zur Y-Achse aus. SAEULEN arbeitet wie BALKEN, es lässt jedoch links und rechts einen schmalen Rand. MULTISAEULEN gruppiert die Quanten aller Intervall-ZR der Axbox in jeweils einem Intervall nebeneinander.

SKYLINE ist standardmäßig gesetzt. Siehe auch ZRInAxBox()

SetInvalid

Syntax: SetInvalid((...) p)
p: Parameter beliebigen Typs.

Beispiel: `SetInvalid (stammdb)`

Beschreibung: Setzt die Variable *p* auf einen ungültigen Wert (siehe IsValid()). Da Variablen einfachen Typs (wie z.B. Zeitpunkt) kopiert werden, das Ungültigsetzen also auf der Kopie stattfindet, ist es nicht möglich, solche Variablen auf ungültig zu setzen.

Sinnvoll ist diese Prozedur bei komplexen Typen wie Tupel oder Relation. Die Zeile

```
SetInvalid (stammdb)
```

entkoppelt die Variable *stammdb* von ihrer Relation. Dies ist sinnvoll, damit die Relation geschlossen wird.

SetKommentar

Syntax: SetKommentar(ZR z, String s)
z:
s:

Beispiel: `SetKommentar( diezr, "Seltsames Simulationsergebnis" )`

Beschreibung: Setzt das Attribut Kommentar der Zeitreihe z auf s.

SetKoord

Syntax: SetKoord(ZR z, GeoPoint g)
z:
g:

Beispiel: `SetKoord(zr1, {2567890,5645322})`

Beschreibung: Setzt die geografische Koordinate der Zeitreihe. Diese ist wichtig für das automatische Bestimmen von Isohyeten oder das automatische Erzeugen von geografischen Übersichten aus dem Datenpool.

SetLebenslauf

Syntax: SetLebenslauf(ZR z, String s)
z:
s:

Beispiel: `SetLebenslauf( zr1, "Mit aqua_flux simuliert" )`

Beschreibung: Setzt den Lebenslauf der Zeitreihe z auf s. Der Lebenslauf wird grundsätzlich automatisch erzeugt, kann aber im Einzelfall anpassungswürdig sein. Siehe auch Lebenslauf().

SetLegende

Syntax: SetLegende(AxBox ax, Real rx, Real ry, String s)
ax: Die AxBox, zu der ein Legendeneintrag gesetzt wird
rx: Legendenspalte
ry: Legendenzeile
s: Legendentext an der Stelle (rx,ry)

Beispiel:
```
SetLegende( axunten, 1, 1, "Station" )
```

Beschreibung: Der Eintrag in der Spalte rx und der Zeile ry der AxBox ax wird auf s gesetzt. Die Nummerierung der Spalten und Zeilen beginnt bei 1. Diese Funktion dient der Kommunikation mit einem Aquagramm (siehe auch ZrInAxBox() oder Plot()). Die Breite der Legendenspalten wird automatisch anhand des breitesten Eintrags berechnet.

Neben Texten kann die Legende an jeder Stelle auch Linien und Symbole beliebiger Farbe und beliebiger Länge/Größe enthalten.

Linien werden durch den Sonderstring

`$linie`(Farbe,Länge,Strichmuster)

erzeugt. Farben beginnen bei 0 und sind Schwarz, Rot, Blau, Grün, Gelb, Violet, Azur, Orange, Grau, Hellgrau, Braun, Dunkelgrün, Dunkelgrau, Hellrot und Dunkelrot in dieser Reihenfolge. Die Länge wird in cm angegeben. Die Strichmuster sind in SetLineStyle erläutert.

Symbole werden durch den Sonderstring

`$sym`(Farbe,Länge,Symboltyp)

erzeugt. Länge legt nicht die „Länge“ des Symbols fest, sondern die Breite des Legendeneintrags.

Rechtecke (z.B. für Intervallzeitreihen) erzeugt man mit dem Sonderstring

`$box`(Farbe,Breite,Höhe,füllen)

Breite und *Höhe* werden in cm angegeben. *füllen* legt fest, ob das Rechteck gefüllt wird (True oder False).

Wird eine Reihe in eine AxBox gezeichnet (ZRInAxBox()), dann wird automatisch ein Legendeneintrag erzeugt. Die erste Zeile einer Legende wird dabei freigelassen, um Kommentierungen zu ermöglichen. In die erste Spalte der jeweiligen Zeile wird eine Linie in der Farbe der Reihe von 1 cm Länge gezeichnet, die zweite Spalte enthält den Text, den man mit `ZRInAxBox` festgelegt hat.

SetLineStyle

Syntax: SetLineStyle(AxBox ax, Real art)
axbox:
art:

Beispiel: `SetLineStyle(axoben, 3 )`

Beschreibung: Legt das Linienmuster fest, mit dem alle folgenden Zeitreihen und Y-Konstanten gezeichnet werden. Mögliche Werte sind:

0	solid	———
1	dashed	_ _ _ _ _
2	dotted	.....
3	longdashed	_ _ _ _
4	dashdotted	_ . _ . _ .
5	dashdotdotted	_ . . _ . .

Voreinstellung ist 0. Siehe auch ZRInAxBox().

SetLingua

Syntax: SetLingua (String sprache)
sprache: Sprachkürzel

Beispiel: `SetLingua("fr")`

Beschreibung: Setzt die Sprache, in der das Benutzer-Interface erscheint.

Das Sprachkürzel entspricht dem Feldnamen in der Datei alingua.dbf. Ist diese nocht vorhanden, oder existiert dort kein Feld *sprache*, findet keine Übersetzung im Benutzer-Interface statt. Sprachkürzel sind z.B. fr, en, es.

SetLinienArt

Syntax: SetLinienArt(AxBox ax, string art)
axbox:
art:

Beispiel: `SetLinienArt(axoben, "PUNKTE" )`

Beschreibung: Legt fest, in welcher Darstellungsart **kontinuierliche** Zeitreihen in der AxBox *ax* dargestellt werden sollen. *art* kann sein: LINIEN, PUNKTE oder DICK. LINIEN ist standardmäßig gesetzt. Siehe auch ZRInAxBox und SetSymbolTyp

SetMessgenau

Syntax: SetMessgenau(ZR z, Real r)
z:
r:

Beispiel: `SetMessgenau( diesezr, 0.1 )`

Beschreibung: Setzt das Attribut Messgenau (Messgenauigkeit) der Zeitreihe z auf r.

SetModify

Syntax: SetModify(Tupel t)
t: Tupel

Beispiel: `SetModify(tup)`

Beschreibung: Setzt explizit die Marke, die anzeigt, dass das Tupel sich verändert hat. Diese Prozedur ist fast immer überflüssig, da ein Tupel die Marke selbst verwaltet. Ausnahme sind Tupel, die neu in eine Relation aufgenommen werden, aber eine gesetzte Recordnummer haben und daher beim Zurückschreiben übersehen würden.

Siehe auch IsModified().

SetName

Syntax: SetName(String name, ... p)
name: neuer Name von p.
p: Parameter beliebigen Typs.

Beispiel: `SetName("ZAx1", axoben )`

Beschreibung: Setzt den Namen eines (komplexen) Azur-Typs, wie AxBox, Karte, Polygon, Layer. SetName auf Typen, die keinen Namen besitzen, ist wirkungslos. SetName auf Typen mit fixen Namen wie ZR oder Relation ist ebenfalls wirkungslos.

Siehe Name().

SetNWGrenze

Syntax: SetNWGrenze(ZR z, Real r)
z:
r:

Beispiel: `SetNWGrenze( diesezr, 0.125 )`

Beschreibung: Setzt das Attribut NWGrenze (Nachweisgrenze) der Zeitreihe z auf r.

SetOrt

Syntax: SetOrt(ZR z, String s)
z:
s:

Beispiel: `SetOrt (zr, "NeuerOrt")`

Beschreibung: Setzt das Attribut Ort der Zeitreihe z auf s. Diese Prozedur sollte mit Bedacht verwendet werden, da der Ort ein **Identifikations**-Attribut ist. Falls es schon eine Reihe gibt, deren Identifikationsattribute nun identisch sind, hat das Setzen des Ortes keine Wirkung.

SetPageColor

Syntax: SetPageColor(Page seite, string farbe)
seite: eine Reportseite
farbe:

Beispiel: `SetPageColor (page, "Rot")`

Beschreibung: Setzt die Farbe für Texte und Linien der Reportseite *seite* auf *farbe*. Alle folgenden Ausgaben auf diese Seite erfolgen in dieser Farbe, bis diese durch erneutes Aufrufen dieser Prozedur wieder geändert wird. Standardmäßig ist die Reportseiten-Farbe auf SSchwarzëingestellt. Die gültigen Farben sind unter ZRInAxBox() erklärt.

Siehe auch SetPageGrauton().

SetPageFont

Syntax: SetPageFont(Page seite, string fontname)
seite:
fontname:

Beispiel: `SetPageFont (page, "Helvetica")`

Beschreibung: Setzt den Font für Texte der Reportseite *seite* auf *fontname*. Alle folgenden Ausgaben auf diese Seite erfolgen in diesem Font, bis dieser durch erneutes Aufrufen dieser Prozedur wieder geändert wird.

Für die Ausgabe im PostScript-Format (siehe PrintPage()) gibt es die Fonts Courier, Helvetica und Times. Courier ist die Voreinstellung.

Unter MS-Windows können alle Fonts gewählt werden, die im System installiert sind.

Wird ein Leerstring oder ein unbekannter Font übergeben, so wird der Standardfont ausgewählt.

Siehe auch SetAxFont().

SetPageGrauton

Syntax: SetPageGrauton (Page seite, Real grauton)
seite:
grauton: in Prozent, 0 bis 100, 0=weiß, 100=schwarz

Beispiel:
```
SetPageGrauton (page, 50)
```

Beschreibung: Setzt den Grauton, in dem Text ausgegeben wird (siehe TextOnPage()). Es ist voreingestellt, dass Text schwarz ausgegeben wird. Der *grauton* gibt an, wie stark der Grauton ist, in dem die Zeichen gezeichnet werden. 0 steht für weiß, also unsichtbar, 100 für schwarz. Mit 50 erreicht man z.B. eine 50-prozentige Grautönung.

Siehe auch SetPageColor().

SetPageLineStyle

Syntax: SetPageLineStyle(Page seite, Real typ)
seite: eine Reportseite
typ:

Beispiel:
```
SetPageLineStyle (page, 3)
```

Beschreibung: Setzt das Linienmuster, in dem alle folgenden Linien auf der Seite gezeichnet werden. Linienmuster sind unter SetLineStyle() aufgeführt.

SetParameter

Syntax: SetParameter(ZR z, String s)
z:
s:

Beispiel:
```
SetParameter (zrt, "Regenspende")
```

Beschreibung: Setzt das Attribut Parameter der Zeitreihe z auf s. Diese Prozedur sollte mit Bedacht verwendet werden, da der Parameter ein **Identifikations**-Attribut ist (siehe SetOrt()).

SetPointerType

Syntax: SetPointerType(String typ)
typ: Name des Pointer-Typs

Beispiel: `SetPointerType( "clock" )`

Beschreibung: Legt fest, wie der Mauszeiger aussehen soll, wenn er sich über dem Canvas des aktuellen AGWindows befindet. Ein Leerstring steht für den Standard-Mauszeiger.

Die zu verwendenden Namen sind systemabhängig. Für ein X-Windows-System sind alle Standardnamen möglich, wobei das führende XC_ im Namen entfällt.

Siehe auch NewTrigger().

SetPubliziert

Syntax: SetPubliziert (ZR zr, B b)
zr: Eine Reihe
b: True oder False

Beispiel: `SetPubliziert (zr1, True)`

Beschreibung: Setzt das Attribut Publiziert der Reihe. Siehe auch Publiziert().

SetPunktArt

Syntax: SetPunktArt(AxBox ax, string art)
axbox:
art:

Beispiel: `SetPunktArt(axoben, "PUNKTE" )`

Beschreibung: Legt fest, in welcher Darstellungsart **Momentan**-Zeitreihen in der AxBox *ax* dargestellt werden sollen. *art* kann sein: PUNKTE, SENKRECHTE oder LINIEN. SENKRECHTE ist standardmäßig gesetzt. Siehe auch ZRInAxBox und SetSymbolTyp

SetQualitaet

Syntax: SetQualitaet(ZR z, Real qual)
z:
qual:

Beispiel: `SetQualitaet(zr1, 1)`

Beschreibung: Setzt die Qualität der Reihe z auf *qual*. Diese gilt für alle folgenden Operationen auf Zeitreihe, die ohne explizite Angabe der Qualität ausgeführt werden (z.B. YWert() und YTextWert()). Alle Operationen mit expliziter Angabe der Qualität (z.B. Quantenfolge()) heben diese Einstellung wieder auf. Siehe auch MaxQualitaet().

SetQuantText

Syntax: SetQuantText(Quant q, String t)
q:
t:

Beispiel: `SetQuantText (q0, "Ger\at defekt")`

Beschreibung: Setzt den **rechten** Text des Quants q auf t. q muss ein TextQuant sein. So kann eine TextQuantenfolge Quant für Quant erzeugt werden. Siehe auch SetYLinks().

SetQuelle

Syntax: SetQuelle (ZR z, String s)
z:
s:

Beispiel: `SetQuelle( diezr, "R" )`

Beschreibung: Setzt das Attribut Quelle der Zeitreihe z auf s. Da es sich bei der Quelle um ein **Identifikationsattribut** handelt, sollte die Benutzung dieser Funktion **vermieden** werden.

SetReihenart

Syntax: SetReihenart(ZR z, String s)
z:
s:

Beispiel:

Beschreibung: Setzt das Attribut ReihenArt der Zeitreihe z auf s. Diese Prozedur sollte **nicht angewendet** werden, da die Reihenart ein **Identifikations**-Attribut ist. Aus diesem Grunde ist hier bewusst auf ein Beispiel verzichtet worden.

SetSubOrt

Syntax: SetSubOrt (ZR z, String s)
z:
s:

Beispiel:
```
SetSubOrt (zr, "PumpeHintenIII")
```

Beschreibung: Setzt das Attribut SubOrt der Zeitreihe z auf s. Diese Prozedur sollte mit Bedacht verwendet werden, da der SubOrt ein **Identifikations**-Attribut ist. Falls es schon eine Reihe gibt, deren Identifikationsattribute nun identisch sind, hat das Setzen des SubOrtes keine Wirkung.

SetSymbolTyp

Syntax: SetSymbolTyp(AxBox ax, Real typ)
ax:
typ:

Beispiel:
```
SetSymbolTyp (ax, 4)
```

Beschreibung: Setzt den Symboltyp, in dem Punkte von Zeitreihen gezeichnet werden sollen, die im folgenden in die AxBox ax gezeichnet werden.

typ kann sein: 0=Dot, 1=Bullet, 2=Kreis, 3=Stern, 4=Kreuz, 5=Quadrat, 6=PfeilHoch, 7=PfeilRunter, 8=PfeilRechts, 9=PfeilLinks, 10=Raute. Symbolnummern > 100 sind benutzerdefinierte Symbole.

Siehe auch SetLinienArt und SetPunktArt.

SetText

Syntax: SetText(Tupel t, String feld, String s)
t: Tupel, das geändert wird
feld: Name des Feldes im Tupel
s: neuer Inhalt des Feldes feld

Beispiel:
```
SetText( t, "Name", "Skat" )
```

Beschreibung: Setzt das Feld mit Namen $feld$ des Tupels t auf den Wert s. $feld$ muss ein Textfeld sein.

SetTextVertical

Syntax: SetTextVertical(AxBox ax, Bool vertical)
ax:
vertical:

Beispiel:
```
SetTextVertical( axoben, TRUE )
```

Beschreibung: Setzt die Richtung der Textbeschriftungen der im folgenden in die AxBox ax zu zeichnenden Zeitreihen. Die Voreinstellung ist $FALSE$, also horizontal. Wird $TRUE$ übergeben, so werden die Texte vertikal gezeichnet.

SetUntrusted

Syntax: SetUntrusted (Tupel t, String feld, Bool on)
t: Tupel
feld: Name des Feldes im Tupel
on: True=auf untrusted setzen, False=auf trusted setzen

Beispiel:
```
SetUntrusted (t, "WERT", False)
```

Beschreibung: Belegt das Feld mit Namen $feld$ des Tupels t mit dem Attribut ungewiss . Das bedeutet, der Wert des Felds ist nicht vertrauenswürdig (z.B. außerhalb einer Schranke).

Wird das Tupel in einem DBGrid dargestellt (siehe NewDBGrid()), werden die ungewissen Felder in rot gezeichnet.

Siehe auch IsUntrusted().

SetVersion

Syntax: SetVersion (ZR z, Real r)
z:
r: Version, ganzahliger Wert

Beispiel:
```
SetVersion (zr, 1)
```

Beschreibung: Setzt das Attribut Version der Zeitreihe z auf r. Diese Prozedur sollte mit Bedacht verwendet werden, da die Version ein **Identifikations**-Attribut ist. Falls es schon eine Reihe gibt, deren Identifikationsattribute nun identisch sind, hat das Setzen der Version keine Wirkung. Die Version ist ein ganzzahliger Wert ohne Nachkommastellen.

Standardwert für Version ist 0.

SetWildcard

Syntax: SetWildcard (Tupel t, String feld)
t: Tupel
feld: Name des Feldes im Tupel

Beispiel: `SetWildcard (t, "UHRZEIT")`

Beschreibung: Setzt das Feld mit Namen $feld$ des Tupels t auf Wildcard. Das bedeutet, das Feld wird als nicht gesetzt markiert. Beim Filtern (siehe beispielsweise Query()) kann man damit festlegen, dass ein Feld bei allen Vergleichsmustern passt.

Siehe auch IsWildcard().

SetXBereich

Syntax: SetXBereich(Quant q, Intervall bereich)
q:
bereich:

Beispiel: `SetXBereich( lq, [@"1980",@"1981"] )`

Beschreibung: Setzt den Zeitbereich des Quantes *lq* auf *bereich*. Diese Funktion ist mit Bedacht einzusetzen, da die Quanten einer Quantenfolge sich nicht überlappen sollten.

SetXDistanz

Syntax: SetXDistanz (ZR z, String s)
z:
s:

Beispiel: `SetXDistanz( zr1, "T" )`

Beschreibung: Setzt das Attribut XDistanz der Zeitreihe z auf s. **Die Benutzung dieser Funktion sollte vermieden werden.** XDistanz ist ein Identifikationsattribut für Intervall-Reihen. Seine Änderung kann zu einem inkonsistenten Datenpool führen.

Siehe auch SetZeitschritt().

SetXEinheit

Syntax: SetXEinheit(ZR zr, String s)
zr: Eine Reihe
s: neue XEinheit von *zr*

Beispiel: `SetXEinheit( zr1, "cm" )`

Beschreibung: Setzt das Attribut XEinheit der Reihe. Siehe auch XEinheit().

SetXFaktor

Syntax: SetXFaktor (ZR z, Real r)
z:
r:

Beispiel: `SetXFaktor( zr1, 1 )`

Beschreibung: Setzt das Attribut XFaktor der Zeitreihe *z* auf *r*. **Die Benutzung dieser Funktion sollte vermieden werden.** XFaktor ist ein Identifikationsattribut für Intervall-Reihen. Seine Änderung kann zu einem inkonsistenten Datenpool führen.

Siehe auch SetZeitschritt().

SetYLinks

Syntax: SetYLinks(Quant q, Real zahl)
q:
zahl:

Beispiel: `SetYLinks(q1, 10.5)`

Beschreibung: Setzt den linken Y-Wert des Quants *q* auf *zahl*. Ist *q* ein IntervallQuant, dann ist es gleichwertig, ob der Wert mit SetYLinks() oder SetYRechts() gesetzt wird. Siehe auch SetQuantText() und SetXBereich().

SetYRechts

Syntax: SetYRechts(Quant q, Real zahl)
q:
zahl:

Beispiel: `SetYRechts(q1, wert)`

Beschreibung: Setzt den rechten Y-Wert des Quants q auf $zahl$. Siehe auch SetYLinks().

SetYTyp

Syntax: SetYTyp(ZR z, String s)
z:
s:

Beispiel:

Beschreibung: Setzt das Attribut YTyp der Zeitreihe z auf s. Diese Prozedur sollte mit Bedacht angewendet werden. Der YTyp legt fest, ob die Reihe eine Reihenreihe ist (F) oder nicht. Aus diesem Grunde ist hier bewusst auf ein Beispiel verzichtet worden.

SetZahl

Syntax: SetZahl(Tupel t, String feld, Real r)
t: Tupel, der geändert wird
feld: Name des Feldes im Tupel
r: neuer Inhalt des feldes feld

Beispiel: `SetZahl( t, "Betrag", 100.0 )`

Beschreibung: Setzt das Feld mit Namen $feld$ des Tupels t auf den Wert r. $feld$ muss ein Zahlfeld sein.

SetZahlZeichnen

Syntax: SetZahlZeichnen(AxBox ax, Bool b, String Pos)
ax:
b:
pos: OBEN, UNTEN, LINKS, RECHTS, MITTE

Beispiel: `SetZahlZeichnen( axoben, TRUE, "UNTEN")`

Beschreibung: Wenn *b* TRUE ist, dann werden an die Quanten aller Zeitreihen, die ab jetzt in die AxBox kommen, die Y-Werte als Text gezeichnet. *pos* gibt die relative Position zur Lage des Quants an.

SetZeit

Syntax: SetZeit (Tupel t, String feld, Zeitpunkt zp)
t: Tupel, das geändert wird
feld: Name des Feldes im Tupel
zp: neuer Inhalt des Feldes feld

Beispiel: `SetZeit(t, "wanngenau", @"1.1.1900 10:30")`

Beschreibung: Setzt das Feld mit Namen $feld$ des Tupels t auf den Wert zp. $feld$ muss ein Zeitfeld sein. Jahr, Monat und Tag des Zeitpunkts werden nicht berücksichtigt.

Um Zeitpunkte in Tupeln abzulegen, gibt es auch die Funktionen GetZP() und SetZP().

Siehe auch SetText(), SetZahl(), SetDatum() und GetZeit().

SetZeitschritt

Syntax: SetZeitschritt(ZR z, Distanz d)
z:
d:

Beispiel: `SetZeitschritt( zrsim, ~"1 Stunden" )`

Beschreibung: Setzt den Zeitschritt für äquidistante Intervallzeitreihen. Dieses Attribut ist ein **Identifikations**-Attribut, deshalb sollte diese Prozedur im Allgemeinen **nicht** verwendet werden.

Siehe auch SetXDistanz() und SetXFaktor().

SetZP

Syntax: SetZP (Tupel t, String feld, Zeitpunkt zp)
t: Tupel, das geändert wird
feld: Name des Feldes im Tupel
zp: neuer Inhalt des Feldes feld

Beispiel:
```
SetZP(t, "wanngenau", @"1.1.1900 10:30")
```

Beschreibung: Setzt das Feld mit Namen *feld* des Tupels *t* auf den Wert *zp*. *feld* muss ein ZP-Feld (Zeitpunkt-Feld) sein.

Siehe auch GetZP(), SetText(), SetZahl(), SetDatum() und GetZeit().

SetZPRaster

Syntax: SetZPRaster (String MonTag, Bool MTPlus, String StdMin, Bool SMMinus)
MonTag: Angabe von Tag und Monat
MTPlus: True=angegebenes Jahr, False=voriges Jahr
StdMin: Angabe von Stunde und Minute
SMMinus: True=voriger Tag, False=angegebener Tag

Beispiel:
```
SetZPRaster ("1.11", False, "7:30", False)
```

Beschreibung: Legt fest, wann ein Jahr und wann ein Tag wechselt. Dies betrifft den Operator `@`, wenn kein Tag/Monat oder keine Stunde/Minute angegeben sind.

Standardmäßig wechselt ein Jahr am 1.11. des Vorjahres und ein Tag um 7:30.

MTPlus legt fest, ob der Jahreswechsel im vorigen Jahr (False) oder im angegeben Jahr (True) erfolgen soll. *SMMinus* legt fest, ob der Tageswechsel im angegebenen Tag (False) oder im vorigen Tag (True) erfolgen soll. Für 1.1. bzw. 0:00 hat diese Angabe keine Bedeutung.

Beispiel:

`zp := @"1980"` ergibt den 1.11.1979 7:30.

Nach

`SetZPRaster("1.1.", False, "0:00", False)`

ergibt sich aus `zp := @"1980"` der 1.1.1980 0:00.

Wenn hinter dem String, der an den `@`-Operator übergeben wird, ein `+` angehängt wird, dann wird, falls kein Tag/Monat angegeben wurde, ein Jahr addiert oder falls keine Stunde/Minute angegeben wurde, ein Tag addiert.

SetZRBearbStand

Syntax: SetZRBearbStand (ZR reihe, Intervall bereich, Real stand)
reihe: zu behandelnde Reihe
bereich: Bereich, für den der Bearbeitungsstand gesetzt wird
stand: Bearbeitungsstand, 0=kein Zustand

Beispiel: `SetZRBearbStand (niederzr, bereich, 5)`

Beschreibung: Setzt die Reihe *reihe* auf dem Bereich *bereich* auf den Zustand *stand*. $stand = 5$ bedeutet **veröffentlicht**

Der Ausgangszustand einer Reihe ist vollständig **nicht gesetzter Bearbeitungszustand**.

Siehe auch ZRBearbStand(), HideZR() und Stempeln().

SignumZR

Syntax: SignumZR (ZR z, Intervall bereich, Bool tmp) : ZR
z:
bereich: Berechnungszeitraum
tmp: Temporärflag

Beispiel: `szr := SignumZR (wzr, WWJ(1995), FALSE)`

Beschreibung: Berechnet die Signumfunktion der Zeitreihe z als Intervall-Zeitreihe. Die Bereiche, in denen z positiv ist, werden zu 1, die Bereiche, in denen z 0 ist werden zu 0 und die Bereiche, in denen z negativ ist, werden zu -1 gesetzt.

Die Ergebniszeitreihe erbt alle Attribute der Ausgangszeitreihe mit Ausnahme von Aussage, welche auf SGN gesetzt wird.

SIGroesse

Syntax: SIGroesse (String s) : String
z:

Beispiel: `s2 := SIGroesse ("hl/ha")` $\Longrightarrow$ `"0.01*mm"`

Beschreibung: Wandelt s in eine SI-Groesse um und gibt dann deren String zurück.

Sin

Syntax: Sin (Real r) : Real
r:

Beispiel: `a := Sin (Pi)`

Beschreibung: Liefert den Sinus des Winkels r im Bogenmaß.

Sleep

Syntax: Sleep (Real seks)

Beispiel: `Sleep(10)`

Beschreibung: Hält das Programm für *seks* Sekunden an.

SortQuantenfolge

Syntax: SortQuantenfolge (QuantList ql)
ql: eine Quantenliste

Beispiel: `SortQuantenfolge (ql)`

Beschreibung: Sortiert die QuantList *ql* nach linkem XPunkt (siehe Links() und LinksReal()) in aufsteigender Reihenfolge.

Spalten

Syntax: Spalten (AxBox ax) : Real
ax:

Beispiel:
```
sp := Spalten( axgross )
```

Beschreibung: Die Anzahl der Spalten der Legende einer AxBox. Diese entstehen durch das Einfügen von Legendeninformation mit der Funktion SetLegende() oder ZRInAxBox().

Siehe auch Zeilen().

Stammdaten

Syntax: Stammdaten (String was) : Tupel
was: spezifiziert einen Stammdateneintrag

Beispiel:
```
stammtupel := Stammdaten ("100057")
```

Beschreibung: *was* enthält einen String, der einen Stammdateneintrag kennzeichnet. Diese Funktion liefert dazu die Stammdaten. Auf welches Feld der Stammdaten *was* passt, spielt keine Rolle. Es wird das Keyfeld (siehe ABDKeyfeld()) und alle Such-Felder durchsucht, die bei ADBInit() bzw. ADBChangeFields() übergeben wurden.

Der Aufbau des Stammdatentupels entspricht der Struktur der an ADBInit() übergebenen Stammdaten-Relation.

Ist kein passender Stammdateneintrag vorhanden, so wird ein ungültiges (IsValid()) Tupel zurückgegeben.

StammdatenByFeld

Syntax: StammdatenByFeld (String was, String feld) : Tupel
was: spezifiziert die Messstelle/Station
feld: gibt an, welches Feld durchsucht werden soll

Beispiel: `stammtupel := StammdatenByFeld ("100057", "DBMSNR")`

Beschreibung: Gibt ein Tupel mit Stammdaten-Information über die Messstelle zurück, die auf *was* passt. Dazu wird das Feld $feld$ überprüft. Wenn $feld$ ein Leerstring ist, arbeitet die Funktion wie Stammdaten().

Das Feld $feld$ kann irgendein Stammdatenfeld sein.

Siehe auch ADBInit().

Stempeln

Syntax: Stempeln(ZR reihe, Intervall bereich, Real qual)
reihe: zu stempelnde Reihe
bereich: Bereich, auf dem gestempelt wird
qual: Qualität, mit der gestempelt wird

Beispiel: `Stempeln(niederzr, bereich, 1)`

Beschreibung: Gibt der Reihe *reihe* auf dem Bereich *bereich* die Qualität *qual*. Der Reihe wird gleichsam ein Stempel aufgedrückt. Gestempelt werden kann mit Qualitäten von 0 bis 48. Reihen können durch stempeln oder durch explizites Editieren ihre Qualität ändern. Siehe auch Qualitaetsfolge().

StoreQF

Syntax: StoreQF (ZR reihe, QuantList qf, Intervall i [,Real qualy [, Bool entkol])
reihe: eine Zeitreihe
qf: eine Quantenfolge
i: Intervall, auf dem Daten gelöscht werden sollen
qualy: optional: gewünschte Qualität
entkol: optional: entkolinearisieren

Beispiel: `StoreQF(reihe, qf, gesamtfocus, 1)`

Beschreibung: Schreibt wie WriteQuantenfolge() eine Quantenfolge in eine Zeitreihe. Der Bereich, der vor dem Schreiben geleert wird, ergibt sich jedoch nicht aus der Quantenfolge, sondern wird explizit vorgegeben.

Siehe auch StoreTextQF().

StoreTextQF

Syntax: StoreTextQF (ZR reihe, QuantList qf, Intervall i [, Real qualy])
reihe: eine Zeitreihe
qf: eine Quantenfolge
i: Intervall, auf dem Daten gelöscht werden sollen
qualy: optional: gewünschte Qualität

Beispiel: `StoreTextQF(reihe, qf, gesamtfocus, 1)`

Beschreibung: Schreibt wie WriteTextQuantenfolge() eine Quantenfolge in eine Zeitreihe. Der Bereich, der vor dem Schreiben geleert wird, ergibt sich jedoch nicht aus der Quantenfolge, sondern wird explizit vorgegeben.

Siehe auch StoreQF().

Str

Syntax: Str((...) p) : String
p: Parameter beliebigen Typs.

Beispiel: `s := Str (30.5)`

Beschreibung: Wandelt den Parameter p in den String s um. Zeitpunkte werden anhand von zpmode() umgewandelt, Realzahlen anhand von RealFormat().

Siehe auch RStr(), GStr() und ZPStr().

StrAnz

Syntax: StrAnz(String s , String teil) : Real
s: Gesamtstring
teil: Teilstring

Beispiel: `anz := StrAnz ("lalLalalala", "lal")` $\Longrightarrow 2$

Beschreibung: Gibt zurück, wie oft der Teilstring *teil* im Gesamstring *s* enthalten ist. Überschneidungen werden nur einmal gezählt und Groß-Klein-Schreibung wird berücksichtigt.

StrDecrypt

Syntax: StrDecrypt(String s) : String
s : Ein String

Beispiel: `wally := StrDecrypt (dilbert)`

Beschreibung: Entschlüsselt den String *s*, der vorher verschlüsselt wurde.

Siehe auch StrEncrypt().

StrEncrypt

Syntax: StrEncrypt(String s) : String
s : Ein String

Beispiel: `dilbert := StrEncrypt (password)`

Beschreibung: Verschlüsselt den String *s*.

Siehe auch StrDecrypt().

StrFromHex

Syntax: StrFromHex (String s) : String
s : Ein Hex-String

Beispiel: `s := StrFromHex (hex)`

Beschreibung: Wandelt einen String, der in Hexadezimaldarstellung angegeben ist, in die entsprechende binäre Bytefolge. Leerzeichen, Tabs und Zeilenumbrüche werden ignoriert.

Beispiel: `StrFromHex("48 61 6C 6C 6F")` ergibt Hallo

Siehe auch StrHex().

StrHead

Syntax: StrHead (String s, Real n) : String
s:
n: Anzahl Zeichen

Beispiel: `s2 := StrHead ("Ameisenfalle",4)` $\Longrightarrow$ `"Amei"`

Beschreibung: Liefert die ersten n Zeichen von s. Wenn n negativ ist, besteht das Ergebnis aus s **ohne** die letzten n Buchstaben.

Beispiel: Sei s = `"Hochhaus"`, dann ist StrHead(s,3) = `"Hoc"` und StrHead(s,-3) = `"Hochh"`.

Siehe auch StrTail() und SubStr().

StrHex

Syntax: StrHex (String s) : String
s : Ein String

Beispiel: `hex := StrHex (s)`

Beschreibung: Erzeugt die Hexadezimaldarstellung aller Zeichen in s mit Leerzeichen getrennt.

Beispiel: `StrHex("Hallo"+Char(10)")` ergibt 48 61 6C 6C 6F 0A.

Siehe auch StrFromHex().

StrHunz

Syntax: StrHunz (String s) : String
s : Ein String

Beispiel:
```
hunzstr := StrHunz (echtstr)
```

Beschreibung: Wandelt Umlaute und ß in Ascii-Zeichen um.

StrSort

Syntax: StrSort(String sl [, String sep]) : String
s: Ausgangsstring
sep: optional Separator, Voreinstellung ist Leerzeichen

Beispiel:
```
sl2 := StrSort (sl)
```

Beschreibung: *sl* enthält eine mit dem Zeichen *sep* (typischerweise das Leerzeichen) getrennte Liste von Wörtern (tokens). Diese Liste wird sortiert und, wiederum mit *sep* getrennt, zurückgeliefert.

Wörter, die *sep* enthalten, müssen in Gänsefüßchen eingeschlossen werden.

Siehe auch Token().

StrSplit

Syntax: StrSplit(String s, String trenner) : Array
s : Ein String, der Wörter enthält
trenner: Trennzeichen

Beispiel:
```
feld := StrSplit (eingabe, ",")
```

Beschreibung: Zerlegt *s* in Wörter (Tokens), die durch das Zeichen *t* (muss genau ein Zeichen lang sein) getrennt sind (Token()).

Die Elemente haben die Indexe 0 bis ArraySize()-1.

Siehe auch StrToArray().

StrTail

Syntax: StrTail (String s, Real n) : String
s:
n: Anzahl Zeichen

Beispiel: `s2 := StrTail ("Ameisenfalle",4)` $\Longrightarrow$ `"alle"`

Beschreibung: Liefert die letzten n Zeichen von s. Wenn n negativ ist, besteht das Ergebnis aus s **ohne** die ersten n Zeichen.

Beispiel: Sei s = `"Trampolin"`, dann ist StrTail(s,3) = `"lin"` und StrTail(s,-3) = `"mpolin"`.

Siehe auch StrHead() und SubStr().

StrToArray

Syntax: StrToArray(String s) : Array
s : Ein String, der das gesamte Array enthält

Beispiel: `feld := StrToArray (eingabe)`

Beschreibung: Erzeugt ein Array aus dem String s. Dieser enthält das gesamte Array in kompakter Form. s kann lediglich eine mit Space getrennte Liste von Wörtern (Elementen) enthalten. Die Indexe zu den Elementen sind die jeweiligen Reihenfolgenummern. Die Indexe können jedoch auch explizit angegeben werden. Sie müssen dann, mit einem `|`-Zeichen getrennt, den Elementen vorangestellt sein.

Beispiel:

```
s := "Bowmore Macallen Springbank oben|Talisker"
```

`Springbank` wird der Index 2 zugeordnet, `Talisker` der Index `oben`

Siehe auch StrSplit(), Array() und Str().

StrToAxBox

Syntax: StrToAxBox (String s) : AxBox
s: ein String, der eine AxBox repräsentiert

Beispiel:
```
ax := StrToAxBox (str)
```

Beschreibung: Wandelt den String s in eine AxBox um. Das Format dieses Strings ist hochkomplex und eignet sich nicht dazu, vom Benutzer manuell erzeugt zu werden. Die Funktion ImportVar() kann stattdessen dazu benutzt werden, eine AxBox als String aus einem Aquagramm zu importieren. Mit StrToAxBox wird dieser String dann in eine AxBox umgesetzt.

Siehe auch StrToReal().

StrToBase64

Syntax: StrToBase64(String s, Bool urlsafe) : String
s : Ein String
urlsafe: Zeichensatzsteuerung

Beispiel:
```
dilbert := StrToBase64 (user+":"+password, True)
```

Beschreibung: Erzeugt die Base64-Darstellung von s. Dies geschieht nach RFC3548 unter Verwendung von `_` und `-` ($urlsafe$=True) oder `/` und `+` ($urlsafe$=False).

Siehe auch Base64ToStr() und MD5Sum().

StrToKarte

Syntax: StrToKarte (String s) : Karte
s: ein String, der eine Karte repräsentiert

Beispiel:
```
map := StrToKarte (str)
```

Beschreibung: Wandelt den String *s* in eine Karte um. Das Format dieses Strings eignet sich nicht dazu, vom Benutzer manuell erzeugt zu werden. Die Funktion ImportVar() kann stattdessen dazu benutzt werden, eine Karte als String aus einem Aquagramm zu importieren. Mit StrToKarte wird dieser String dann in eine Karte umgesetzt.

Siehe auch StrToAxBox().

StrToReal

Syntax: StrToReal (String s) : Real
s: ein String, der eine Realzahl repräsentiert

Beispiel:
```
r := StrToReal ("3.14")
```

Beschreibung: Wandelt den String *s* in eine Realzahl um.

Wenn aus *s* keine Zahl zu gewinnen ist, wird 0 geliefert. Ein Leerstring ergibt beispielsweise 0.

Im Gegensatz zu IsReal() wird *s* nicht vollständig ausgewertet, sondern nur bis zu dem Zeichen, bis zu dem sich eine Zahl ergibt. Wenn ein Zeichen erreicht wird, das zusammen mit den vorhergehenden Zeichen keine Zahl ergeben würde, wird die Auswertung vorher beendet. Beispiele: 10abc ergibt 10, 45-8 ergibt 45, 1.2e3 ergibt 1200 und 1.2e3.4 auch 1200.

Die Strings Luecke und Lücke (Groß-Kleinschreibung beachten) ergeben die Konstante [**?**].

Struktur

Syntax: Struktur (Relation Rel — Tupel tup) : String
Rel: eine Relation **oder** tup: ein Tupel

Beispiel: `s := Struktur (stammdaten)`

Beschreibung: Liefert die Struktur, also den Aufbau der Felder, einer Relation oder eines Tupels. Diese kann z.B. dazu benutzt werden, um ein Tupel zu erzeugen (siehe Tupel()).

STUDTest

Syntax: STUDTest (ZR zr, Intervall bereich, Real alpha) : Tupel
zr: zu überprüfende Zeitreihe (Stichprobe)
bereich: zu überprüfender Zeitraum der Zeitreihe
alpha: Signifikanzniveau (0,1)

Beispiel: `tup := STUDTest(zrem, MaxFocusZR(zrem), 0.05)`

Beschreibung: Testet, ob die Zeitreihe *zr* auf dem Ausschnitt *bereich* keinen **signifikanten linearen Trend** aufweist.

alpha gibt die Irrtumswahrscheinlichkeit der Aussage vor. Es sind nur die Werte $0.5, 0.2, 0.1, 0.05, 0.02, 0.01, 0.002, 0.001 und 0.0001$ möglich.

Das Ergebnis des Tests (abgelehnt) und die nach STUDENT verteilte Prüfgröße t sind Felder des Ergebnistupels.

Die Berechnung erfolgt nach [2]. Die Tabelle mit Signifikanzschranken ist jedoch, weil umfangreicher, [6] entnommen. Um zu gewährleisten, dass auch bei negativen Trends eine korrekte Signifikanzbetrachtung erfolgt, wird t als Absolutbetrag berechnet. Die Zeitreihe muss auf *bereich* mindestens drei Werte enthalten. Angenommen wird, dass wenn ein Trend vorliegt, dieser linear ist. Liegt ein nichtlinearer Trend vor, dann läuft die Berechnung ins Leere und liefert keine korrekte Aussage.

Ungültige Werte von *alpha* führen zu einem ungültigen Tupel Siehe auch Trend() und KSTest().

Stunde

Syntax: Stunde (Zeitpunkt zp) : Real
zp:

Beispiel: `s := Stunde (tmpzp)`

Beschreibung: Liefert die Stunde eines Zeitpunkts.

Siehe auch Minute().

SubGroesse

Syntax: SubGroesse (ZR z, String sg, Intervall i, String sp, Bool b) : ZR
z:
sg: Größe
i: Berechnungszeitraum
sp: Parameter der neuen Zeitreihe
b: Temporärflag

Beispiel: `sum := SubGroesse (zr1, "30 mm/h", i, "Nieder2", FALSE)`

Beschreibung: Es wird eine neue Zeitreihe erzeugt, die sich aus der Subtraktion der Zeitreihe *z* und der Konstanten *sg* ergibt. *sg* ist als einheitbehaftete Zahl aufzufassen, deren Einheit mit der der Zeitreihe *z* kompatibel ist. Der Parameter der neuen Zeitreihe wird auf *sp* gesetzt.

SubOrt

Syntax: SubOrt (ZR z) : String
z:

Beispiel: `wogenau := SubOrt (z)`

Beschreibung: Liefert das Attribut *SubOrt* der Zeitreihe.

SubStr

Syntax: SubStr (String s, Real von[, Real bis]) : String
s:
von: Startposition
bis: Endposition (optional)

Beispiel: `s2 := SubStr ("Ameisenfalle",4,7 )` $\Longrightarrow$ `"senf"`

Beschreibung: Liefert den Teilstring von der Position *von* bis *bis* des Strings s. Positionen starten bei 0. Wenn *bis* nicht angegeben wird, so wird das Zeichen an der Position *von* geliefert.

Siehe auch StrHead() und StrTail().

SubZR

Syntax: SubZR (ZR z1, ZR z2, ZI i, String s, Bool b [, Bool lueckzunull]) : ZR
z1:
z2:
i: Berechnungszeitraum
s: Ort der neuen Zeitreihe
b: [**?**] lueckezunull: optional: Sollen Lücken als 0 gelten.

Beispiel: `sum := SubZR (z1, z2, MAXFOCUS, s, FALSE)`

Beschreibung: Subtrahiert zwei Zeitreihen. Die Einheiten müssen kompatibel sein. Handelt es sich um Intervall-Zeitreihen, dann müssen die Intervallmengen exakt übereinstimmen. Bei kontinuierlichen Zeitreihen findet eine Vereinigung der Stützpunktmengen statt. Siehe auch AddZR().

Ist *lueckezunull* True, werden Lücken in $z1$ oder $z2$ als 0 gewertet. Wenn sowohl $z1$ als auch $z2$ eine Lücke aufweisen, ist das Ergbnis **nicht 0, sondern Lücke**. Die Voreinstellung für *lueckezunull* ist False.

Summe

Syntax: Summe (ZR reihe, Intervall i [, R l]) : Real
reihe:
i: Berechnungszeitraum
l: optional Lückenlimit in % (Voreinstellung 100)

Beispiel:
```
r := Summe(reihe, breite)
```

Beschreibung: Bestimmt die Summe der Zeitreihe *reihe* über dem Zeitraum *i*. Wenn *reihe* eine Momentan-Reihe ist, oder Aussage Sum ist, dann ergibt sich die Summe als Summe aller in *i* liegenden Werte. Ansonsten wird das bestimmte Integral berechnet.

Der optionale Parameter *l* legt fest, ab wieviel Prozent lückenhafter Daten eine Lücke erzeugt wird. Wird dieser Wert nicht angegeben, so wird erst eine Lücke für das Intervall erzeugt, wenn die Zeitreihe dort ausschließlich Lücke ist.

SummenHeberLinien

Syntax: SummenHeberLinien (ZR z, Intervall i, String param, Real ymin, Real ymax, R ystart, Bool b) : ZR
z:
i: Berechnungszeitraum
param: Parameter der Ergebniszeitreihe
ymin: Minimaler Y-Wert
ymax: Maximaler Y-Wert. Bei ymax wird auf ymin abgehebert
ystart: Startwert der Linie
b: Temporärflag

Beispiel:
```
int := SummenHeberLinien(z, i, "N", 0, 10, 0, FALSE)
```

Beschreibung: Wandelt die Zeitreihe *z* in Summenlinien um, die jeweils bei Erreichen des Y-Werts *ymax* auf den Wert *ymin* abhebern. Mit Hilfe dieser Funktion ist es möglich, Zeitreihen in der Form eines Schreiberaufschriebes zu betrachten.

Die Berechnung der Einheit findet automatisch statt.

SummenLinien

Syntax: SummenLinien (ZR z, Intervall i, Distanz d, Bool b) : ZR
z:
i: Berechnungszeitraum
d: Breite eines Auswertungsintervalls
b: Temporärflag

Beispiel: `int := SummenLinien (z, i, ~"1 Monat" , FALSE)`

Beschreibung: Berechnet das unbestimmte Integral der Zeitreihe z jeweils auf Intervallen der Breite d. Bei jedem Intervallanfang wird mit 0 begonnen. Die Werte werden dazu integriert (zum Algorithmus siehe [4]). Die Berechnung der Einheit findet automatisch statt.

SummenWerte

Syntax: SummenWerte (ZR z, Intervall i, Distanz d, Bool b [, R l]) : ZR
z:
i: Berechnungszeitraum
d: Breite der Intervalle, für die die Berechnung stattfindet
b: Temporärflag
l: optional Lückenlimit in % (Voreinstellung 100)

Beispiel: `sw := SummenWerte(z, MAXFOCUS, ~"5 Minuten", FALSE`)

Beschreibung: Erzeugt eine Intervall-Zeitreihe, deren Intervalle die Breite d haben. d kann z.B. `~"5 Minuten"` sein. Der Y-Wert zu diesen Intervallen ist jeweils das bestimmte Integral über dem Intervall. Die Aussage der Ergebniszeitreihe wird auf Sum gesetzt, die Intervallbreite wird in den Attributen XDistanz und XFaktor (siehe dazu Zeitschritt()) vermerkt, die Herkunft wird auf A gesetzt, alle weiteren Attribute vererbt. Die Berechnung der Einheit findet automatisch statt.

Der optionale Parameter l legt fest, ab wieviel Prozent lückenhafter Daten eine Lücke erzeugt wird. Wird dieser Wert nicht angegeben, so wird erst eine Lücke für das Intervall erzeugt, wenn die Zeitreihe dort ausschließlich Lücke ist.

SyncZR

Syntax: SyncZR (ZR z, Intervall be, ZR szr, Real vqual, R bqual, Bool neu) : ZR
z: Ausgangsreihe
be: Berechnungszeitraum
szr: Momentan-Zeitreihe mit Synchropunkten
vqual: Ausgangsqualität aus *z*
bqual: Zielqualität
neu: True=neue Reihe anlegen

Beispiel:
```
zr := SyncZR (zr,bereich, synchrozr, 0, 1, False)
```

Beschreibung: Synchronisiert eine Reihe mit Hilfe einer Synchropunkte-Reihe. Die zu synchronisierenden Daten werden der Qualität *vqual* der Reihe *z* entnommen. Das Ergebnis wird entweder in die Qualität *bqual* derselben Reihe, oder in einer neuen, temporären Reihe abgelegt.

Die Reihe, in die die synchronisierten Daten geschrieben werden, wird als Ergebnis zurückgeliefert.

szr enthält Synchropunkte, die als Textquanten abgelegt sind. Der Zeitpunkt eines Textquants gibt die (falsche) Zeit in der Reihe *z* an, die auf die (richtige) Zeit synchronisiert wird, der als Text im freien Format (z.B. `2.5.2001 7:30:35`) im Quant abgelegt ist.

Synchronoisiert wird immer zwischen zwei Synchropunkten. Wenn links oder rechts von *be* kein Synchropunkt vorhanden ist, wird an der entsprechenden Seite von *be* ein Synchropunkt angenommen, der auf sich selbst synchronisiert. Die Daten aus *z* werden zeitlich so gestaucht oder gestreckt und verschoben, dass alle Synchropunkte exakt passen.

Siehe auch KalibZR().

SynthetisiereEreignis

Syntax: SynthetisiereEreignis (ZR zr, QuantList quantile)
zr: aufzufüllende Reihe
quantile: das Ereignis beschreibende Information

Beispiel: `SynthetisiereEreignis (nzrauf, quantile)`

Beschreibung: Erzeugt ein synthetisches Ereignis in der Zeitreihe *zr* aus der Information, die in *quantile* gespeichert ist. Diese Funktion findet ihre Anwendung in der kontinuierlichen Auffüllung von Lücken.

quantile werden vorher mit der Funktion ExtrahiereEreignis() aus einer **anderen** Zeitreihe oder mittels Interpolation aus den Quantilen mehrerer anderer Zeitreihen extrahiert.

Siehe FindeEreignis() und NextLuecke().

System

Syntax: System(String s)
s: String, der den Systemaufruf enthält.

Beispiel: `System( "aqua_flux rotbach.szenario" )`

Beschreibung: Setzt den String *s* als Systemaufruf ab.

Siehe auch SystemIO(), ClientExec() und GetPID().

SystemIO

Syntax: SystemIO (String befehl, String eingabe) : String
befehl: String, der den Systemaufruf enthält.
eingabe: String, der an den Systemaufruf weitergegeben wird.

Beispiel: `SystemIO( "ps aux | grep azur" ,"")`

Beschreibung: Setzt den String *befehl* als Systemaufruf ab. Die Ausgaben dieses Befehls werden zurückgeliefert. Falls *eingabe* nicht leer ist, wird es als Eingabe an Systemaufruf übergeben.

Siehe auch System().

Tag

Syntax: Tag (Zeitpunkt zp) : Real
zp:

Beispiel:
```
t := Tag (tmpzp)
```

Beschreibung: Liefert den Tag (im Monat) eines Zeitpunkts.

Siehe auch Stunde().

Tagessummen

Syntax: Tagessummen (ZR z, Intervall i, Bool b [, R l]) : ZR
z:
i: Berechnungszeitraum
b: Temporärflag
l: optional: Lückenlimit in % (Voreinstellung 100)

Beispiel:
```
tw := Tagessummen (z, MAXFOCUS, FALSE)
```

Beschreibung: Erzeugt eine Intervall-Zeitreihe, deren Intervalle einen Tag breit sind. Der Y-Wert zu diesen Intervallen ist jeweils das bestimmte Integral über dem Tag. Die Aussage der Ergebniszeitreihe wird auf Sum gesetzt, die Herkunft wird auf A gesetzt, alle weiteren Attribute vererbt. Die Berechnung der Einheit findet automatisch statt.

Der optionale Parameter *l* legt fest, ab wieviel Prozent lückenhafter Daten eine Lücke erzeugt wird. Wird dieser Wert nicht angegeben, so wird erst eine Lücke für das Intervall erzeugt, wenn die Zeitreihe dort ausschließlich Lücke ist.

Tan

Syntax: Tan (Real r) : Real
r:

Beispiel:
```
t := Tan (2*Pi - winkel)
```

Beschreibung: Liefert den Tangens des Winkels *r* im Bogenmaß.

TCPClose

Syntax: TCPClose (String addr) : Bool
addr: Adresse:Port

Beispiel:
```
gut := TCPClose ("192.168.20.200:8051")
```

Beschreibung: Schließt den Socket zu *addr*. Falls kein Socket mit dieser Adresse verknüpft ist oder beim Schließen ein Fehler auftrat, wird False geliefert, sonst True.

Siehe auch TCPOpen(), TCPGet() und TCPPut().

TCPGet

Syntax: TCPGet (String addr, String ende—Real anz, Real timeout) : String
addr: Adresse:Port
ende: Endezeichen, bis zu dem gelesen wird
oder anz: Anzahl Zeichen, die gelesen werden sollen
timeout: Timeout in Sekunden, unendlich = 0

Beispiel:
```
gut := TCPGet ("192.168.20.200:8051", Char(3), 3)
```

Beschreibung: Liest Bytes von einem Socket, der vorher mit TCPOpen() geöffnet wurde. Der Socket wird mit *addr* adressiert.

Es können entweder alle Bytes bis zu dem Endezeichen *ende* oder *anz* Bytes gelesen werden. Falls die Gegenseite *timeout* Sekunden kein Zeichen schickt, wird das Lesen abgebrochen.

Falls keine Verbindung zustande kommt, wird die Zeichenfolge EOT NAK (Char(4) und Char(21)) gesendet.

Siehe auch TCPClose() und TCPPut().

TCPOpen

Syntax: TCPOpen (String addr) : Bool
addr: Adresse:Port

Beispiel:
```
gut := TCPOpen ("192.168.20.200:8051")
```

Beschreibung: Öffnet zu *addr* einen Socket. Auf diesen kann im Folgenden mit *addr* zugegriffen werden.

Falls keine Verbindung zustande kam, wird False geliefert, sonst True.

Siehe auch TCPClose(), TCPGet() und TCPPut().

TCPPut

Syntax: TCPPut (String addr, String daten, Real timeout) : Bool
addr: Adresse:Port
daten: zu sendende Daten
timeout: Timeout in Sekunden

Beispiel:
```
gut := TCPPut ("192.168.20.200:8051", daten, 2)
```

Beschreibung: Schickt *daten* an *addr*. Falls dabei ein Fehler auftritt, wird False geliefert, sonst True.

Siehe auch TCPOpen(), TCPClose() und TCPGet().

TempName

Syntax: TempName (String dir) : String
dir: Name eines Verzeichnisses

Beispiel:
```
name := TempName( "/tmp" )
```

Beschreibung: Liefert einen Namen, der für eine temporäre Datei benutzt werden kann. Jeder Aufruf von TempName() erzeugt einen anderen Namen.

dir legt das Verzeichnis fest, in dem die Datei erzeugt werden soll. Leerstring steht für das aktuelle Verzeichnis. Die Datei wird jedoch nicht von TempName() erzeugt.

Der zurückgegebene Name enthält *dir*, kann also direkt als Dateiname benutzt werden.

Siehe auch [**?**].

Terminwerte

Syntax: Terminwerte (ZR z, Intervall i, Distanz d, Bool b) : ZR
z:
i: Berechnungszeitraum
d: Abstand der Zeitpunkte, für die die Berechnung stattfindet
b: Temporärflag

Beispiel: `zrt := Terminwerte( z, focus, ~"1 Tag", FALSE)`

Beschreibung: Erzeugt eine Momentan-Zeitreihe mit Terminwerten.

Der erste Terminwert wird zu Beginn des Berechnungszeitraums erzeugt. Dann wird alle d ein Terminwert berechnet, bis der Berechnungszeitraum überschritten wird. Der letzte Wert liegt innerhalb oder genau auf der rechten Grenze von i.

Die Ergebniszeitreihe erbt die Attribute von z, mit Ausnahme der DefArt. XDistanz und XFaktor werden aus d gewonnen.

Siehe auch IntervallMittel().

TextCenterOnPage

Syntax: TextCenterOnPage(Page page, Real y, String text, Real style, [Real winkel])
page:
y: Zeile ($0 \leq y < Maxzeile$)
text:
style: siehe TextOnPage()
winkel: optional 0 bedeutet waagerecht

Beispiel:
```
TextCenterOnPage(page, 13, "Haupttabelle", BOLD+UNDERLINE)
```

Beschreibung: Setzt einen Text in der angegebenen Zeile zentriert auf die Seite. Siehe auch NewPage().

TextIQuant

Syntax: TextIQuant(Intervall xbereich, string text) : Quant
xbereich: Zeitintervall des Quants
text: Text-Wert

Beispiel:
```
q := TextIQuant( [von,bis], "Schmelze" )
```

Beschreibung: Erzeugt ein Text-IntervallQuant, das sich über das Intervall *xbereich* erstreckt. Der y-Wert wird auf *text* gesetzt.

Siehe auch IntervallQuant() und TextQuant().

TextOnPage

Syntax: TextOnPage(Page page, Real x, Real y, String text, Real style, [Real winkel —String ausricht])
page:
x: Spalte ($0 \leq x < Maxspalte$)
y: Zeile ($0 \leq y < Maxzeile$)
text:
style: Kombination folgender Konstanten : BOLD, UNDERLINE, ITALIC,
OUTLINE, VERYSMALL, SMALL, NEXTSMALL, HALFSMALL, SEMISMALL, NORMAL,
SEMIBIG, HALFBIG, NEXTBIG, BIG, VERYBIG, HUGE, VERYHUGE
winkel: optional 0 bedeutet waagerecht. oder:
ausricht: optional Textausrichtung (LINKS, RECHTS, ZENTRIERT)

Beispiel:
```
TextOnPage(page, 12,13, "Haupttabelle", BOLD+UNDERLINE)
```

Beschreibung: Setzt einen Text an der angegebenen Position auf die Seite. Die linke obere Ecke hat die Koordinate (0,0). Siehe auch NewPage(). *winkel* gibt den Winkel des Textes in Grad an. Wird dieser Parameter weggelassen, wird er auf 0 gesetzt.

Das Verhältnis der Textgrößen zu NORMAL ist von VERYSMALL bis VERYHUGE: 1/3, 1/2, 4/7, 2/3, 3/4, 1, 4/3, 3/2, 7/4, 2, 3, 5 und 10.

ausricht ist ein optionaler Parameter. Mit ihm kann die Ausrichtung des Textes angegeben werden. Als Voreinstellung wird der Text so ausgegeben, dass er links bündig an die Position x, y anschließt. Wenn *ausricht* RECHTS ist, wird der Text so ausgegeben, dass sein Ende rechts bündig an x, y anstößt. Mit ZENTRIERT wird der Text um x, y zentriert. *winkel* und *ausricht* können nicht gemeinsam angegeben werden.

Siehe auch DrawTextOnPage(), PlotTextOnPage(), SetPageGrauton() und TextWidth().

TextQuant

Syntax: TextQuant(XPunkt x, string text) : Quant
x: X-Punkt des Quants, Zeitpunkt oder Real
text: Text-Wert

Beispiel:
```
q := TextQuant(@"23.5.1949", "Er"offnung")
```

Beschreibung: Erzeugt ein Text-MomentanQuant zum X-Punkt x, dessen y-Wert auf *text* gesetzt wird. Der Parameter x kann ein Zeitpunkt oder ein Realpunkt sein. Der Typ muss dem Typ der Zeitreihe entsprechen, in die das Quant geschrieben wird.

Siehe auch MomentanQuant() und TextIQuant().

TextQuantenfolge

Syntax: TextQuantenfolge (ZR z, Intervall i, [Real qualy]) : QuantList
z:
i: Berechnungszeitraum
qualy: Qualität

Beispiel:
```
ql := TextQuantenfolge (zr1, bereich)
```

Beschreibung: Liefert die Folge der Text-Quanten, die in der Zeitreihe enthalten sind. Auf die Texte eines Quants kann mit der Funktion YText() zugegriffen werden.

TextWidth

Syntax: TextWidth (Page seite, String txt, Real stil) : Real
seite:
txt:
stil: siehe TextOnPage()

Beispiel:
```
breit := TextWidth (seite, "Printeneis", NORMAL)
```

Beschreibung: Berechnet die Breite, die der Text *txt* einnehmen würde, wenn er auf die Seite *seite* im Stil *stil* gezeichnet würde. Der Font (siehe SetPageFont()) wird berücksichtigt.

ThrowEvent

Syntax: ThrowEvent (String eventname)
eventname: Name des Ereignisses

Beispiel: `ThrowEvent ("@explsel")`

Beschreibung: Löst explizit die Aktion *eventname* aus. An das Auftreten der Aktion können Handles geknüpft werden. Diese werden aufgerufen.

Sie dazu AddHandle() und SetHandle().

Token

Syntax: Token(String s, Real num [, String sep]) : string
s: Ausgangsstring
num: Nummer des Tokens (beginnt bei 1)
sep: optional Separator

Beispiel: `wort := Token (zeile, 3)`

Beschreibung: Liefert das *num*te Wort eines String. Wörter (Tokens) sind durch beliebig viele Leerzeichen oder Tabs voneinander getrennt. Wörter, die Leerzeichen oder Tabs enthalten, müssen in Gänsefüßchen eingeschlossen sein. Wird *sep* angegeben, dann ist das Trennzeichen *sep* statt Leerzeichen. Wenn im Beispiel *zeile* den Wert `"Dies ist 3" dies auch 3` hat, dann erhält *wort* den Wert `auch`.

Ist *num* 0, so wird der gesamte String zurückgeliefert. Enthält die Zeile weniger als *num* Wörter, dann wird ein Leerstring geliefert.

Transform

Syntax: Transform(ZR z, ZR kurve, Intervall bereich, String param, Bool temp[,Real version]) : ZR
z: Ausgangsreihe
kurve: Transformationsvorschrift als Realreihe
bereich: Berechnungszeitraum
param: Parameter der Ergebnis-Reihe
temp: Temporärflag
version: optional Version der Ergebniszeitreihe

Beispiel:
```
bfzr := Transform(zr, bkurve, bereich, "Beckenf\ulle", FALSE)
```

Beschreibung: Setzt die Reihe z auf dem Bereich *bereich* mittels der Vorschrift *kurve* in eine neue Reihe um. *kurve* muss eine Realreihe sein (x-Bezug Real statt Zeit) mit realwertigen y-Werten. Der y-Wert jedes Stützpunktes von z wird als x-Wert in *kurve* aufgesucht, der y-Wert an diesem Punkt wird dann als y-Wert der Ergebnisreihe abgelegt. Der Parameter wird *param*, die Herkunft abgeleitet, alle weiteren Attribute mit Ausnahme von Einheit, Messgenau, FToleranz und NWGrenze (welche aus *kurve* entnommen werden) werden z entnommen.

Transpose

Syntax: Transpose(ZR z, String s)
z:
s: neue Einheit

Beispiel:
```
Transpose( niederzr, "l/(s*ha)" )
```

Beschreibung: Ändert die Einheit von z in s und transformiert die Y-Werte. Einheiten wie die oben benutzte `l/(s*ha)` werden automatisch ausgewertet. Die neue Einheit muss kompatibel mit der alten Einheit sein. Im obigen Beispiel könnte die Zeitreihe z.B. die Einheit `mm/h` haben, welche kompatibel zu `l/(s*ha)` (Regenspende) ist. Wichtig ist, dass die Groß- Kleinschreibung exakt eingehalten wird. Siehe auch die Angaben zu Größe im Anhang. μ wird als `\m` angegeben.

TransZR

Syntax: TransZR (ZR z,Intervall be,String o,Distanz xd,Real yd,Bool temp) : ZR
z: Ausgangszeitreihe
be: Berechnungszeitraum
o: neuer Ort der Zeitreihe
xdelta:
ydelta:
temp: temporär-Flag

Beispiel:
```
zr:=TransZR (zr,bereich, "Floyd", ~"12 Stunden",1.5, false)
```

Beschreibung: Erzeugt aus einer Zeitreihe eine neue, indem die erstere um die Zeitdistanz *xd* nach rechts (negative Distanz nach links) und den Y-Offset *yd* verschoben wird. Diese neue Zeitreihe erbt alle Attribute der alten und erhhält den neuen Ort() *o*.

Trend

Syntax: Trend (ZR z,Intervall be) : Real
z: Ausgangszeitreihe
be: Berechnungszeitraum

Beispiel:
```
tr:=Trend (zr,bereich)
```

Beschreibung: Unter der Annahme, dass *z* einen linearen Trend auf dem Bereich *be* besitzt, wird die Steigung der Trendgeraden berechnet. Diese wird in Y-Einheiten pro Jahr berechnet.

Die Steigung der Trendgeraden ist der Regressionskoeffizient von Y zu X.

Siehe auch STUDTest().

TrendBereinigung

Syntax: TrendBereinigung (ZR z,Intervall be, Real trend, Bool temp) : ZR
z: Ausgangsreihe
be: Berechnungszeitraum
trend: Steigung der Trendgeraden
temp: Temporärflag

Beispiel:
```
trendweg := TrendBereinigung (reihe, bereich, trendzahl, TRUE)
```

Beschreibung: Die Ausgangszeitreihe *z* wird auf dem Bereich *be* um einen linearen Trend bereinigt, der durch seine Steigung *trend* (in Y-Einheit pro Jahr) angegeben wird (siehe auch Trend()). Die Trendgerade schneidet in der Mitte des Bereichs *be* die (Summenlinie der) Reihe.

Das Ergebnis ist wiederum eine Reihe, deren Attribut Herkunft auf B gesetzt ist.

Trim

Syntax: Trim (String s) : String
s:

Beispiel:
```
t := Trim(s)
```

Beschreibung: Löscht alle führenden und abschließenden White-Space-Zeichen aus dem String *s*. White-Space-Zeichen sind z.B. Leerzeichen, Tabulatorzeichen und Returns.

TupAdd

Syntax: TupAdd (Tupel a, Tupel b)

Beispiel:
```
TupAdd(a, b)
```

Beschreibung: Addiert die einzelnen Zahlkomponenten von Tupel *a* und Tupel *b*. Die Summe wird in Tupel *a* gespeichert.

Die Struktur der beiden Tupel muss gleich sein. Ist a ungültig, werden alle Komonenten von b nach a kopiert. Ist b ungültig, bleibt Tupel a unverändert.

Siehe auch TupPrjct().

TupClearRecNum

Syntax: TupClearRecNum (Tupel tup)
tup: Tupel

Beispiel:
```
TupClearRecNum (tup)
```

Beschreibung: Löscht die Recordnummer des Tupels t, setzt diese also auf -1.

Siehe auch TupRecNum() und AppTupel().

TupCodesResolve

Syntax: TupCodesResolve (Tupel t) : Tupel
t: Tupel

Beispiel:
```
 tupmit := TupCodesResolve(tupohne)
```

Beschreibung: Erzeugt ein neues Tupel aus t. Alle Nicht-Code-Felder werden kopiert, alle Code-Felder werden entsprechend der hinterlegten ResolveInfo aufgelöst. Aufgelöste Code-Felder ändern ihre Struktur, der Typ ist immer Text, die Breite so, dass das Displayfeld hineinpasst.

Siehe auch ADBSetResolveInfo().

Tupel

Syntax: Tupel(String aufbau — Relation R) : Tupel
aufbau: Format des Tupels oder R: Relation, aus der der Aufbau generiert wird

Beispiel:
```
t := Tupel ("Name#10s,Betrag#7.2n")
```

Beschreibung: In der ersten Form (wie im Beispiel) wird ein neues Tupel mit der Struktur $aufbau$ erzeugt. Diese Struktur kann auch mittels Struktur() aus einer Relation gewonnen werden.

In der zweiten Form wird ein Tupel unmittelbar aus einer Relation, zu dieser passend, erzeugt. Das Tupel erbt in diesem Falle auch die Information über Schlüssel-Felder von R (siehe CreateIndex()).

Es besteht die Möglichkeit, ein ungültiges Tupel anzulegen. Dazu übergibt man einen Leerstring als $aufbau$.

Siehe auch NewMemRelation().

Die Syntax des Formates ist im Anhang beschrieben.

TupelList

Syntax: TupelList () : TupelList

Beispiel: `TL := TupelList ()`

Beschreibung: Erzeugt eine TupelListe. In eine TupelListe können beliebige Tupel (auch mit unterschiedlichem Format aufgenommen werden. Sie werden mit + oder += angehängt.

Beispiel:

`TL += tup`

TupelStr

Syntax: TupelStr (Tupel t, String trenner [,String feld1, String feldn]) : String
t: Tupel
trenner: Trennzeichen
feld1,feldn: optional: Einschränkung der Komponenten

Beispiel: `s := TupelStr (t, "|")`

Beschreibung: Liefert einen String zurück, der alle Komponenten hintereinander durch *trenner* getrennt in ihrem jeweiligen Format enthält. Bei Angabe von $feld1$ und $feldn$ werden nur die Komponenten von $feld1$ bis $feldn$ berücksichtigt, sonst alle.

Siehe auch TupToCSV() und Tupel().

TupEqual

Syntax: TupEqual (Tupel t1, Tupel t2) : Bool
t1: ein Tupel
t2: ein anderes Tupel

Beispiel:
```
IF (TupEqual (t, maskent))
```

Beschreibung: Prüft Feld für Feld, ob $t1$ und $t2$ gleich sind. Wildcard-Felder spielen dabei keine Sonderrolle, d.h. sie werden nur als gleich angesehen, wenn sie beide Wildcard sind.

Siehe auch Match().

TupIndex

Syntax: TupIndex (Tupel tup, String feld) : Real
tup: Tupel
feld: Feldname

Beispiel:
```
d := TupIndex (t, "ZRHide")
```

Beschreibung: Liefert die Rangfolgenummer des Feldes $feld$ in der Struktur des Tupels *tup*. Das erste Feld hat die Nummer 0. Ist $feld$ kein Feld des Tupels, wird -1 zurückgegeben.

Siehe auch GetZahl(), GetText() usw.

TupIsSelected

Syntax: TupIsSelected (Tupel t) : Bool
t: Tupel

Beispiel:
```
IF (TupIsSelected (tuppy))
```

Beschreibung: Liefert zurück, ob das Tupel t selektiert ist.

Siehe auch TupSelect().

TupJoin

Syntax: TupJoin (Tupel tup1, Tupel tup2) : Tupel
tup1: Haupttupel
tup2: Zusatztupel

Beispiel: `kombitup := TupJoin (kerntup, niedertup)`

Beschreibung: Erzeugt ein Tupel, das alle Felder aus *tup*1 und zusätzlich alle Felder aus *tup*2, die nicht in *tup*1 sind, enthält.

Siehe auch TupJoinTo() und TupPrjct().

TupJoinTo

Syntax: TupJoinTo (Tupel tup1, Tupel tup2)
tup1: Haupttupel
tup2: Zusatztupel

Beispiel: `TupJoinTo (kerntup, niedertup)`

Beschreibung: Erweitert das Tupel *tup*1 um die Komponenten aus *tup*2, die nicht bereits vorhanden sind.

Diese Prozedur ist schneller als TupJoin().

TupKey

Syntax: TupKey (Tupel t) : String
t: Tupel

Beispiel: `s := TupKey(tuppy)`

Beschreibung: Gibt den Inhalt des Keyfelds (der Keyfelder) von Tupel zurück. Auf der Relation, aus dem *tup* stammt, muss ein Index erzeugt worden sein (siehe CreateIndex()).

Siehe auch RelKey().

TupPrjct

Syntax: TupPrjct (Tupel quelle, Tupel ziel)
quelle: Quell-Tupel
ziel: Ziel-Tupel

Beispiel:
```
TupPrjct(masktup, reltup)
```

Beschreibung: Projiziert das Tupel *quelle* auf das Tupel *ziel*.

Alle Felder von *ziel*, die in *quelle* vorhanden sind, werden aus diesem kopiert. Alle übrigen Felder bleiben unverändert.

Siehe auch RelPrjct() und TupJoin().

TupQFAdd

Syntax: TupQFAdd (Tupel a, QuantList qf, String feld1, String feldn [, Real limit])
a: Tupel
qf: bestehende QuantList aus Intervallquanten
feld1: erstes zu bearbeitendes Feld
feldn: letztes zu bearbeitendes Feld
limit: optional: Obergrenze

Beispiel:
```
TupQFAdd(a, qf, "FeldJan", "FeldDez")
```

Beschreibung: Addiert die Komponenten $feld1$ bis $feldn$ von Tupel a auf die QuantList (Quantenfolge) qf. Der Inhalt von $feld1$ wird auf das erste Quant in qf addiert, das nächste Feld auf das zweite usw.

Optional kann mit *limit* eine Obergrenze angegeben werden. Enhält eine Komponente einen Wert, der größer als *limit* ist, so wird lediglich *limit* addiert und nicht der Wert.

Überschüssige Quanten bleiben unverändert. Überschüssige Felder bleiben unbearbeitet. Quanten, die eine Lücke enthalten, werden übersprungen.

qf darf nur IntervallQuanten enthalten (siehe IntervallQuant() und AppendQuant()).

Siehe auch TupAdd().

TupRecNum

Syntax: TupRecNum () : Real
t: Tupel

Beispiel: `i := TupRecNum ()`

Beschreibung: Liefert die Recordnummer des Tupels t. Diese wird gesetzt, wenn das Tupel aus einer dbf-Relation (siehe Relation()) gelesen wird.

Die Funktion kann beispielsweise für LockTupel() verwendet werden.

Wenn das Tupel nicht aus einer dbf-Relation stammt, oder ungültig ist, wird -1 zurückgeliefert.

Siehe auch TupClearRecNum(), Rewrite() und DBFilter().

TupRestruct

Syntax: TupRestruct (Tupel tup, String feldaufbau)
tup:
feldaufbau: bestehender Feldname mit neuem Aufbau

Beispiel: `TupRestruct (tup, "Nachname#30S")`

Beschreibung: Gibt einem Feld eines Tupels einen neue Struktur. Man kann damit beispielsweise erreichen, dass Felder verbreitert werden.

$feldaufbau$ muss einen Feldnamen enthalten, der bereits im Tupel vorhanden ist.

TupSelect

Syntax: TupSelect (Tupel t, Bool sel)
t: Tupel
sel: True=selektieren, False=deselektieren

Beispiel: `TupSelect (t, True)`

Beschreibung: Selektiert bzw. deselektiert ein Tupel. Mit TupIsSelected() kann man abfragen, ob ein Tupel selektiert ist.

Siehe auch SelectAll().

TupSetReadOnly

Syntax: TupSetReadOnly (Tupel tup, String feld, Bool an)
tup: Tupel
feld: Feldname
an: True=ReadOnly

Beispiel:
```
TupSetReadOnly (tup, "AEO", True)
```

Beschreibung: Setzt das Feld *feld* des Tupels *tup* auf ReadOnly (*an*=True) oder auf schreibbar (*an*=False). Der ReadOnly-Zustand bewirkt, dass die Zelle in einem DBGrid zum Editieren gesperrt wird. Darüberhinaus hat diese Prozedur keine Auswirkung.

Siehe NewDBGrid().

TupSetStyle

Syntax: TupSetStyle (Tupel tup, String feld, Real style)
tup: Tupel
feld: Feldname
style: Kombination (Summe) von NORMAL, BOLD, ITALIC und UNDERLINE

Beispiel:
```
TupSetStyle (tup, "AEO", BOLD)
```

Beschreibung: Steuert, wie das Feld *feld* des Tupels *tup* in einem DBGrid dargestellt werden soll. Es ist möglich, dass das Fenstersystem UNDERLINE oder eine Kombination der Stile nicht darstellen kann.

Siehe NewDBGrid() und TextOnPage()

TupToCSV

Syntax: TupToCSV (Tupel t, String trenner) : String
t: Tupel
trenner: Trennzeichen

Beispiel:
```
s := TupToCSV (t, "|")
```

Beschreibung: Liefert einen String zurück, der alle Komponenten hintereinander durch *trenner* getrennt in ihrem jeweiligen Format enthält. Im Unterschied zu TupelStr() werden Inhalte von Textfeldern jedoch in Gänsefüßchen eingeschlossen und nicht mit angehängten Leerzeichen ausgegeben.

Siehe auch TupelStr() und CSVToTup().

UniversQuery

Syntax: UniversQuery (String dsn, String url, String query) : String
dsn: Name der entfernten Datenbank
url: host:port
query: Anfrage-String

Beispiel:
```
xmldaten := UniversQuery ("nordwind test", "pferd:6928", sqlquery)
```

Beschreibung: Sendet eine Datenbank-Anfrage an einen UniverSQL-Server und liefert deren Ergebnis zurück.

UnlockTupel

Syntax: UnlockTupel (Relation R, Real num)
R: dbf-Relation
num: Nummer des Tupels in der Relation (0 ..)

Beispiel:
```
UnlockTupel( Stamm, 100)
```

Beschreibung: Das Lock des Tupels Nummer *num* der dbf-Relation *R* wird gelöst. Schreibzugriffe sind nun wieder möglich.

Siehe auch LockTupel().

UpCase

Syntax: UpCase (String s) : String
s:

Beispiel:
```
Print(UpCase( "Azur" ))
```

Beschreibung: Wandelt alle Kleinbuchstaben des Strings *s* in Großbuchstaben um.

UpdateBenutzer

Syntax: UpdateBenutzer (String loginname)
loginname: Loginname des neuen Benutzers

Beispiel:
```
UpdateBenutzer ("root")
```

Beschreibung: Setzt einen neuen aktiven Benutzer. Die Benutzerrelation muss das Feld LOGIN besitzen. Das Tupel in der Benutzerrelation, das im Feld LOGIN den Wert *loginname* hat, wird ausgewählt. Ist kein solches Tupel vorhanden, wird der Benutzer ungültig.

Siehe ADBInit() und Benutzer().

UpdateCanvas

Syntax: UpdateCanvas()

Beispiel:
```
UpdateCanvas()
```

Beschreibung: Lädt alle Zeitreihen aller Axboxen neu und zeichnet den gesamten Canvas neu. Diese Prozedur ist nützlich, wenn Zeitreihen, die zurzeit in einer Axbox dargestellt werden, verändert wurden.

Siehe RedrawCanvas().

URLParams

Syntax: URLParams (String url) : Array
url: komplette URL mit Parametern oder nur URL-Parameter

Beispiel:
```
A := URLParams (url))
```

Beschreibung: Erzeugt ein Array mit allen Parametern, die in URL enthalten sind. Sonderzeichen werden umgewandelt.

Siehe auch MakeURLParams().

Username

Syntax: Username() : String

Beispiel:
```
u := Username()
```

Beschreibung: Liefert den Namen des aktuellen Systembenutzers. Im AquaWeb-Betrieb ist dies der Name, der beim Anmelden benutzt wurde.

UTMToGrad

Syntax: UTMToGrad (String p) : GeoPoint
p: Punkt in UTM-Koordinaten

Beispiel:
```
p2 := UTMToGrad(p)
```

Beschreibung: Berechnet die geografischen Koordinaten (Länge, Breite) aus dem Punkt p, der in UTM-Koordinaten angegeben ist. p hat das Format Zone Band Quadrat x y, also z.B. 32 U ME 53400 24700.

Siehe auch GradToUTM().

Zur Transformation in andere Koordinatensysteme siehe auch LambertToGrad() und GKToGrad().

UVSRelation

Syntax: UVSRelation (String name, String dsn, String url) : Relation
name: Name der Relation
dsn: DSN-String der Datenbank
url: host:port

Beispiel:
```
R := UVSRelation ("stammdat", "suedwind test", "hai:6928")
```

Beschreibung: Öffnet die Relation mit Namen *name*. Ist keine Relation dieses Namens vorhanden oder der *name* ein Leerstring, dann wird eine ungültige Relation geliefert (siehe IsValid()).

dsn ist der DSN-String (Database-Source-Name) der Datenbank (database), in der die Relation (table) gespeichert ist. *url* enthält den Namen des Rechners, auf dem der UniverseSQL-Server läuft und, mit : getrennt, den Port, auf den der UniverseSQL-Server hört. Der Port kann auch weggelassen werden, dann wird standardmäßig 6928 verwendet.

Siehe auch NewUVSRelation().

Varianz

Syntax: Varianz (ZR zr, Intervall bereich) : Real
zr1:
bereich: Auswertungszeitraum

Beispiel:
```
var := Varianz(zr1, bereich)
```

Beschreibung: Berechnet die Varianz der Zeitreihe *zr* über *bereich*. Durch Angabe von *bereich* wird die Stichprobe festgelegt. Die Zeitreihe muss eine äquidistante Intervall-Zeitreihe sein (diskrete Zufallsvariable). Siehe auch Mittel(), Median(), Trend() und Covarianz().

Die Standardabweichung ist die Wurzel der Varianz.

Version

Syntax: Version (ZR z) : Real
z:

Beispiel: `ver := Version(abflusszr)`

Beschreibung: Liefert das Attribut Version der Zeitreihe. Versionen sollten mit Bedacht Anwendung finden. Meist unterscheiden sich zwei Zeitreihen in einem anderen Identifikationsattribut, z.B. sind Tagessummen aus Messern und berechnete Tagessummen aus Schreibern im Attribut Aussage unterschieden (O bzw. A).

Eine typische Anwendung für das Attribut Version sind Zeitreihen, die aus verschiedenen Simulationsläufen entstehen.

Verteilung

Syntax: Verteilung(ZR zr, Intervall zi, String typ, Bool log, Real maxtn, Real version, Bool tmp [, Bool korr]) : ZR
zr: partielle oder jährliche Serie
zi: Auswertungszeitraum
typ: Typ der Verteilungsfunktion (Gamma, Pearson, Gumbel oder Weibull)
log: `TRUE`=die Berechnung findet logarithmisch statt
maxtn: maximale Jährlichkeit
version: Attribut Version der Ergebnisreihe
tmp: Temporärflag korr: optional Korrektur. Voreinstellung: `TRUE`

Beispiel: `zrgamma := Verteilung(smzr, zi, "Gamma", FALSE, 100, 0 , FALSE)`

Beschreibung: Erzeugt die statistische Verteilungsfunktion des gewünschten Typs aus einer partiellen oder jährlichen Serie *zr*.

version legt das Attribut Version der Ergebnisreihe fest.

Die Stichprobe ergibt sich aus allen Werten in zr, die in zi liegen. Ist log `TRUE`, dann werden diese logarithmiert. Darauf werden die Parameter der entsprechenden Verteilungsfunktion berechnet. Für einen Bereich knapp über 0 bis $maxtn$ werden an ausreichend vielen Stützstellen die Funktionswerte zu den Jährlichkeiten berechnet (und ggf. entlogarithmiert). Diese Werte werden in die Ergebnisreihe eingetragen.

$korr$ legt fest, ob bei jährlichen Serien eine Korrektur bei der Berechnung der Jährlichkeit T_n stattfinden soll (siehe [8]).

Das Ergebnis ist eine kontinuierliche Realreihe, die alle Attribute von zr erbt, mit Ausnahme von: XEinheit (a) und Aussage (Ga, Pe, Gu bzw. We, im logarithmischen Fall lGa, lPe, lGu bzw. We).

Verteilungsparameter

Syntax: Verteilungsparameter(ZR serie, Intervall bereich, Bool ulog, Bool wlog, Bool temp [, Real gII, Real gIII]) : ZR
serie: partielle oder jaehrliche Serie
bereich: Auswertungszeitraum
ulog : FALSE = Ausgleich von u einfach, TRUE = doppelt logarithmisch
wlog : FALSE = Ausgleich von w einfach, TRUE = doppelt logarithmisch
temp : Temporärflag
gII : optional : Grenze zwischen Bereich I und II, default 9=3 h
gIII : optional : Grenze zwischen Bereich II und III, default 16=48 h

Beispiel:
```
vp := Verteilungsparameter (pserie, dekade, true, false, true)
```

Beschreibung: Berechnet die Parameter u und w der Verteilungsfunktionen aller Dauerstufen aus *serie*. *serie* ist eine partielle oder eine jährliche Serie, die z.B. mit den Funktionen PartielleSerie oder JaehrlicheSerie berechnet werden können. Berücksichtigt werden alle Werte, die im Auswertungszeitraum *bereich* liegen. Für partielle Serien ist zu beachten, dass der Auswertungszeitraum mit dem Auswertungszeitraum übereinstimmt, mit dem die partielle Serie berechnet worden ist. Beispielsweise ergeben sich für die partielle Serie andere Werte, wenn sie über 20 zusammenhängende Jahre berechnet wurde, als sich ergeben, wenn sie über zweimal 10 Jahre berechnet wurde. Für jährliche Serien gilt diese Einschränkung nicht. Falls im Attribut NWGrenze() ein Wert > 0 steht, wird die Anzahl der Jahre daraus gewonnen, statt aus der Breite des Auswertezeitraums.

Für partielle Serien wird die Exponentialverteilung $h_N(T_n) = u + w \cdot \ln T_n$ angesetzt, für jährliche Serien die Extremal-I-Verteilung (Gumbel-Verteilung) $h_N(T_n) = u + w(-\ln \ln \frac{T_n}{T_n - 1}$. [3]

Die Parameter u und w werden in den Qualitätsschichten 0 und 1 abgelegt. In einem zweiten Schritt werden sie nach DVWK-Regel 124/1985 [3] angepasst. Dabei wird ein einfach- oder ein doppelt-logarithmisches Verfahren verwendet (gesteuert durch die Parameter *ulog* und *wlog*). Die DVWK-Regel legt nahe, für u den doppelt- und für w den einfach- logarithmischen Ausgleich zu verwenden. Die angepassten Parameter u und w werden in den Qualitätsschichten 2 und 3 abgelegt.

Die Parameter sind jeweils relativ zum Startzeitpunkt von *bereich* mit der Distanz ihrer jeweiligen Dauerstufe abgelegt. Beispiel: wenn die Auswertung vom 1.11.1979 7:30 an erfolgte, findet sich der 5 min-Wert am 1.11.1979 7:35, der 18 h-Wert am 2.11.1979 1:30 usw.

Die Ergebniszeitreihe ist eine Momentan-Zeitreihe, deren Aussage auf pVP oder jVP ist, je nachdem, ob sie aus einer partiellen oder einer jährlichen Serie entstanden ist.

Die Grenzen der 3 Dauerstufen-Bereiche liegen als Voreinstellung (wie in [3] vorgeschlagen) auf 3 h und 48 h. Das heißt, alle Dauerstufen ≤ 3 h liegen in Bereich I, alle anderen ≤ 48 h in Bereich II und alle übrigen in Bereich III. Diese Grenzen können optional eingestellt werden. Man gibt sie als Rangnummer an, wobei die erste Dauerstufe (5 min) die Rangnummer 0 hat.

Die Übergabeparameter *ulog*, *wlog*, gII und $gIII$ werden in den Kommentar() der Ergebniszeitreihe geschrieben. Im obigen Beispiel würde der Kommentar `T F 9 16` enthalten.

Zur weiteren Verarbeitung dient die Funktion Regenhoehenlinie.

WebGet

Syntax: WebGet (String url [, String addheader]) : String
url: Adresse der zu holenden Daten
addheader: optional: weitere Headerzeilen (mit CR-LF getrennt)

Beispiel:
```
daten := WebGet ("tstp.aquaplan.de:8030?Cmd=Query&Ort=000*")
```

Beschreibung: Schickt eine URL an einen Server und empfängt das Ergebnis.

Falls der Server nicht erreichbar ist oder nicht antwortet, wird ein Leerstring zurück geliefert.

Siehe auch WebPut() und ImportQF().

WebPut

Syntax: WebPut (String url, String daten [, String addheader]) : String
url: Adresse der zu sendenden Daten
daten: Daten, die an die URL verschickt werden sollen
addheader: optional: weitere Headerzeilen (mit CR-LF getrennt)

Beispiel:
```
WebPut ("tstp.aquaplan.de:8030?Cmd=Put&ZRID=8543", daten)
```

Beschreibung: Schickt Daten an einen über *url* bestimmten Server. *url* enthält den Befehl und ggf. weitere Parameter. *daten* ist ein Datenblock, der mit http-POST verschickt wird.

Der Rückgabewert ist eine Fehlermeldung oder eine positive Bestätigung. Die Auswertung ist dem Aufrufer überlassen.

Siehe auch ExportQF() und WebGet().

WerteMenge

Syntax: WerteMenge (Relation rel, S feld) : Array
rel: Memory-Relation
feld: Name des Feldes

Beispiel:
```
substrat := WerteMenge (zeit_geber, "ORT")
```

Beschreibung: Liefert alle unterschiedlichen Werte des Feldes $feld$ in rel.

Siehe auch RelDateMatch().

WerteQF

Syntax: WerteQF(ZR z, Intervall i, [Real qualy [, String readmode]]) : QuantList
z:
i: Berechnungszeitraum
qualy: optional: Qualität
readmode: optional: Interpoliermodus

Beispiel:
```
wql := WerteQF(zr, bereich)
```

Beschreibung: Liest aus einer Zeitreihe eine Momentan-Quantenfolge, die deren Verlauf als Stützstellenliste widerspiegelt. Diese kann z.B. dazu benutzt werden, einen Wertepaar-Editor zu erstellen.

Unterstützt wird diese Funktion durch die Möglichkeit, auch in Intervall- oder kontinuierliche Zeitreihen Momentanquantenfolgen zu schreiben. Es ist sichergestellt, dass eine mit WertQF gewonnene Folge mittels WriteQuantenfolge wieder zurückgeschrieben werden kann, ohne dass sich die Zeitreihe verändert.

WildcardMatch

Syntax: WildcardMatch(String s, String muster) : Bool
s: ein String
muster: Musterstring

Beispiel:
```
IF (WildcardMatch(name, "A?x*")
```

Beschreibung: Prüft, ob der String *name* auf das Muster *muster* passt. *muster* kann die Jokerzeichen `*` und `?` enthalten.

Wenn im Beispiel *name* = `"Aix la Chapelle"` ist, dann wird TRUE zurückgeliefert.

Anderes Beispiel für Muster:

Man möchte ein Muster angeben, dass auf alle Strings passt, die mit `Schacht` anfangen, dann eine zweistellige Zahl enthalten, danach eine mit Leerzeichen getrennte weitere Beschreibung. Das Muster dazu ist `Schacht?? *`

Siehe auch RegExpMatch().

Der hier beschriebene Wildcardmechanismus wird verwendet in Stammdaten(), ADBQuery(), DBFilter(), StammdatenByFeld() und Match().

WnachQ

Syntax: WnachQ (ZR z, Intervall i, String s, Bool b [,Real v]) : ZR
z: Wasserstands-Zeitreihe
i: Überarbeitungsbereich
s: Verfahren (STAU, ETA, ETA+STAU)
b: [?]
v: optional Attribut Version

Beispiel:
```
z := WnachQ (z, i, "ETA+STAU", b)
```

Beschreibung: **Diese Funktion ist veraltet und sollte nicht mehr benutzt werden. Stattdessen sollte die Funktion QvonW() benutzt werden.**

Diese Funktion setzt Wasserstand (W) in Abfluss (Q) um. Wie üblich, findet die Berechnung nicht auf der gesamten Zeitreihe sondern nur auf dem Intervall i statt. Soll die Berechnung für die gesamte Wasserstandszeitreihe erfolgen, so gibt man für i die Konstante MAXFOCUS an.

v ist ein optionaler Parameter, der als Voreinstellung auf 0 gesetzt ist. Er steuert das Attribut Version der Ergebniszeitreihe. Dies kann nützlich sein für Testversionen der Abflusszeitreihe oder um mehrere Abflüsse vorzuhalten, die nach verschiedenen Verfahren berechnet wurden.

Der Abfluss kann nach dem Stauwerte- ($\Delta - W$-) Verfahren, dem Eta- ($\Delta - Q$-) Verfahren oder einer Kombination beider Verfahren berechnet werden (siehe [1]).

Beide Verfahren greifen automatisch auf weitere Reihen zu, die im gleichen Datenpool (Verzeichnis) vorhanden sein müssen. Im einzelnen sind dies:

Stauwerte-Verfahren

- Die Stauwerte-Zeitreihe. Diese muss kontinuierlich sein und den gleichen Ort wie die Wasserstands-Zeitreihe und den Parameter Stauwert besitzen.
- Die Abflusskurven-Zeitreihe. Dies ist eine Intervallzeitreihe, deren Intervalle die zeitliche Gültigkeit der entsprechenden Abflusskurve darstellen. Der Ort muss mit dem der Wasserstands-Zeitreihe identisch sein, der Parameter muss Abflusskurven lauten. Jede Abflusskurve selbst ist der Y-Wert zu diesem Intervall. Die Namen der Wertedateien sind als Text direkt in der Reihe abgelegt. Die Abflusskurven-Zeitreihe kann über das AQZ-Ascii-Format in den Datenpool eingespeist werden (siehe [5]).
- Alle Abflusskurven. Dies sind jeweils Realreihen, also Reihen, deren Definitionsbereich Realzahlen sind. Dieser Definitionsbereich ist der Wasserstand, der Wertebereich ist der zugehörige Abfluss. Der Ort muss derselbe sein, wie der Ort der Wasserstands-Zeitreihe. Auch diese Reihen können über das AQZ-Ascii-Format erzeugt werden.

Etawerte-Verfahren

- Die Etawerte-Zeitreihe. Diese muss kontinuierlich sein und den gleichen Ort, wie die Wasserstands-Zeitreihe und den Parameter Etawert besitzen.
- Die Abflusskurven-Zeitreihe. Dies sind Intervallzeitreihen, deren Intervalle die zeitliche Gültigkeit der entsprechenden Hüllkurve darstellen. Beide Abflusskurven-Zeitreihen müssen denselben Ort wie die Wasserstands-Zeitreihe haben. Die Parameter sind OHuellen für die oberen Hüllkurven und UHuellen für unteren Hüllkurven. Jede Hüllkurve selbst ist der Y-Wert zu diesem Intervall. Die Namen der Wertedateien sind als Text direkt in der Reihe abgelegt. Die Abflusskurven-Zeitreihe kann über das AQZ-Ascii-Format in den Datenpool eingespeist werden (siehe [5]).
- Alle oberen und unteren Hüllkurven. Dies sind jeweils Realreihen, also Reihen, deren Definitionsbereich Realzahlen sind. Dieser Definitionsbereich ist der Wasserstand, der Wertebereich ist der zugehörige Abfluss, jeweils als untere oder obere Hülle. *Zu beachten ist, dass die untere Hüllkurve größere Abflüsse als die obere Hüllkurve besitzt. Die Hüllkurven sind (vgl. [1]) nicht die mathematischen Hüllkurven der Punktewolke der Funktion $Q = f(W)$, sondern die grafisch ermittelten „Hüllkurven“ der Punktewolken, deren Definitionsbereich senkrecht und deren Wertebereich waagerecht aufgetragen wird.* Auch diese Reihen können über das AQZ-Ascii-Format erzeugt werden.

Erzeugt wird eine Abfluss-Zeitreihe (Parameter Abfluss, gleicher Ort). Der Wechsel von einer Abflusskurve zur nächsten wird, entsprechend den Angaben in der Abflusskurven-Zeitreihe, automatisch ausgeführt.

Zu beachten ist, dass die Einheit des Definitionsbereichs der Abflusskurven identisch ist mit der Einheit der Wasserstandszeitreihe. Die Einheit des Wertebereichs der Abflusskurven muss kompatibel zu m^3/s sein. Die Abflusskurven können auch jeweils unterschiedliche Einheit besitzen. Die Werte werden dann automatisch umgerechnet.

Falls die Resultats-Zeitreihe neu erzeugt wird, erhält sie ebenfalls die Einheit m^3/s. Ist die Resultats-Zeitreihe schon vorhanden und enthält sie eine andere Einheit, die aber kompatibel zu m^3/s ist, (z.B. l/s), dann werden die Werte entsprechend skaliert. Liegt eine unkompatible, also falsche, Einheit vor, so erscheint eine Fehlermeldung und die Werte werden negativ eingetragen.

WnachQHZB

Syntax: WnachQHZB (ZR wzr, Intervall i, Bool b [,Real v]) : ZR
wzr: Wasserstands-Zeitreihe
i: Überarbeitungsbereich
b: Temporär-Flag
v: optional Attribut Version

Beispiel:
```
z := WnachQHZB (wreihe, i, false)
```

Beschreibung: Diese Funktion setzt, den österreichischen Regeln folgend, Wasserstand (W) in Abfluss (Q) um.

Grundlage sind die Wasserstands-Zeitreihe und die Pegelschlüsselkurven mit deren Gültigkeiten.

wzr ist die kontinuierliche Wasserstands-Zeitreihe. Die Pegelschlüsselkurven-Zeitreihe ergibt sich automatisch durch den Ort von *wzr* und dem Parameter Pegelschlüssel.

Ist *b* TRUE, dann wird die Abfluss-Zeitreihe nur temporär erzeugt, d.h. sie wird nach Beendigung des Azurprogramms (oder des Aquagramms) gelöscht.

v ist ein optionaler Parameter mit der Vorbelegung 0 und gibt das Attribut Version vor. Dies ist sinnvoll, wenn Abflüsse mit verschiedenen Methoden berechnet und gegenübergestellt werden sollen.

Die Berechnung findet auf dem Bereich i statt. Ist die Abfluss-Zeitreihe schon vorhanden, dann wird der neue Bereich eingefügt bzw. ersetzt. Die Ausgabe-Zeitreihe erbt die Attribute von wzr mit Ausnahme von

- Parameter: Abfluss
- FToleranz: 0.01
- Einheit: m^3/s
- Herkunft: A (abgeleitet)

Ist die Abfluss-Zeitreihe schon vorhanden, dann werden die bestehenden Attribute übernommen. Falls die Einheit nicht m^3/s ist, aber dazu kompatibel (z.B. l/s), dann werden die Werte entsprechend skaliert.

Jede Pegelschlüsselkurve besteht aus bis zu drei Abschnitten I, II und III. Der Funktionsverlauf dieser Abschnitte folgt jeweils der Formel:

$$Q = a(W - c)^b$$

Die Parametersätze a, b, c inkl. der Bereichsgrenzen s, sowie der Angabe, ab welchem Wasserstand das Gewässer trockenfällt (WfuerQ0), werden in der Relation schluess.dbf vorgehalten.

Die Angaben, welcher Schlüssel wann gültig ist, werden in einer Zeitreihe mit Parameter Pegelschlüssel vorgehalten.

Alle Wasserstände, die kleiner als $WfuerQ0$ sind, erzeugen einen Abfluss von 0. Dies gilt auch für negative Wasserstände.

Die Berechnung des Abflusses Q aus W erfolgt quantweise für jedes Quant der Wasserstands-Zeitreihe. In einem ersten Schritt werden die Quantgrenzen umgesetzt.

Da der Verlauf der Schlüsselkurve in diesem Bereich nicht linear sein muss, werden in die Abfluss-Zeitreihe in das entsprechende Quant zusätzliche Punkte aufgenommen. Dies geschieht an den Punkten der größten Abweichung des theoretischen und des angenäherten Verlaufs.

Diese größte Abweichung lässt sich durch Ableitung der Differenzfunktion bestimmen. Der
(fälschlicherweise) lineare Verlauf zwischen den Endepunkten des Q-Quants sei durch die Funktion

$$g(W) = dW + e$$

gegeben. Die Differenzfunktion lautet demnach

$$(f - g)(W) = a(W - c)^b - dW + e$$

Abgeleitet ergibt sich

$$(f - g)'(W) = ab(W - c)^{b-1}) - d$$

Der Nulldurchlauf ergibt sich zu

$$W = e^{\frac{ln\frac{ab}{d}}{b-1}} + c$$

Der gesuchte Zeitpunkt ist sofort interpolierbar, da der Verlauf von W auf dem Bereich linear ist. An diesem Zeitpunkt wird ein neuer Stützpunkt eingefügt.

So werden sukzessive Stützpunkte eingefügt, bis die Fehlertoleranz unterschritten wird.

Sollte ein Bereichswechsel der Schlüsselkurve innerhalb eines Quants vorliegen, die Funktion $f(W)$ sich also ändern, dann wird das Quant exakt an den Bereichsgrenzen geteilt und die Berechnung für beide Teile separat durchgeführt.

Vorgehen bei Änderung der Schlüsselkurve

Es sind zwei Fälle von Änderung der Schlüsselkurve (über die Zeit) zu unterscheiden: ein gleitender Wechsel von einem Schlüssel zum nächsten (linearer, nicht konstanter Verlauf der Pegelschlüsselgültigkeits-Zeitreihe) und ein abruptes Ändern der Schlüsselkurve (konstanter Verlauf).

Im ersten Fall, wird der Wasserstand durch beide an den Randpunkten vorgegebene Schlüssel umgesetzt. Zwischen den so berechneten Werten wird dann linear entsprechend dem Zeitpunkt interpoliert.

Im zweiten Fall liegt zwar nur ein Schlüssel vor, es muss also nicht interpoliert werden, der Sprung an den Gültigkeitsgrenzen bedarf jedoch der gesonderten Behandlung. Dazu wird genau am Sprung der Wasserstand einmal mit der bis dahin gültigen und einmal mit der ab dort gültigen Schlüsselkurve berechnet und dann gemittelt.

WriteDBF

Syntax: WriteDBF(Relation R, String name)
R: (Mem-)Relation
name: Dateiname der dbf-Datei (ohne .dbf)

Beispiel:
```
WriteDBF (FiltR, "filtered")
```

Beschreibung: Schreibt eine Relation als DBF-Datei. Dies ist nur sinnvoll für Relationen, die nur im Speicher vorliegen, also MemRelationen.

Siehe dazu auch NewMemRelation().

WriteFile

Syntax: WriteFile (String filename, String inhalt)
filename: name der Datei
inhalt: Inhalt der Datei

Beispiel:
```
WriteFile ("daten.gz", data)
```

Beschreibung: Schreibt *inhalt* in die Datei *filename*. *inhalt* darf auch binäre Daten enthalten.

Siehe auch ReadFile() und PrintFile().

WriteKarte

Syntax: WriteKarte (Karte K)
K:

Beispiel: `WriteKarte (Kart)`

Beschreibung: Erzeugt oder überschreibt eine Szenerie-Datei für die Karte. Deren Name ist der Name der Karte (siehe Name()) mit der Endung .szn. Layer, die noch nicht als Datei existieren (siehe WriteLayer()), werden als Datei gespeichert. Layer, deren Inhalte sich gegenüber der Layer-Datei geändert haben, werden ebenfalls neu geschrieben.

Siehe auch ReadKarte() und Karte().

Eine ausführliche Beschreibung der Geo-Formate findet sich im Handbuch [**?**].

WriteLayer

Syntax: WriteLayer (Layer L, String format)
L:
format: Geometrie-Format

Beispiel: `WriteLayer (Isos, "AI")`

Beschreibung: Schreibt den Layer L im Format $format$ in eine Datei. Format kann sein AI, PIA, BIN oder DXF.

Der Name der Datei ergibt sich aus dem Namen des Layers (siehe Name()) und einer Endung, die dem Format entspricht: .ai für format AI, ,obz und .seg für format PIA, .agb für Format BIN und .dxf für Format DXF.

Siehe auch ReadLayer(), Layer() und Karte()

Eine ausführliche Beschreibung der Geo-Formate findet sich im Handbuch [**?**].

WritePalmDB

Syntax: WritePalmDB (Relation R, String datei)
R: Relation
datei: Dateiname der PDB-Datei

Beispiel: `WritePalmDB (FiltR, "tour1.pdb")`

Beschreibung: Schreibt eine Relation als PDB-Datei (PalmOS Data Base).

Die Relationen dürfen keine Memo-Felder enthalten.

Siehe auch ReadPalmDB().

WriteQuantenfolge

Syntax: WriteQuantenfolge (ZR reihe, QuantList qf, [Real qualy [, Bool entkol])
reihe: eine Zeitreihe
qf: eine Quantenfolge
qualy: optional: gewünschte Qualität
entkol: optional: entkolinearisieren

Beispiel: `WriteQuantenfolge(reihe, qf, 1)`

Beschreibung: Schreibt die Quantenfolge qf in die Qualität *qualy* der Zeitreihe.

Ist *qualy* nicht angegeben oder -1, dann wird in die höchste Qualität geschrieben.

Normalerweise werden die Daten beim Schreiben in die Zeitreihe nicht entkolinearisiert. Ist dies gewüscht, gibt man als vierten Parameter True an, Beispiel: `WriteQuantenfolge(reihe, qf, -1, True)`.

Siehe auch WriteTextQuantenfolge() und StoreQF().

WriteTextQuantenfolge

Syntax: WriteTextQuantenfolge (ZR reihe, QuantList qf, [Real qualy])
reihe: eine Zeitreihe
qf: eine Quantenfolge
qualy: optional: gewünschte Qualität

Beispiel:
```
WriteTextQuantenfolge(reihe, qf, 1)
```

Beschreibung: Schreibt die Textquantenfolge *qf* in die Qualität *qualy* der Zeitreihe. Ist *qualy* nicht angegeben, dann wird in die höchste Qualität geschrieben.

Ist *qualy* nicht angegeben oder -1, dann wird in die höchste Qualität geschrieben.

Siehe auch WriteQuantenfolge() und StoreTextQF().

WWJ

Syntax: WWJ (Real jahr) : Intervall
jahr: Wasserwirtschaftsjahr

Beispiel:
```
z := WWJ( 1980 )
```

Beschreibung: Liefert das Zeitintervall, das das Wasserwirtschaftsjahr *jahr* angibt. Diese Funktion ist sehr nützlich, um den Berechnungszeitraum einer Berechnung zu setzen.

XBereich

Syntax: XBereich (Quant q) : Intervall
q:

Beispiel:
```
bereich := XBereich(q)
```

Beschreibung: Liefert das Zeitintervall, über den sich das Quant *q* erstreckt. Siehe auch YLinks() und YRechts().

XDistanz

Syntax: XDistanz (ZR z) : String
z:

Beispiel: `a := XDistanz (z)`

Beschreibung: Liefert das Attribut XDistanz der Intervall-Zeitreihe. Dieses Attribut erlangt seinen Wert erst im Zusammenhang mit XFaktor().

Siehe auch Zeitschritt().

XEinheit

Syntax: XEinheit(ZR zr) : String
zr: Eine Reihe

Beispiel: `einh := XEinheit( kurvezr )`

Beschreibung: Liefert das Attribut XEinheit der Reihe. Dieses Attribut ist nur relevant bei Reihen, deren X-Bezug Real statt Zeitpunkt ist, Siehe auch SetXEinheit().

XFaktor

Syntax: XFaktor (ZR z) : Real
z:

Beispiel: `a := XFaktor (z)`

Beschreibung: Liefert das Attribut XFaktor der Intervall-Zeitreihe. Dieses Attribut erlangt seinen Wert erst im Zusammenhang mit XDistanz().

Siehe auch Zeitschritt().

XKoo

Syntax: XKoo (GeoPoint g) : Real
g:

Beispiel:
```
x := XKoo( Koord(zr1) )
```

Beschreibung: Liefert die X-Koordinate des GeoPoints g.

XLSToDB

Syntax: XLSToDB (String xlsfile [, R x0 [, R y0 [, R xn [, R yn]]]]) : Datenbank
xlsfile: Name der xls-Datei
x0: optional: Startspalte
y0: optional: Startzeile
xn: optional: Endspalte
yn: optional: Endzeile

Beispiel:
```
rel := XLSToDB ("rrb.xls", 12)
```

Beschreibung: Liest eine xls-Datei ein und erzeugt für jedes Arbeitsblatt eine Relation. Das Ergebnis ist eine Datenbank, die alle Relationen enthält.

Mit $x0$, $y0$, xn, yn kann ein Ausschnitt aus dem Spaltenraster definiert werden. Die erste Spalte und die erste Zeile haben die Nummer 0.

YKonstInAxBox

Syntax: YKonstInAxBox(AxBox ax, Real lage, String text, String farbe, Real pos)
ax:
lage: Höhe in der AxBox
text:
farbe: siehe ZRInAxBox
pos: Position des Textes

Beispiel:
```
YKonstInAxBox( ax, 12.5, "Warngrenze",  "Blau", 1)
```

Beschreibung: Definiert eine neue Y-Konstante für die AxBox *ax*. Ein Strich über der gesamten Breite der AxBox wird in der Höhe *lage* in der Farbe *farbe* gezeichnet. *text* ist die Beschriftung der Konstanten, die an 6 Stellen erfolgen kann, die mit *pos* gesetzt werden: 1=oben rechts, 2=oben links 3=unten links, 4=unten rechts, 5=oben mittig, 6=unten mittig

YKoo

Syntax: YKoo (GeoPoint g) : Real
g:

Beispiel:
```
y := YKoo( Koord(zr1) )
```

Beschreibung: Liefert die Y-Koordinate des GeoPoints *g*.

YLinks

Syntax: YLinks (Quant q) : Real
q:

Beispiel:
```
y := YLinks(q)
```

Beschreibung: Liefert den linken Y-Wert des Quants *q*. Siehe auch YRechts() und XBereich().

YRechts

Syntax: YRechts (Quant q) : Real
q:

Beispiel: `y := YRechts(q)`

Beschreibung: Liefert den rechten Y-Wert des Quants q. Siehe auch YLinks() und XBereich().

YText

Syntax: YText(Quant q) : String
q:

Beispiel: `s := YText( q1 )`

Beschreibung: Liefert den Text eines Text-Quantes. Falls das Quant kein Text-Quant ist, wird der (rechte) Realwert als String zurückgeliefert.

YTextWert

Syntax: YTextWert (ZR z, XPunkt xp) : String
z:
xp: Zeitpunkt oder Real

Beispiel: `s := YTextWert(niederzr, zeit)`

Beschreibung: Liefert den Text-Wert der Reihe z zum Zeitpunkt xp. Falls zum Punkt xp kein Text existiert, dann wird ein Leerstring zurückgegeben.

YTyp

Syntax: YTyp(ZR zr) : String
zr: Eine Reihe

Beispiel: `s := YTyp( qzr )`

Beschreibung: Liefert das Attribut YTyp der Reihe. YTyp ist F, wenn die Reihe Reihen als Y-Werte enthält, sonst leer. Es ist nicht möglich, dieses Attribut zu setzen.

YWert

Syntax: YWert (ZR z, XPunkt xp) : Real
z:
xp: Zeitpunkt oder Real

Beispiel: `y := YWert(niederzr, zeit)`

Beschreibung: Liefert den Y-Wert der Reihe z zum Punkt xp. Dieser Wert ist stützpunktunabhängig; bei kontinuierlichen Zeitreihen wird demnach interpoliert.

Zeilen

Syntax: Zeilen (AxBox ax) : Real
ax:

Beispiel: `z := Zeilen (axoben)`

Beschreibung: Die Anzahl der Zeilen der Legende einer AxBox. Siehe Spalten().

Zeitschritt

Syntax: Zeitschritt (ZR z) : Distanz
z:

Beispiel: `d := Zeitschritt (zrmittel)`

Beschreibung: Der Zeitschritt einer Intervall-Zeitreihe als Kombination der [?] XDistanz und XFaktor.

Siehe auch XFaktor() und XDistanz().

ZIBox

Syntax: ZIBox (String von, String bis, Bool zeit) : Intervall
von: Vorbelegung der linken Seite des Intervalls
bis: Vorbelegung der rechten Seite des Intervalls
zeit: True= Zeitpunkte, False=Realwerte

Beispiel:
```
focus := ZIBox (oldvon, oldbis, True)
```

Beschreibung: Erzeugt ein neues Fenster, welches eine Zeile mit zwei Eingabefeldern enthält, in die ein Zeitintervall oder ein Realintervall eingegeben werden kann. Zusätzlich wird ein OK- und ein Abbruch-Button erzeugt.

Das AGWindow, aus dem diese Box gestartet wurde, ist solange inaktiv, bis der Benutzer zwei gültige Werte eingegeben hat und den OK-Button oder Return in einem Feld drückt. Wenn der Benutzer den Abbruch-Button betätigt, wird ein ungültiges Intervall zurückgegeben.

Siehe auch OkBox(), InputBox(), ElementBox(), SelectBox() und MultiBox().

ZIStr

Syntax: ZIStr(Intervall i, String f) : String
i: ein Intervall
f: das Format entsprechend zpmode() oder RealFormat()

Beispiel:
```
s := ZIStr( focus, "#d.#h.#Y" )
```

Beschreibung: Wandelt das Intervall i gemäß dem Format f in einen String um. Das Format des Parameters f richtet sich nach dem Typ des Intervalls, das entweder, wie im Beispiel, ein Zeitintervall oder ein Realintervall sein kann.

Siehe auch ZPStr() und GStr().

ZKoo

Syntax: ZKoo (GeoPoint g) : Real
g:

Beispiel:
```
wert := ZKoo(punkt)
```

Beschreibung: Liefert die Z-Koordinate des GeoPoints g.

ZPAbrunden

Syntax: ZPAbrunden (Zeitpunkt zp, Distanz d) : Zeitpunkt
zp:
d:

Beispiel: `zpneu := ZPAbrunden(von, ~"1 Monat")`

Beschreibung: Rundet den Zeitpunkt *zp* auf das nächst kleinere ganze Vielfache der Distanz *d*. Wäre im Beispiel *von* = 10.12.1990 7:30, so ergäbe die Funktion den 1.12.1990 00:00.

ZPMode

Syntax: ZPMode(String s)
s: Ausgabemodus

Beispiel: `ZPMode( "\#d.\#Z.\#y" )`

Beschreibung: Setzt den String *s* als Modus, mit dem Zeitpunkte ausgegeben werden sollen. Siehe dazu die Angaben im Anhang.

ZPQuant

Syntax: ZPQuant(ZR reihe, Real qual, Zeitpunkt zp) : Quant
reihe: eine Zeitreihe
qual: gewünschte Qualität
zp: maßgebender Zeitpunkt

Beispiel: `lq := ZPQuant(zr2, 1, @"1.11.94")`

Beschreibung: Sucht von *zp* aus nach links und rechts den nächsten Nicht-Lücke-Wert. Diese beiden Wertepaare, zwischen denen also eine Lücke liegen kann, werden als Linienquant zurückgeliefert. Eine eventuelle Lücke kann man mit YWert() feststellen. Falls links oder rechts kein Wert mehr vorhanden ist, dann wird MinusInfty bzw. PlusInfty und Lücke geliefert.

Liegt der angegebene Zeitpunkt auf dem eines in der ZR abgelegten Wertes, dann enthält der gelieferte Linienquant dieses und das nachfolgende Wertepaar.

ZPQuant ist für Momentanzeitreihen nicht definiert, da diese keinen dichten (zusammenhängenden) X-Bereich besitzen. Das Ergebnis bei Momentanzeitreihen ist ein ungültiges Quant.

Wenn *qual* nicht bekannt ist, übergibt man -1.

Siehe auch YLinks(), YRechts() und XBereich().

ZPStr

Syntax: ZPStr(Zeitpunkt z, String f) : String
z: ein Zeitpunkt
f: das Format entsprechend zpmode()

Beispiel:
```
s := ZPStr(derzp, "#d.#h.#Y")
```

Beschreibung: Wandelt den Zeitpunkt z gemäß dem Format f in einen String um.

Siehe auch ZIStr(), RStr() und GStr().

ZRAnyTup

Syntax: ZRAnyTup (ZR z) : Tupel
z:

Beispiel:
```
tup := ZRAnyTup (zr1)
```

Beschreibung: Liefert das mit ZRSetAnyTup() an die Zeitreihe geknüpfte Tupel. Wurde keine Tupel angeknüpft, wird ein ungültiges Tupel geliefert.

Siehe auch ZRAttr().

ZRAttr

Syntax: ZRAttr (ZR z) : String
z:

Beispiel: `s := ZRAttr(zr1)`

Beschreibung: Jede Zeitreihe hat neben den Attributen, die in einem Tupel gehalten und permanent gespeichert werden, ein frei benutzbares Attribut vom Typ String. Mit ZRAttr wird es abgefragt. Diese Attribut wird nicht permanent gespeichert, sondern ist nur zur Laufzeit verfügbar.

Siehe auch ZRSetAttr(), ZRInfo() und ZRAnyTup().

ZRBearbStand

Syntax: ZRBearbStand (ZR reihe, Intervall bereich) : Real
reihe: zu behandelnde Reihe
bereich: Bereich, für den der Zustand abgefragt wird

Beispiel: `stand := ZRBearbStand (niederzr, bereich)`

Beschreibung: Fragt den Bearbeitungszustand der Reihe *reihe* für das Intervall *bereich* ab. Falls auf *bereich* verschiedene Bearbeitungszustände eingetragen sind, wird der niedrigste geliefert.

Siehe auch BearbStaende() und SetZRBearbStand().

ZRFolge

Syntax: ZRFolge (RL idrel, RL formrel, Tupel attrs) : ZR
idrel: ID-Zeitreihen-Zuordnung
formrel: Folgenvorschrift
attrs: die gewünschten Attribute der ZRFolge

Beispiel: `zrf := ZRFolge (idrel, vrel, meineattrs)`

Beschreibung: Verkettet Zeitreihen zeitlich zu einer Zeitreihe. Jedes Stück erlaubt eine Vielzahl von Umrechnungsfunktionen.

Das Ergebnis ist eine Zeitreihe *zr*, die keine Werte, sondern eine Vorschrift speichert, auf welchem Bereich auf welche Ausgangszeitreihe zugegriffen werden soll. So ist sichergestellt, dass nachträgliche Änderungen in den Ausgangszeitreihen sich sofort in *zr* niederschlagen.

In *idrel* sind die Ausgangszeitreihen angegeben. Jeder Zeitreihe ist eine Kennung zugeordnet, unter der sie in den Formeln auftaucht. Die Struktur von *idrel* muss lauten `Id#4S,Parameter#20S,Ort#20S,DefArt#1S,`
`Aussage#3S,XDistanz#1S,XFaktor#5N,Herkunft#1S,Reihenart#1S,Version#4N,Su`

Die Struktur von *formrel* muss lauten: `VonD#D,VonZ#T,Formel#250S`.

Bei der Angabe der Kennungen (in der *idrel* und in den Formeln) muss die Groß-Kleinschreibung beachtet werden.

Die Ergebniszeitreihe hat immer die Herkunft F und die Reihenart Z.

Aufbau einer Formel

Eine Formel kann aus beliebig vielen durch + oder − verbundenen Termen bestehen. Jeder Term besteht aus einer Zeitreihenkennung oder einer Zeitreihenauswertung mit optionalem Faktor oder einer Konstanten.

Die ZR-Terme können durch Anfügen von ¿¿*distanz* oder ¡¡*distanz* zeitlich verschoben werden. *distanz* kann z.B. sein: 1h, 90min, 2d.

Beispiele:

`zr1*10 + 1490` oder `IMit(zr2,T) + zr1`

Mögliche Operationen

- IMit (zr, breite [,maxlueckproz]) breite: beliebige Distanz z.B. T oder M. Berechnet Intervallmittel. Mit maxlueckproz kann man optional festlegen, dass Lücken-Intervalle erzeugt werden, wenn zuviele Lücken in der Ausgangszeitreihe vorhanden sind.
- IMax, wie IMit, berechnet Intervallmaxima
- IMin, wie IMit, berechnet Intervallminima
- ISum, wie IMit, berechnet Intervallsummen
- MMax (zr, breite), berechnet Maxima als Momentanwerte (mit Zeitpunkt)
- MMin, wie MMax, berechnet Minima
- GMit (zr, breite), berechnet gleitende Mittelwerte
- GMin, wie GMIt, brechnet gleitende Minima
- GMax, wie GMIt, brechnet gleitende Maxima
- Q (zr) oder QvonW(zr,quelle). Berechnet den Abfluss aus der Wasserstandszeitreihe. Siehe QvonW().
- QvonV (vzr, wzr, FvonW [, kvonW oder k]). Berechnet den Abfluss aus der Fließgeschwindigkeit (v) und dem Wasserstand (w). FvonW ist die Kurve, die die durchströmte Fläche über den Wasserstand berechnet. Wenn v nicht die mittlere Fließgeschwindigkeit ist, sondern nur eine Annäherung daran (z.B. eine Ultraschallmessung), kann mit kvonW eine zweite Kurve angegeben werden, die die Beziehung zwischen dem gemessenen v und dem mittleren v angibt, oder ein Faktor k, der nicht von W abhängt.
- O2Sat (o2zr, tmpzr). Berechnet die Sauerstoffsättigung aus der Sauerstoffkonzentration und der Wassertemperatur. Für die Berechnung wird die Höhe der Messung benötigt, die sich im Attribut Hoehe() der O_2-Zeitreihe finden muss.
- I2K (zr), macht aus Intervall-ZR kontinuierliche. Dadurch wird meist eine Datengenauigkeit vorgespiegelt, die nicht vorhanden ist.
- SCHWELLE (zr, ¡wert) oder (zr, ¿wert), erzeugt wird eine Reihe, die für die Bereiche, die kleiner (oder größer) als *wert* sind, die Daten von *zr* enthählt, sonst Lücke.
- MÜ0 (zr) berechnet die Pegelnullpunkt-ZR zu dem Ort von *zr*. Dazu wird auf die Stammdaten-Relation zeit_pegellatte zugegriffen
- VH (zr) berechnet die Verdunstung nach Haude. *zr* ist die Temperatur-ZR. Es wird auch auf die Luftfeuchte-ZR mit gleichem Ort zugegriffen (wenn diese nicht da ist, auf die Dampfdruck-ZR). Die beteiligten ZR können beliebiger DefArt sein, zugegriffen wird immer auf das Maximum zwischen 12 und 15 Uhr pro Tag. Das Ergebnis ist eine Tagesmittel-ZR.
- Prio (zr1,zr2,...) erzeugt eine Zeitreihe, die die Daten von $zr1$ erhält, oder, wenn $zr1$ eine Lücke aufweist, die Daten von $zr2$, usw.
- Func (zr1,kurve) setzt die Werte in $zr1$ mittels der Funktion um, die durch *kurve* definiert ist. *kurve* muss eine Realreihe sein.

ZRFormat

Syntax: ZRFormat (String datei) : String
datei: Datei mit Zeitreihe im Fremdformat

Beispiel:
```
form := ZRFormat("muehlen.82")
```

Beschreibung: Stellt fest, welches Zeitreihen-Austauschformat *datei* besitzt. Falls das Format nicht erkennbar ist, wird UnknowFormat zurückgeliefert. Dies ist auch der Rückgabewert, wenn die Datei nicht vorhanden ist.

Siehe auch Import() und ImportListe().

ZRInAxBox

Syntax: ZRInAxBox (ZR z, AxBox ax, String s [,String farbe [,Bool mittext [,Real qual]])
z:
ax: AxBox, in die die Zeitreihe gezeichnet wird
s: Legendentext
farbe: Die gewünschte Farbe der Ganglinie
mittext: Sollen die Texte der Reihe gezeichnet werden
qual: Qualität der Zeitreihe, Voreinstellung: höchste Qualität

Beispiel:
```
ZRInAxBox( zr2, GrossBox, Ort(zr2),"Rot",false)
```

Beschreibung: Zeichnet *z* in die AxBox *ax* und schreibt *s* in die Legende. farbe gibt die Farbe an, in der die Zeitreihe gezeichnet werden soll. Wird dieser Parameter weggelassen, dann ergibt sich die Farbe aus der Position, die diese Zeitreihe in der AxBox hat, die erste Zeitreihe wird in Schwarz, die zweite in Rot usw. gezeichnet.

Standardmäßig wird die Legende jedoch **nicht gezeichnet**. Dies geschieht nur, wenn ihre Position mit SetAxLegPos() festgelegt wird.

Farben sind: Weiß, Schwarz, Rot, Blau, Gruen, Gelb, Violet, Aqua, Orange, Grau, Hellgrau, Braun, Dunkelgrün, Dunkelgrau, Hellrot, Dunkelrot und Blassgrün. Es können auch die Strings der Zahlen 0 - 17 verwendet werden.

mittext gibt an, ob die Texte der Reihe mit gezeichnet werden sollen. Standardmäßig sind diese ausgeschaltet.

Ist der Parameter *qual* angegeben, wird die Qualität *qual* der Reihe gezeichnet, sonst die höchste Qualität.

Wenn die Zeitreihe sich schon in der AxBox befindet, dann wird sie ersetzt.

Die Legendeneinträge können auch explizit gesetzt werden. Siehe dazu SetLegende().

ZRInfo

Syntax: ZRInfo (ZR z) : String
z:

Beispiel:
```
s := ZRInfo(zr1)
```

Beschreibung: Jede Zeitreihe führt einen frei setzbaren, permanenten Infotext mit sich. Mit ZRInfo kann dieser abgefragt werden.

Siehe auch ZRSetInfo() und ZRAttr().

ZRLInfo

Syntax: ZRLInfo (ZRList zrl) : String
zrl:

Beispiel:
```
s := ZRLInfo(zrl)
```

Beschreibung: Jede Zeitreihenliste hat ein frei setzbares Attribut Info. Dieses kann mit ZRLInfo abgefragt werden.

Siehe auch ZRLSetInfo().

ZRList

Syntax: ZRList() : ZRList

Beispiel:
```
zrl := ZRList()
```

Beschreibung: Liefert eine leere ZRList (Reihen-Liste), deren Focus auf MaxFocus gesetzt ist.

ZRLSetInfo

Syntax: ZRLSetInfo (ZRList zrl, String info)
zrl:
info: neue Info

Beispiel:
```
ZRLSetInfo(zrl, "Alle Zeitreihen sind defekt")
```

Beschreibung: Jede Zeitreihenliste hat ein frei setzbares Attribut Info. Dieses kann mit ZRLSetInfo gesetzt werden.

Siehe auch ZRLInfo().

ZRLZip

Syntax: ZRLZip(ZRList zrl)

Beispiel:
```
ZRLZip(zrl)
```

Beschreibung: Bringt die Zeitreihenliste *zrl* in eine kompaktere Form, die zwar etwas langsamer zu verarbeiten ist, dafür aber deutlich weniger Speicherplatz verbraucht.

Zeitreihenlisten, die mit Query(), OrtQuery() oder ParamQuery() gewonnen werden, besitzen automatisch kompakte Form.

ZRMD5Sum

Syntax: ZRMD5Sum(Tupel rt) : String
rt : ein ReiheTupel

Beispiel:
```
md5 := ZRMD5Sum (zr.Attribute())
```

Beschreibung: Berechnet die MD5-Summe der Identifikationsattribute eines Reihetupels (siehe MD5Sum()). Diese kann als eindeutiger Schlüssel für die Zeitreihe benutzt werden.

Siehe auch ZRStr().

ZRModifyInfo

Syntax: ZRModifyInfo (ZR z) : Tupel
z: Zeitreihe

Beispiel:
```
tup := ZRModifyInfo (z)
```

Beschreibung: Liefert ein Tupel, das die Information über die letzte Änderung an der Zeitreihe enthält. Es hat den Aufbau `Datum#D,Zeit#T,User#8S,Host#8S,Von#14S,Bis#14S`. `Datum` und `Zeit` enthalten den Zeitpunkt der Änderung, `User` den Benutzer, der die Änderung vornahm, `Host` den Rechner, auf dem die Änderung stattfand und `Von` und `Bis` das Intervall, das verändert wurde. Die letzten beiden Felder sind Strings, damit darin sowohl Zeitpunkte als auch Realzahlen gespeichert sein können. Zum Zugriff auf Felder des Tupels siehe z.B. GetText().

Siehe auch Lebenslauf().

ZRNr

Syntax: ZRNr (ZRList zrl, Real nr) : ZR
zrl: eine Zeitreihenliste
nr: Nummer der ZR in der Liste

Beispiel:
```
zrn := ZRNr(zrsel, 3)
```

Beschreibung: Liefert die *nr*.te Zeitreihe einer Zeitreihenliste. Die erste ZR hat die Nummer 1, die letzte die Nummer AnzZR(). Ist *nr* außerhalb dieses Bereichs leer, so wird eine ungültige Zeitreihe zurückgegeben.

Siehe auch FirstZR().

ZRSchreibbar

Syntax: ZRSchreibbar (ZR zr) : Bool
zr:

Beispiel:
```
IF (ZRSchreibbar(zr))
```

Beschreibung: Ermittelt, ob die Zeitreihe *zr* schreibbar ist. Diese Eigenschaft wird beim Öffnen der Zeitreihe ermittelt und bleibt dann unverändert.

Siehe auch RelSchreibbar(), ExistsFile(), FileSchreibbar() und FileLesbar().

ZRSetAnyTup

Syntax: ZRSetAnyTup (ZR z, Tupel tup)
z:
tup: beliebiges Tupel

Beispiel:
```
ZRSetAnyTup (zr1, tup)
```

Beschreibung: Knüpft ein beliebiges Tupel *tup* an die Zeitreihe z. Eine denkbare Anwendung ist das Speichern von Darstellungsattributen von Zeitreihen in Achsenkreuzen. Das Tupel wird nicht permanent gespeichert, d.h. die Verknüpfung geht verloren, wenn das Azurprogramm beendet wird (z seine Gültigkeit verliert).

Siehe auch ZRAnyTup().

ZRSetAttr

Syntax: ZRSetAttr (ZR z, String s)
z:
s: neues Attribut

Beispiel:
```
ZRSetAttr(zr1, "Rot")
```

Beschreibung: Setzt das, frei vergebbare und nicht permanent gespeicherte, Attribut der Zeitreihe z.

Siehe auch ZRAttr(), ZRSetInfo() und ZRSetAnyTup().

ZRSetBurst

Syntax: ZRSetBurst(ZR z, Bool onoff)
z:
onoff: TRUE=Burst ein, FALSE=Burst aus

Beispiel:
```
ZRSetBurst(zr1, TRUE)
```

Beschreibung: Schaltet den Burstmodus der Zeitreihe z ein oder aus. Nach dem Öffnen ist jede Zeitreihe nicht im Burstmodus.

Der Burstmodus kann nur benutzt werden, wenn nur lesend auf die Zeitreihe zugegriffen wird. Alle Daten der Zeitreihe werden im Burstmodus in den Speicher geladen, die Zeitreihe wird also von der ZR-Datenbank entkoppelt. Das ist beim Multi-User-Zugriff zu bedenken. Diese Prozedur sollte also nur benutzt werden, wenn sichergestellt ist, dass kein anderer Benutzer die Datei gleichzeitig schreibt.

ZRSetInfo

Syntax: ZRSetInfo (ZR z, String s)
z:
s: neue permanente Info

Beispiel:
```
ZRSetInfo(zr1, "Ist am 8.1. von Meier gesetzt worden")
```

Beschreibung: Jede Zeitreihe führt einen frei setzbaren, permanenten Infotext mit sich. Mit ZRSetInfo kann dieser gesetzt werden.

Siehe auch ZRInfo() und ZRSetAttr().

ZRStr

Syntax: ZRStr(Tupel zrtup) : String
z:

Beispiel:
```
s := ZRStr(Attribute(zr1))
```

Beschreibung: Liefert zum Attribute-Tupel einer Reihe einen lesbaren, einzeiligen String.

Wenn der ADB-Manager initialisiert wurde (siehe ADBInit()), wird der Ort der Zeitreihe ersetzt durch den Namen, der dem Ort in den Stammdaten zugeordnet ist (siehe ADBName()).

ZRToCSV

Syntax: ZRToCSV(ZR zr, ZI focus, S file, S info1, S info2, B app)
zr: zu exportierende Zeitreihe
focus: Ausgabeintervall
file: Name der Ausgabedatei
info1: z.B. Stationsname
info2: z.B. Gewässername
app: True=Datei erweitern, False=Datei neu erstellen

Beispiel:
```
ZRToCSV (zr1, bereich, path, edvnr, gwnam, True)
```

Beschreibung: Exportiert die Zeitreihe zr auf dem Bereich $focus$ in die CSV-Datei $file$. Die Datei kann mittels *app* entweder erweitert oder neu erstellt werden.

Es wird ein Dateikopf erzeugt, der in der ersten Zeile den Ort der Zeitreihe, $info1$, $info2$ und $focus$ enthält. In der zweiten Zeile wird der Parameter und die Einheit ausgegeben. Die dritte Zeile enthält, wenn vorhanden, die Aussage und die Distanz der Zeitreihe, sonst wird eine Leerzeile erzeugt.

Pro Wert in $focus$ wird daran anschließend eine Zeile mit $info1$, Zeitpunkt und Wert ausgegeben. Die Felder sind mit Semikolon getrennt. Das Format der Werte ist 8.3f. Die Zeitpunkte werden mit Uhrzeit ausgegeben, wenn es sich um eine kontinuierliche Zeitreihe oder eine Intervall-ZR mit Zeitschritt < 1 Tag handelt.

Für Intervall-ZR werden pro Zeile die linke und die rechte Grenze des Intervalls ausgegeben, mit einem - getrennt. Für Lücken wird der Wert Lücke ausgegeben.

ZRToRel

Syntax: ZRToRel(ZR z, Intervall i, Real qualy, Real maxtxt, Bool mitzeit) : Relation
z:
i: Ausgabeintervall
qualy: Qualität, die exportiert werden soll
maxtxt: Breite des Textfeldes (0=keins)
mitzeit: Soll ein Feld für Stunde-Minute-Sekunde erzeugt werden?

Beispiel:
```
memrel := ZRToRel(zr1, bereich, MAXQUAL, 20, TRUE)
```

Beschreibung: Exportiert die Zeitreihe z auf dem Bereich i in der Qualität *qualy* in eine Memrelation (siehe NewMemRelation()).

Die Funktion arbeitet analog der Prozedur ExportDBF().

Kapitel 6

Besonderheiten

6.1 Sonderzeichen

Azur verarbeitet den ISO-LATIN1-Zeichensatz. Dieser Zeichensatz ist der Standard auf allen X-Windows-Systemen, sowie unter MS-Windows. Des Weiteren können alle PostScript-Drucker (Level 2) diesen Zeichensatz verarbeiten. Die unteren 128 Zeichen entsprechen dem ASCIIZeichensatz, ASCII ist also eine Untermenge. Auf den Stellen 128 - 255 finden sich viele nützliche Zeichen, wie Umlaute oder Sonderzeichen. Falls die Eingabe dieser Zeichen über die Tastatur nicht möglich ist, können Azur-Escape-Sequenzen benutzt werden, die im folgenden aufgeführt sind.

Escape-Sequenz	Ergebnis
`\t`	TAB
`\n`	NEWLINE
`\r`	CARRIAGE RETURN
`\f`	FORMFEED
`\A`	Ä
`\O`	Ö
`\U`	Ü
`\a`	ä
`\o`	ö
`\u`	ü
`\s`	ß
`\q`	“ („Gänsefüßchen“)
`\m`	μ
`\0`	^{o} , Gradzeichen
`\1`	^{1} , hochgestellte 1
`\2`	^{2} , hochgestellte 2
`\3`	^{3} , hochgestellte 3
`\\`	\
`\p`	%
`\C`	Copyright-Zeichen
`\E`	Euro-Währungs-Zeichen
`\/`	Char(130)
`|`	Char(131)
`\x<hex1><hex2>`	Hexzahl, Beispiel `\x3C`

Es ist nicht möglich, ein % unmittelbar einzugeben, es muss die Escape-Sequenz `\p` benutzt werden.

Eine weitere Besonderheit ist die Escape-Sequenz `\N`. Sie erzeugt ebenso wie `\n` ein NEWLINE, jedoch erst, wenn der String in einem Aquagramm dargestellt wird. Dies ist notwendig, weil `\n` direkt ein NEWLINE erzeugt, was die Kommunikation zwischen Azur und Aquagramm stört.

6.2 Einschränkungen bei boolschen Vergleichen

Wie aus der formalen Grammatik (2) zu entnehmen, ist die Benutzung von boolschen Vergleichen leicht eingeschränkt. Auf der linken Seite kann lediglich eine Variable, oder

ein Boolscher Ausdruck stehen. Vergleiche der Form

```
IF ((a+2) < 10)
```

sind daher nicht möglich.

Offensichtlich ist diese Einschränkung jedoch nicht sehr erheblich.

6.3 Auswertungsreihenfolge in Ausdrücken

Die Auswertungsreihenfolge von Ausdrücken folgt den bekannten Regeln. Ein multiplikativer Operator bindet stärker als ein additiver. Der Ausdruck

```
10 - 2 * 4
```

wird ausgewertet wie

```
10 - (2 * 4)
```

Azur definiert jedoch auch einen Potenzoperator `^`. Dieser bindet **nicht** stärker als `*` oder `/`, sondern wird von links nach rechts ausgewertet. Der Ausdruck

```
10 - 2 * 4 ^ 3
```

wird ausgewertet wie

```
a := 10 - (2 * 4) ^ 3
```

Kapitel 7

Formale Syntaxbeschreibung

7.1 Die formale Grammatik von AZUR

Die folgenden Regeln bilden formal die Grammatik, die die Sprache AZUR beschreibt. Bis zur Regel `const ->` finden sich solche, die kein Terminalsymbol produzieren, demnach auch kein Eingabesymbol verbrauchen, ab `ident ->` sind dann die Terminalproduktionen angegegben. Man beachte, dass die Grammatik nur den syntaktischen Aufbau der Sprache beschreibt. Weitergehende Definitionen zum Erstellen **semantisch** korrekter Programme sind im Kapitel *Semantik* aufgeführt.

```
azurprogram    -> azurfunc | azurfunc ; azurprogramm
azurfunc       -> ident parameters returntype; cmdlist end |
                  extern ident parameters returntype; |
                  uses ident
parameters     -> () | ( decllist ) | (ellipsis)
decllist       -> decl | decl , decllist | decl , ellipsis
decl           -> ident ident | ident ident = expression
returntype     -> : ident | e
cmdlist        -> command; | command ; cmdlist
command        -> forloop | whileloop | repeatloop | ifstmt |
                  assignment | procedurecall | returnstmt
forloop        -> forall ident in ident[(expr)]; cmdlist endfor |
                  forall ident in ident[(expr)] while ( condition );
                  cmdlist endfor
whileloop      -> while ( condition ); cmdlist endwhile
```

```
repeatloop     -> repeat; cmdlist until ( condition )
ifstmt         -> if ( condition ); cmdlist elsebranch
elsebranch     -> endif | else; cmdlist endif | elseif (condition );
                  cmdlist elsebranch
assignment     -> ident := expression | arrayelement := expression |
                  increment | decrement | plusequal | minusequal
procedurecall  -> ident paramlist | ident . ident paramlist
paramlist      -> () | ( exprlist )
returnstmt     -> return expression
exprlist       -> expression | expression , exprlist
condition      -> monomlist | monomlist or condition
monomlist      -> monom | monom and monomlist
monom          -> ident | ident cmpop expression | functioncall
                  not monom | (condition)
cmpop          -> = | # | < | > | <= | >= | ~=
intervall      -> [expression , expression]
zp             -> @ factor
distanz        -> ~ factor
geopoint       -> { expression , expression } |
                  { expression , expression, expression}
expression     -> term | term addop expression
term           -> factor | factor mulop term
factor         -> functioncall | const | ident | ( expression ) |
                  - expression | NOT expression | intervall |
                  zp | distanz | geopoint | arrayelement
addop          -> + | -
mulop          -> * | / | ^
functioncall   -> ident paramlist | ident . ident paramlist
const          -> number | stringconst | ident
ident          -> _|$|a-z|A-Z [_|a-z|A-Z|0-9 [...] ]
number         -> NUMBER
stringconst    -> STRINGCONST
bracketopen    -> (
bracketclose   -> )
braceopen      -> [
braceclose     -> ]
scopeopen      -> {
scopeclose     -> }
```

```
Assignment     -> :=
equal          -> =
smaller        -> <
greater        -> >
smallerequal   -> <=
greaterequal   -> >=
notequal       -> #
semicolon      -> ;
period         -> .
comma          -> ,
colon          -> :
alpha          -> @
tilde          -> ~
ellipsis       -> ...
plus           -> +
minus          -> -
mult           -> *
div            -> /
tothepower     -> ^
and            -> AND
or             -> OR
not            -> NOT
if             -> IF
else           -> ELSE
endif          -> ENDIF
forall         -> FORALL
in             -> IN
endfor         -> ENDFOR
while          -> WHILE
endwhile       -> ENDWHILE
repeat         -> REPEAT
until          -> UNTIL
return         -> RETURN
extern         -> EXTERN
end            -> END
arrayelement   -> ident [ ident ]
uses           -> USES
increment      -> ident ++
```

```
decrement       -> ident --
plusequal       -> ident += expression
minusequal      -> ident -= expression
Increment       -> ++
Decrement       -> --
Plusequal       -> +=
Minusequal      -> -=
elseif          -> ELSEIF
caseequal       -> ~=
```

Wie an der Regel `factor ->` zu erkennen, handelt es sich nicht um eine LR(1)-Grammatik. Es ist sicher möglich, diese Grammatik in eine LR(1) umzuwandeln. Darauf wurde aber bewusst verzichtet, da es die Lesbarkeit erheblich beeinträchtigt hätte.

Kapitel 8

Beispiele

8.1 Tagesmittel bilden

Gegeben sei der Parameter Wasserstand am Pegel kie0011 als kontinuierliche Zeitreihe. Aus dieser Zeitreihe möchten wir Tagesmittelwerte bilden. Die Resultatzeitreihe wird eine **Intervallzeitreihe** sein. Den Bereich, auf dem diese Mittelwertbildung ausgeführt wird, wollen wir als Parameter übergeben.

Schrittweise sei nun die Erstellung des dazu notwendigen Azurprogramms erklärt. Jedes Azurprogramm besteht mindestens aus der Funktion AZUR. Diese besteht aus einem Funktionskopf mit Parameterdeklaration:

```
AZUR (Zeitpunkt von, Zeitpunkt bis)
```

In diesem Falle werden auf der Kommandozeile (oder auch aus einem Aquagramm heraus) die Parameter `von` und `bis` übergeben, welche beide vom Typ `Zeitpunkt` sind.

Dann folgt der Funktionskörper. Zuerst wird die Zeitreihe zum Pegel (Ort) kie0011 mit dem Parameter Wasserstand der Variablen `zr1` zugewiesen, die dadurch automatisch den Typ `ZR` erhält.

```
   zr1 := GetZR ("kie0011", "Wasserstand", "", "K");
```

`GetZR` ist eine eingebaute Funktion, die 4 Argumente hat und eine Variable vom Typ `ZR` zurückliefert. Das erste Argument ist der Ort, dann folgt der Parameter der Zeitreihe, das dritte Argument spezifiziert die Aussage der Zeitreihe, ist es wie hier leer, wird die erste passende Zeitreihe geöffnet. Wir nehmen hier an, es gibt nur eine. Das letzte Argument gibt die Definitionsart der Zeitreihe an, `K` steht für kontinuierlich, `I` für Intervall und `M` für Momentan.

Der zweite und letzte Schritt ist das Bilden von Tagesmittelwerten. Dazu gibt es die eingebaute Funktion `IntervallMittel`. Mit ihr kann für Intervalle beliebiger Breite der Mittelwert berechnet werden. Das Resultat ist immer eine Intervallzeitreihe, die sich nur in der Definitionsart und der Herkunft von der Originalzeitreihe unterscheidet. Der Bereich, auf dem diese Auswertung stattfindet, wird angegeben. Soll dies der gesamte Bereich sein, für den Daten vorliegen, so ist dies `MAXFOCUS`. Hier geben wir ihn mittels `von` und `bis` vor:

```
   bereich := [von,bis];
```

`bereich` ist ein `Intervall`. Die eckigen Klammern stellen einen Operator dar, der zwei Zeitpunkte zu einem Intervall verknüpft. Dieses Intervall ist ein Argument für die folgende Funktion:

```
   zr2 := IntervallMittel (zr1, bereich, ~"1 Tag", FALSE);
```

`zr2` ist die Ergebniszeitreihe, sie wird hier nicht weiter verwendet. `~"1 Tag"` ist eine `Distanz`. Sie wird durch das Anwenden des unären Operators `~` auf einen String erzeugt. Der String enthält eine Zahl und die Angabe des Zeitschrittes in Klartext. Herauszustellen sind dabei die Zeitschritte `Monat` und `Jahr`, die mit einem aufwendigen Algorithmus mit Zeitpunkten verknüpft werden. Das letzte Argument gibt an, ob die entstehende Zeitreihe temporär oder dauerhaft sein soll. Temporäre Zeitreihen haben vor allem als Zwischenergebnis oder im Zusammenhang mit einem Aquagramm eine Bedeutung.

Ein

```
END
```

schließt die Funktion `AZUR` und damit das Azurprogramm ab. Insgesamt ergibt sich also folgendes Programm:

```
AZUR (Zeitpunkt von, Zeitpunkt bis)
   zr1 := GetZR ("kie0011", "Wasserstand", "", "K");
   bereich := [von,bis];
   zr2 := IntervallMittel (zr1, bereich, ~"1 Tag", FALSE);
END
```

8.2 Monatsmittel in Tabellenform ausgeben

Siehe dazu auch die Dokumentation zu den Reports.

Kapitel 9

Anhang

Literaturverzeichnis

[1] LAWA,BMV, 1991, Pegelvorschrift Stammtext, Verlag Paul Parey, Hamburg-Berlin

[2] DVWK, Regeln 120/1983, Niedrigwasseranalyse, Verlag Paul Parey, Hamburg-Berlin

[3] DVWK, Regeln 124/1985, Starkregenauswertung, Verlag Paul Parey, Hamburg-Berlin

[4] aqua_plan, 1992, Zeitreihen und ihre Benutzung. Analyse und Design des Datenmodells. Strategie-Papier

[5] aqua_plan, 1993, Der aqua_plan Zeitreihen Manager. Strategie-Papier.

[6] Sachs, L., 1992, Angewandte Statistik, 7. Auflage Springer-Verlag, Berlin

[7] DVWK, 1982, Weiterbildendes Studium Hydrologie - Wasserwirtschaft, Kapitel 6 Ermittlung von Hochwasser (DVWK)

[8] DVWK, 1979, Regeln zur Wasserwirtschaft, Empfehlung zur Berechnung der Hochwasserwahrscheinlichkeit. Verlag Paul Parey, Hamburg

[9] DVWK, 1999, Merkblätter zur Wasserwirtschaft, Statistische Analyse von Hochwasserabflüssen. Wirtschaft- und Verlagsgesellschaft Gas und Wasser, Bonn

[10] Hammer, N., 1993, Eine optimierte Starkniederschlagsauswertung, Teil III: OWUNDA, Optimierungsverfahren zur Erstellung von Regenhöhenlinien und Regenspendenlinien. ZAMG/HZB, Wien

[11] Engeln-Müllges, Reuter, 1996, Numerik-Algorithmen, 8. Auflage, VDI Verlag, Düsseldorf