infoGrips GmbH Informationssysteme Engineering & Consulting
Technoparkstrasse 1
8005 Zürich
Tel.: 044 / 350 10 10
Fax.: 044 / 350 10 19

iG/Script Benutzer- und Referenzhandbuch

Zusammenfassung

Diese Dokumentation beschreibt die Skriptsprache iG/Script und die in iG/Script verfügbaren ICS (infoGrips Conversion System) Klassen.

Inhaltsverzeichnis

1. Einleitung

iG/Script ist eine interpretierte Programmiersprache für die Steuerung der ICS-Schnittstellenprodukte (ICS = infoGrips Conversion System) **INTERLIS Tools** bzw. **INTERLIS Tools Professional**. Mit iG/Script können ICS-Schnittstellen sehr flexibel für einen bestimmten Anwendungszweck konfiguriert werden. Falls Sie eigene Konfigurationen erstellen möchten, oder die mit den ICS-Schnittstellen mitgelieferten Basiskonfigurationen um spezielle Funktionen ergänzen möchten, finden Sie in dieser Dokumentation die dazu notwendigen Informationen.

1.1. Aufbau dieser Dokumentation

Diese Dokumentation enthält:

- in Kapitel 2 das iG/Script Benutzerhandbuch mit einer allgemeinen Einführung in die Konzepte von ICS (**i**nfoGrips **C**onversion **S**ystem) und die Programmierung mit iG/Script
- in Kapitel 3 das iG/Script Referenzhandbuch mit einer detaillierten Beschreibung aller zur Verfügung stehenden Kassen und Methoden

1.2. Konventionen

In dieser Dokumentation werden folgende Konventionen eingehalten:

Kursiv	Namen von Dateien und URL's
fett	neue Begriffe, Namen von Funktionen oder Methoden
`courier`	Programmtext oder Eingaben im Betriebssystem

2. Skriptsprache

2.1. Einleitung

Schnittstellen werden heute meist als C-, C++ oder BASIC-Programme codiert. Der Programmieraufwand für die Erstellung einer Schnittstelle ist vor allem im Bereich der GIS-Systeme gross, da neben Sachdaten auch Geometriedaten übertragen werden müssen. Dies führt dazu, dass nur relativ wenige Schnittstellen (hauptsächlich DXF) mit begrenztem Funktionsumfang von den Systemherstellern angeboten werden.

Mit der Entwicklung von **ICS** wurde ein anderer Weg beschritten. Bei ICS handelt es sich um ein flexibles Schnittstellensystem für die schnelle Entwicklung von Schnittstellenprogammen. Folgende Ideen lagen der Entwicklung von ICS zugrunde:

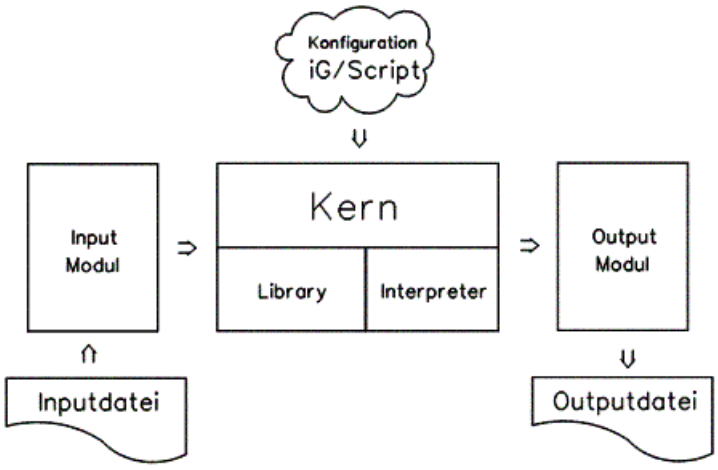
- Jedes Schnittstellenprogramm kann in einen Input- und in einen Outputmodul zerlegt werden. Dabei liest der Inputmodul Objekte aus der Inputdatei und wandelt sie in ein neutrales internes Objektformat um. Der Outputmodul nimmt Objekte vom Inputmodul entgegen und schreibt sie in die Outputdatei. Damit wird eine Entkoppelung des Schnittstellenprogramms in zwei unabhängige, wiederverwendbare Module erreicht

- Im **Kern** werden die Funktionen zusammengefasst, welche von allen Modulen benötigt werden (z.B. String- und Geometriefunktionen). Dadurch müssen diese Funktionen nur einmal programmiert werden

- Der Datenfluss der Objekte vom Inputmodul zum Outputmodul wird nicht durch ein compiliertes Programm gesteuert, sondern über die Skriptsprache **iG/Script**

Nachfolgend ist die Architektur einer ICS-Schnittstelle dargestellt:

Abbildung 1.



☞ Der Kern und die Input- bzw. Outputmodule werden von der infoGrips GmbH entwickelt. iG/Script-Programme können auch vom Benutzer geschrieben werden.

Der ICS-Kern enthält, neben den allgemeinen Funktionen für die Behandlung von Datenstrukturen (Strings, Geometrie, Maps etc.), einen Interpreter für die iG/Script-Sprache. iG/Script ist eine allgemeine Programmiersprache mit einem vordefinierten Satz von Standardfunktionen. Die Sprache enthält neben arithmetischen-, logischen- und Zuweisungsoperationen, auch Kontrollstrukturen wie IF und WHILE. Daneben bietet sie die Möglichkeit den Sprachumfang

durch Prozeduren zu erweitern. Als Basistypen kennt die iG/Script-Sprache die Typen Integer, Real, String, Boolean und Geometrie. Strukturierte Datentypen können über den Datentyp Map erzeugt werden. Input- bzw. Outputmodule können, falls nötig, zusätzliche Datentypen implementieren.

Zu bestehenden Programmiersprachen ist iG/Script am ehesten mit der Programmiersprache FORTH verwandt. Mit FORTH verbindet sie, dass sie ebenfalls alle Operationen über einen **Stack** abwickelt und eine klammerfreie Darstellung von Ausdrücken verwendet. Der wesentliche Unterschied zwischen FORTH und iG/Script liegt darin, dass FORTH für die hardwarenahe Programmierung entwickelt wurde, iG/Script hingegen ist eine hardwareunabhängige Sprache die sich besonders für die Entwicklung von Schnittstellenapplikationen eignet.

2.2. Objekte

Objekte sind die Basisdatenstruktur von iG/Script. Objekte können mit iG/Script über die Input- bzw. Outputmodule gelesen, geschrieben und über Methoden des Kerns manipuliert werden. Bei den Objekten wird zwischen einfachen und stukturierten bzw. zwischen benannten und unbenannten Objekten unterschieden.

2.2.1. Einfache Objekte

Unter einfachen Objekten versteht man in iG/Script Konstanten der **Basistypen**. In iG/Script sind folgende Basistypen fest eingebaut:

- **String**, z.B. `'hello, World'`
- **Integer**, z.B. `1234`
- **Real**, z.B. `123.456`
- **Boolean**, d.h. `TRUE` oder `FALSE`
- **Geometrie**, d.h. Punkt, Linie oder Fläche
- **Blob**, d.h. beliebiger binärer Wert

2.2.2. Strukturierte Objekte

Strukturierte Objekte bestehen aus einer oder mehreren Komponenten. Komponenten haben einen Namen und einen Wert. Der Wert einer Komponente kann ein Basystyp oder auch ein Objekt sein, d.h. strukturierte Objekte sind im allgemeinen Fall hierarchisch aufgebaut.

Beispiel 1. IN-Objekt

Ein Objekt, dass vom INTERLIS-Inputmodul ILIN aus der Tabelle LFP gelesen wurde, hat u.a. folgende Komponenten (`Entstehung`, `Nummer`, `Numpos`, etc.):

Komponente	**Wert**
`Entstehung`	`NULL`
`Nummer`	`'1234.223'`
`Numpos`	`670530.23/270820.74`
`...`	`...`

Strukturierte Objekte werden in iG/Script als sog. Map's implementiert. Mehr dazu in Abschnitt 2.9, „Die drei Bedeutungen von Maps“.

2.2.3. Listen

Listen sind beliebige, einfach verknüpfte Ketten von ICS Objekten. Jeder Basistyp oder jedes sturkturierte Objekt (Map) oder auch jede Liste kann wieder Element einer Liste sein. Für die Bearbeitung von Listen stellt der ICS Kern eine Reihe von eingebauten Methoden zur Vefügung. Listen können z.B. dynamisch wachsen oder schrumpfen, Listen können durchsucht werden, etc.

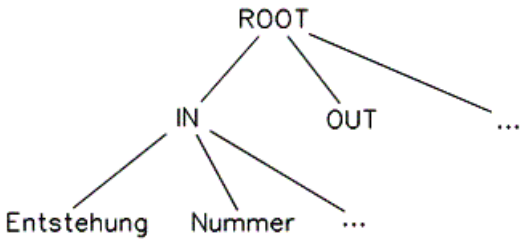
2.2.4. Arrays

Arrays sind Gruppen von ICS Objekten. Die einelnen Objekte eines Arrays können über einen ganzahligen Index angesprochen (indiziert) werden. Array haben im Gegensatz zu den Listen immer eine fixe Länge.

2.2.5. Benannte Objekte

Die meisten Objekte von iG/Script sind Komponenten des vordefinierten Systemobjekts `ROOT`. Komponenten von `ROOT` haben einen Namen über den sie in der Scriptsprache direkt adressiert werden können. z.B. hat das IN-Objekt aus dem ILIN Beispiel im System den Namen `ROOT.IN` und die Nummer Komponente den Namen `ROOT.IN.Nummer`.

Abbildung 2.



Der Name des Objekts entspricht also dem Pfad von der Wurzel (ROOT) bis zum Objekt (analog zu Dateinamen in einem Dateibaum). Allgemein kann ein Objekt in iG/Script über folgenden Namen angesprochen werden:

```
<Objektname> := ROOT.<Komponentename>
<Komponentenname> := <Name> | <Komponentenname> . <Name>
<Name> := <Zeichenkette>
```

Um die Schreibweise zu vereinfachen, ist es erlaubt `ROOT` am Anfang des Objektnamens wegzulassen. Die `Nummer` Komponente des IN-Objekts kann daher in iG/Script auch als `IN.Nummer` angesprochen werden.

2.3. Ausdrücke

iG/Script arbeitet ähnlich wie ein HP-Taschenrechner, d.h. iG/Script kommt vollständig ohne Klammerung von Ausdrücken aus. Alle Zwischenergebnisse werden auf dem sog. **Stack** abgelegt. Will man z.B. 1 + 2 berechnen so schreibt man in iG/Script:


```
1 2 +
```

weitere Beispiele:

Ausdruck	iG/Script-Lösung
`2 * (1 + 7)`	`2 1 7 + *`
`(8 - 5) / 3`	`8 5 - 3 /`

Konstanten der Basistypen werden sofort auf den Stack geschoben. Sehen wir uns also den Stack während der Berechnung des Ausdrucks 2 * 3 genauer an:

Eingabe	Stack
2	`[2]`
3	`[2 3]`
*	`[6]`

In der obigen Darstellung wird der Stack durch `[ ... ]` repräsentiert (diese Darstellung wurde aus Platzgründen gewählt, normalerweise wird ein Stack oft als senkrechte Kolonne dargestellt). Das am weitesten rechts stehende Objekt wird aktuelles, erstes oder auch **oberstes Objekt** des Stacks genannt. Alle iG/Script-Methoden nehmen die N obersten Objekte des Stacks als Argumente und liefern M Objekte auf dem Stack als Resultate zurück. Z.B. konsumiert die Multiplikationsmethode 2 Argumente auf dem Stack und liefert nach ihrer Beendigung das Resultat als oberstes Objekt des Stacks zurück.

2.4. Klassen und Methoden

Neben arithmetischen Methoden kennt iG/Script-Skript weitere **Standardmethoden** für die Bearbeitung von Strings oder die Ausgabe auf den Bildschirm etc.. Gleichartige Methoden werden zu einer **Klasse** zusammengefasst (eine Klasse wird durch ein Modul implementiert). Die iG/Script-Standardmethoden gehören z.B. zu der Klasse ICS. Will man eine Methode aus einer bestimmten Klasse aufrufen so schreibt man in iG/Script

```
<Klasse>.<Methode>
```

also z.B. `ILIN.READ_OBJECT` (Aufruf der Methode `READ_OBJECT` aus der Klasse `ILIN`). Die Standardmethoden sind in der Klasse ICS implementiert. Methoden der Klasse ICS müssen jedoch nicht mit dem Klassennamen qualifiziert werden (ICS.<Methode). Es genügt den Methodennamen anzugeben, die Angabe des Klassennamens ICS ist jedoch erlaubt.

Beispiel 2. Abgekürzter Methodenaufruf

`-` und `ICS.-` sind beides erlaubte Schreibweisen für die Subtraktionsmethode

Man sieht, dass Methoden gleich adressiert werden wie Objekte. Tatsächlich sind Methoden in ICS nichts anderes als Komponenten eines Klassenobjekts.

2.5. Ein Beispiel

Damit das nachfolgende Beispiel besser verstanden werden kann, sollen hier kurz einige wichtige Standardmethoden eingeführt werden (eine vollständige Liste der Standardklassen und Standardmethoden ist im Referenzhandbuch enthalten):

Methode	Beschreibung
DISP	zeigt das oberste Objekt des Stacks auf dem Bildschirm an
DUP	dupliziert das oberste Objekt auf dem Stack
APP	hängt die obersten beiden Objekte des Stacks als String zusammen
LEN	berechnet die Länge des obersten Stringobjekts

Damit sind wir nun in der Lage, ein einfaches iG/Script-Beispiel anzugeben. In dem Beispiel soll die Länge des Strings `'hello, World'` berechnet werden und zusammen mit dem String auf den Bildschirm ausgegeben werden. Wir benutzen dazu die Standardmethoden `DUP`, `LEN`, `APP` und `DISP`.

Beispiel 3. iG/Script Code

```
'hello, World' ! Stack: ['hello, World']
DUP ! Stack: ['hello, World', 'hello, World']
LEN ! Stack: ['hello, World', 12]
APP ! Stack: ['hello, World12']
DISP ! Stack: [] Ausgabe auf Bildschirm: hello, World12
```

Man sieht an diesem Beispiel wie die Methoden ihre Argumente vom Stack entgegennehmen und ihre Resultate wieder auf dem Stack ablegen. Dieses Verhalten ist für iG/Script typisch.

2.6. Zuweisungen

Oft ist es notwendig, Zwischenresultate einer Berechnung permanent abzulegen. Der Stack ist dazu ungeeignet, da seine Werte durch die Methoden ständig neu überschrieben werden. Für die permanente Speicherung von Objekten verfügt die iG/Script-Sprache deshalb über den Objekttyp **Map**. Maps müssen durch den Benutzer deklariert werden und zwar vor der ersten ausführbaren Anweisung im Skript. Mapdeklarationen haben folgende Syntax:

```
MAP <Mapname>
   {<Name1> => <Wert1>}+
END_MAP
```

Beispiel 4. Map Deklaration

```
MAP TEST
   VAR1 => 'hello, World'
   VAR2 => 'dies ist ein Test'
END_MAP
```

In obigem Beispiel enthält das Map-Objekt TEST die Komponenten VAR1 und VAR2. Der Wert von VAR1 ist 'hello, World' und der Wert von VAR2 ist 'dies ist ein Test'. Der Wert einer Komponente kann im Skript mit:

```
<Mapname>.<Komponentenname>
```

angesprochen werden (s.a. 2.2). Analog zu den Basistypen wird der Wert einer Komponente sofort nachdem sie im Skript angetroffen wurde auf den Stack geschoben.

Beispiel 5. Arbeiten mit dem Stack

```
TEST.VAR1 ! Stack: ['hello, World']
TEST.VAR2 ! Stack: ['hello, World','dies ist ein Test']
APP ! Stack: ['hello, Worlddies ist ein Test']
DISP ! Stack: [] Bildschirm: hello, Worlddies ist ein Test
```

Komponenten kann wie folgt ein neuer Wert zugewiesen werden:

=> <Komponente>

oder

-> <Komponente>

Mit => wird jeweils das oberste Objekt des Stacks der Komponente zugewiesen. Mit -> wird nur der Teilstring des obersten Stackelements bis zum ersten Komma zugeordnet (nur bei Stringobjekten möglich). Der Rest des Strings bleibt auf dem Stack erhalten.

Beispiel 6. Benutzung von => und ->

```
'hallo' => TEST.VAR1 ! TEST.VAR1 enthält nun 'hallo'
                     !Stack: []
'hello, World' -> TEST.VAR1 ! TEST.VAR1 enthält nun 'hello'
                            ! Stack: [' World']
=> TEST.VAR2 ! TEST.VAR2 enthält nun ' World'
             ! Stack: []
```

Es ist auch erlaubt nicht existierenden Komponenten einen Wert zu zuweisen (die Map muss allerdings bereits existieren). Will man also z.B. den Wert von `TEST.VAR1` und `TEST.VAR2` vertauschen so kann man wie folgt vorgehen:

Beispiel 7. Zuweisung von Variablen

```
TEST.VAR1 => TEST.TMP ! die Zwischenvariable TEST.TMP wird erzeugt
TEST.VAR2 => TEST.VAR1
TEST.TMP => TEST.VAR2
```

Komponenten werden häufig als Variablen in einem iG/Script Programm gebraucht. Für die Speicherung von Variablen stellt ICS daher die vordefinierte Map `VAR` bereit

2.7. Kommentare

Jeglicher Text nach einem ! bis zum Ende der aktuellen Zeile wird in iG/Script als Kommentar aufgefasst.

Beispiel 8. Kommentare

```
IN.RADIUS 2.0 * 3.14159 * ! Berechnung des Kreisumfangs
```

2.8. DISPLAY Methode

Neben der DISP Methode, gibt es noch eine spezielle eingebaute Methode **DISPLAY** welche sich besonders für die Ausgabe von Objekten eignet:

DISPLAY <Objekt>

Gibt den Wert des Objekts auf den Bildschirm aus

DISPLAY $

Gibt das oberste Objekt des Stacks aus (analog DISP)

DISPLAY <String>,<Objekt>

Gibt den Inhalt des Strings und des Objekts aus (z.B. `DISPLAY 'TEST.VAR1=',TEST.VAR1`). Dabei wird der Wert des Objekts in einen String umgewandelt und am Ende von <String> angehängt. Es dürfen auch mehrere Strings und Objekte in der Liste aufgeführt werden. Die einzelnen Werte müssen jeweils durch ein Komma getrennt werden.

Neben `DISPLAY` gibt es die Befehle `ERROR` und `STATUS`. Mit `ERROR` können Fehlermeldungen ausgegeben werden. Der ICS interne Fehlerzähler wird dabei um 1 erhöht. `STATUS` erlaubt die Ausgabe von Statusmeldungen. Die Syntax von `ERROR` und `STATUS` enspricht der Syntax von `DISPLAY`.

2.9. Die drei Bedeutungen von Maps

Map-Objekte werden in der iG/Script-Sprache in drei verschiedenen Zusammenhängen gebraucht. Die erste Bedeutung als Variablenspeicher haben wir bereits kennengelernt (s.a. 2.6). Maps können aber auch als Abbildungstabellen oder als Objektspeicher benutzt werden. Betrachten wir z.B. folgende Map:

```
MAP Jahreszeiten
   Januar => Winter
   Mai => Frühling
   Juni => Sommer
   September => Herbst
END_MAP
```

Dieses Map-Objekt kann als Abbildungstabelle von Monaten auf Jahreszeiten aufgefasst werden. Schreibt man in iG/Script z.B. `'Januar' Jahreszeiten` so wird auf dem Stack `'Winter'` abgelegt.

Beispiel 9. Abbildungen mit Maps

```
'Januar' Jahreszeiten DISP ! Bildschirm: 'Winter'
'Mai' Jahreszeiten DISP ! Bildschirm: 'Frühling'
```

Abbildungstabellen können auch kombiniert werden. Es ist z.B. folgende zusätzliche Map definiert

```
MAP Temperatur
   Winter => kalt
   Sommer => warm
   DEFAULT => mittel
END_MAP
```

Dann führen die Ausdücke zu folgenden Resultaten:

```
'Januar' Jahreszeiten Temperatur DISP ! Bildschirm: kalt
'September' Jahreszeiten Temperatur DISP ! Bildschirm: mittel
```

Im letzten Beispiel führte die erste Abbildung `'September' Jahreszeiten` zu `'Herbst'`. `'Herbst'` wurde dann durch Temperatur abbgebildet. Da aber keine Komponente Herbst in der Map Temperatur definiert wurde, wurde der Wert der Komponente `DEFAULT` zurückgeliefert. Falls in einer Map keine `DEFAULT`-Komponente existiert und auf eine unbekannte Komponente zugegriffen wird, bricht der iG/Script-Interpreter mit einer entsprechenden Fehlermeldung ab.

Die letzte Bedeutung von Maps sind Maps als Objektspeicher. Wie bereits erklärt, liefern die Inputmodule Objekte in einem neutralen Datenformat und geben diese an die Outputmodule weiter. In ICS wurde als Konvention festgelegt, dass Inputmodule ihre Objekte in der Map IN liefern müssen. Ebenfalls als Konvention wurde festgelegt, dass alle Outputmodule ihre Objekte in der Map OUT entgegennehmen müssen.

Beispiel 10. IN-Objekt als Map

Es wird mit der Methode `MSIN.READ_OBJECT` ein Text aus einem DGN-File gelesen, dann enthält die IN Map folgende Komponenten:

```
MAP IN
   TXT => 'Test' ! Textinhalt
   GEOM => 740380.150/270810.950 ! Textposition
   ROT => 270.0 ! Textwinkel
   FONT => 27 ! Font etc.
END_MAP
```

Die `IN` und die `OUT` Map können in iG/Script wie jede andere Map bearbeitet werden. Es ist daher möglich die Komponenten als Variablen (`'Test2' => IN.TXT`), oder die Map als Abbildungstabelle zu interpretieren (`'TXT' IN DISP`, Bildschirm: Test).

2.10. Listen

Wie bereits angetönt, kann man mit Listen oder Arrays mehrere Objekte zu einem neuen Objekt zusammenfassen. Für Listen stehen u.A. folgende Methoden zur Verfügung (s.a. Anhang):

Methode	Beschreibung
`CREATE_LIST ! []l list]`	Erzeugt eine leere Liste auf dem Stack.
`APPEND_TO_LIST ! [l list,o object][l list]`	Hängt eine Objekt an die Liste `<l>` an.

Beispiel 11. Erzeugen einer Liste

Folgendes Beispiel zeigt den Einsatz von Listen.

```
CREATE_LIST
'abc' APPEND_TO_LIST
'uvw' APPEND_TO_LIST
=> VAR.L
VAR.L DISP
```

Obwohl Listen keine Arrays sind, kann man trotzdem auf die einzelnen Elemente einer Liste indiziert zugreifen. Es stehen folgende Indexfunktionen zur Verfügung:

Methode	Beschreibung
`list.<i>`	Liefert das `<i>`. Elemente einer Liste auf dem Stack. Das 1. Element hat den Index 1.
`list.FIRST`	Liefert das 1. Element der Liste uns ist daher equivalent zu `list.1`.
`list.LAST`	Liefert das letzte Element der Liste.
`list.SIZE`	Liefert die Anzahl Elemente der Liste.

Beispiel 12. Indizierter Zugriff auf Listen

Aufbauend auf dem letzten Beispiel erhält man folgende Resultate:

```
VAR.L.1 ! ['abc']
VAR.L.FIRST ! ['abc']
VAR.L.LAST ! ['uvw']
VAR.L.SIZE ! [2]
```

2.11. Arrays

Arrays sind spezielle Liste, welche eine fixe, unveränderliche Länge aufweisen. Auf die Elemente eines Arrays kann über einen numerischen Index zugegriffen werden. Der Zugriff auf einzelne Elemente des Arrays ist effizienter möglich als bei Listen. Dafür können Arrays in der Grösse nicht verändert werden.

Für Arrays stehen u.A. folgende Methoden zur Verfügung (s.a. Anhang):

Methode	Beschreibung
`CREATE_ARRAY ! [i size][a array]`	Erzeugt einen leeren Array der Grösse `size`.
`INSERT_ARRAY ! [a array,i index, o object][]`	Schreibt das Objekt an der Position `index` in den Array.

Beispiel 13. Erzeugen eines Arrays

Folgendes Beispiel zeigt den Einsatz von Arrays.

```
5 CREATE_ARRAY => VAR.A
'abc' => VAR.A.0
'uvw' => VAR.A.1
VAR.A DISP ! ganzen Array anzeigen
VAR.A.0 DISP ! 1. Element des Arrays
VAR.A.1 DISP ! 2. Element des Arrays
```

2.12. Kontrollstrukturen

Wie die meisten Programmiersprachen verfügt auch iG/Script über Kontrollstrukturen um den Ablauf eines Programms zu steuern. In iG/Script sind die Kontrollstrukturen `IF` und `WHILE` enthalten.

Syntax der `IF` Kontrollstruktur:

```
IF <Bedingung1> THEN
   <Ausdruck1>
ELSIF <Bedingung2> THEN
   <Ausdruck2>
ELSE
   <Ausdruck3>
END_IF
```

Die IF-Kontrollstruktur hat die übliche Bedeutung, d.h. ist die Bedingung1 erfüllt (TRUE) dann wird Ausdruck1 ausgeführt, falls Bedingung2 erfüllt ist dann wird Ausdruck2 ausgeführt etc. Der ELSE und der ELSIF Zweig in einer IF-Kontrollstruktur sind optional. Der ELSIF-Zweig kann beliebig oft wiederholt werden. Eine IF-Kontrollstruktur ist selber wieder ein Ausdruck, d.h. IF-Kontrollstrukturen können geschachtelt werden.

Syntax von Bedingungen:

```
<Wert1> =  <Wert2> ! Gleichheit
<Wert1> <> <Wert2> ! Ungleichheit
<Wert1> <  <Wert2> ! <Wert1> kleiner als <Wert2>
<Wert1> <= <Wert2> ! <Wert1> kleiner oder gleich <Wert2>
<Wert1> >  <Wert2> ! <Wert1> grösser als <Wert2>
<Wert1> >= <Wert2> ! <Wert1> grösser oder gleich <Wert2>
```

Bsp: `IF TEST.VAR1 = 'hello' THEN 'hello, World' DISP ELSE 'tschüss' DISP END_IF`

Syntax der WHILE-Kontrollstruktur

```
WHILE <Bedingung> DO
   <Ausdruck>
END_WHILE
```

Die WHILE-Kontrollstruktur hat die übliche Bedeutung, d.h. solange die Bedingung erfüllt (TRUE) ist, wird der Ausdruck ausgeführt. Die WHILE-Kontrollstruktur ist selbst wieder ein Ausdruck, d.h. WHILE-Kontrollstrukturen können geschachtelt werden.

Beispiel 14. WHILE Schlaufe

```
! dieses Beispiel zeigt 'hello, World' 10 mal auf
! dem Bildschirm an
0 => VAR.I
WHILE VAR.I < 10 DO
   'hello, World' DISP
   VAR.I INC => VAR.I
END_WHILE
```

Die Ausführung der WHILE-Kontrollstruktur kann an einer beliebigen Stelle mit BREAK oder CONTINUE unterbrochen werden. BREAK verzweigt dabei zur ersten Anweisung nach der WHILE-Kontrollstruktur. CONTINUE verzweigt an den Anfang der WHILE-Kontrollstruktur (d.h. unmittelbar vor die <Bedingung>). Die Anweisungen BREAK und CONTINUE sind nur innerhalb von WHILE-Kontrollstrukturen erlaubt.

Beispiel 15. Beenden der WHILE Schlaufe mit BREAK

```
! dieses Beispiel zeigt 'hello, World' 10 mal auf
! dem Bildschirm an (Variante mit BREAK)
0 => VAR.I
WHILE TRUE DO
   IF VAR.I = 10 THEN
      BREAK
   END_IF
   'hello, World' DISP
    VAR.I INC => VAR.I
END_WHILE
```

Bei mehrfach geschachtelten WHILE-Kontrollstrukturen, unterbrechen BREAK und CONTINUE jeweils die WHILE-Kontrollstruktur in der sie direkt enthalten sind.

2.13. Prozeduren

Das letzte Sprachelement von iG/Script, das noch nicht beschrieben wurde, sind die **Prozeduren**. Mit Prozeduren kann der Benutzer eigene Befehle aus Standardmethoden erzeugen. Prozeduren müssen wie Maps deklariert werden, bevor sie verwendet werden können. Nachfolgend ist die Syntax einer Prozedurdeklaration angegeben:

```
PROCEDURE <Prozedurname>
   <Ausdruck>
END_PROCEDURE
```

Wir wollen nun eine neue Prozedur QUADRAT definieren mit der das Quadrat des obersten Elements des Stacks berechnet wird. Das Resultat soll das oberste Element auf dem Stack ersetzen.


```
PROCEDURE QUADRAT ! [n input][n output]
   DUP *
END_PROCEDURE
```

Der Kommentar `! [n input][n output]` gibt an welche Argumente die Prozedur auf dem Stack erwartet bzw. welche Resultate die Prozedur auf em Stack zurück liefert. Im Fall der Prozedur `QUADRAT` wird z.B. ein numerisches Argument (`n` = Integer oder Real) als Argument erwartet und ein numerisches Resultat auf dem Stack zurück geliefert.

Die neue Prozedur können wir nun wie folgt benutzen:

```
2 ! Stack: [2]
QUADRAT ! Stack: [4] hier wurde DUP * durchgeführt
DISP ! Stack:[] Bilschirmausgabe: 4
```

Prozeduren sind also nichts anderes als benannte Teile eines Skripts. Ihre Argumente beziehen Prozeduren meistens über den Stack. Es können allerdings auch Maps für die Parameterübergabe benutzt werden. Folgendes Beispiel benutzt die vordefinierte Map VAR für die Parameterübergabe:

```
PROCEDURE QUADRAT2
   VAR.VAL
   DUP * => VAR.VAL
END_PROCEDURE

5 => VAR.VAL
QUADRAT2
VAR.VAL DISP ! liefert als Resultat 25 auf dem Bildschirm
```

Prozeduren können an einer beliebigen Stelle mit `RETURN` verlassen werden

Für die Speicherung von Zwischenresultaten innerhalb der Prozedur steht ausserdem noch die vordefinierte Map `LOCAL` zur Verfügung. Jede Prozedur hat ihre eigene `LOCAL` Map welche nicht mit anderen Prozeduren geteilt wird. Der Inhalt der `LOCAL` Map ist ausserdem nur während der Ausführung der Prozedur definiert. Die `LOCAL` Map kann also nicht für die Parameterübergabe zwischen Prozeduren benutzt werden (die `VAR` Map hingegen schon).

Beispiel 16. Einsatz von LOCAL

```
PROCEDURE DISPLAY_LIST ! [l list][]
! Diese Prozedur gibt alle Elemente einer Liste aus und
! und benutzt dazu die Map LOCAL als Zwischenspeicher.
   => LOCAL.LIST
   &LOCAL.LIST RESET_READ
   WHILE &LOCAL.LIST READ_NEXT DO
      DISP
   END_WHILE
END_WHILE
```

2.14. Diverses

2.14.1. Referenzen

Konstanten und Objekte werden zur Laufzeit sofort auf den Stack geschoben. Es wird dabei eine vollständige Kopie des Objekts auf dem Stack abgelegt. Bei Maps und anderen strukturierten Objekten (z.B. Geometrie) kann das Kopieren aus Effizienzgründen unerwünscht sein,

man möchte eigentlich nur eine **Referenz** und keine Kopie des Objekts auf dem Stack ablegen. In iG/Script kann dies erreicht werden indem man dem Objektnamen ein `&` voranstellt (z.B. `&IN`, `&IN.Nummer`).

Map's können nur über eine Referenz auf den Stack geschoben werden. In ICS wird nicht zwischen dem Orginalobjekt und der Referenz unterschieden. Beide sind aus der Sicht von ICS vollständig identisch. Intern sind Referenz und Objekt als Zeiger auf den gleichen Speicherplatz implementiert. Ein Referenzzähler hält fest wie oft ein Objekt in ICS referenziert wird. Falls ein Objekt nicht mehr referenziert wird, wird es automatisch von ICS gelöscht.

Beispiel 17. Beispiel mit Referenzen

```
MAP TEST
END_MAP

&TEST => IN.A ! TEST und IN.A zeigen nun auf das gleiche Objekt
              ! Der interne Referenzzähler des Objekts hat den Wert 2

'hello' => TEST.B
DISPLAY IN.A.B ! Resultat: 'hello'
DISPLAY TEST.B ! Resultat: 'hello'

'World' => IN.A.B
DISPLAY IN.A.B ! Resultat: 'World'
DISPLAY TEST.B ! Resultat: 'World'
```

2.14.2. Der NULL Wert

Objekte haben nicht immer einen definierten Wert. Z.B. kann der INTERLIS Inputmodul (`ILIN`) Komponenten ohne Wert liefern (falls Attribute in INTERLIS als `OPTIONAL` deklariert sind). Diese Komponenten haben den speziellen Wert **NULL**. Mit dem NULL-Wert kann per Definition weder gerechnet, noch kann der NULL-Wert mit anderen Werten verglichen werden. Nur das Zuweisen von NULL-Werten auf Komponenten ist erlaubt. Für die Bearbeitung von NULL-Werten steht ein spezieller Satz von Standardmethoden zur Verfügung:

SET_NULL setzt das oberste Objekt des Stacks auf NULL

```
1234 SET_NULL Resultat [NULL]
```

`IS_NULL` liefert `TRUE` falls das Objekt `NULL` ist

```
IF IN.A IS_NULL THEN
   DISPLAY 'IN.A ist NULL'
END_IF
```

`IS_NOT_NULL` liefert `TRUE` falls das Objekt ungleich `NULL` ist

```
IF IN.A IS_NOT_NULL THEN
   DISPLAY 'IN.A ist nicht NULL'
END_IF
```

NVL liefert das 2. Argument falls das 1. Argument `NULL` ist sonst das 1. Argument

```
IN.A 70 NVL
```

ist daher identisch mit

```
IF IN.A IS_NULL THEN
   70
ELSE
```



```
   IN.A
END_IF
```

Die NVL Methode ist besonders praktisch, wenn sichergestellt werden soll, dass nach einer Zuweisung der Inhalt einer Komponente nicht `NULL` ist.

```
IN.A 70 NVL => OUT.B ! stellt sicher, dass OUT.B nach der
                     ! Zuweisung nie NULL ist
```

2.14.3. Operatoren

Normalerweise dürfen Methoden erst aufgerufen werden, wenn alle ihre Argumente bereits auf dem Stack sind (postfix Notation). Eine Ausnahme dieser Regel bilden die sog. Operatoren die zwischen ihren Argumente geschrieben werden dürfen (Infixnotation). In iG/Script sind z.Zt. folgende Operatoren verfügbar:

Operator	Beschreibung
,	Hängt beide Argumente durch ein Komma getrennt als String zusammen.
.	Hängt die beide Argumente als String zusammen (ohne Komma).

Beispiel 18. Beispiel mit Infix Operatoren

```
'abc' , 1 , 2 ! Resultat: ['abc,1,2']
'abc' . 1 . 2 ! Resultat: ['abc12']
```

Die beiden Beispiele können natürlich auch mit der eingebauten Methode `ICS.APP` programmiert werden. Die Programmierung mit Operatoren ist hier jedoch einfacher und übersichtlicher.

2.14.4. Winkelsystem

Mit iG/Script können Geometrien verarbeitet werden. In der Regel werden Geometrien mit Input-Modulen von Systemen/Formaten gelesen und mit Output-Modulen nach Systeme/Formate geschrieben. Mit Geometrien werden oft Winkel verarbeitet. Die Systeme/Formate können unterschiedliche Winkelsystems aufweisen.

DIe Methoden von iG/Script verwenden immer das mathematische Winkelsystem:

`0 Grad`	Osten
`Bereich`	0.0 .. 360.0
`Positiv`	Gegenuhrzeigersinn zunehmend
`Negativ`	Mit Uhrzeigersinn abnehmend

Winkel aus einem System/Format mit einem zu iG/Script unterschiedlichen Winkelsystem werden von den Input/Output Modulen von/nach dem iG/Script-Winkelsystem umgerechnet.

Beispiel:

INTERLIS Modell:

INTERLIS Modell: Winkel definiert als GRADS 0.0 .. 399.9

Vermessungstechnisch: 0 Grad = Norden, Bereich = 0.0 .. 399.9, Positiv: Uhrzeigersinn, Negativ: Gegenuhrzeigersinn

INTERLIS lesen:

Das Input Modul ILIN liest den Winkel als INTERLIS GRADS und rechnet diesen in einen iG/Script Winkel um.

INTERLIS GRADS Winkel 350 Grad (Nord-Westen) wird zu iG/Script Winkel 135 Grad (Nord-Westen)

INTERLIS Schreiben:

Das Output Modul ILOUT erhält einen iG/Script Winkel und rechnet diesen in INTERLIS GRADS um.

iG/Script Winkelt 135 Grad (Nord-Westen) wird zu INTERLIS GRADS Winkel 350 Grad (Nord-Westen).

2.14.5. Debugging

Für das Debugging von iG/Script Programmen stehen folgende Methoden / Operatoren zur Verfügung:

`DISPLAY oder DISP`

`DISPLAY` oder `DISP` können an beliebigen Stellen im Skript dazu verwendet werden, ein bestimmtes Objekt in der .log Datei auszugeben:

```
DISPLAY IN ! zeigt das aktuelle IN-Objekt an (Praefixnotation)
&IN DISP ! das Gleiche wie oben in Postfixnotation
DISPLAY 'Der ist Wert ist ',IN.VALUE ! Formatierte Ausgabe
                                     ! mit DISPLAY
```

`ICSCPU.DISPLAY_STACK`

Mit `ICSCPU.DISPLAY_STACK` kann an einer beliebigen Stelle in einem iG/Script Programm der Inhalt des Stack ausgegeben werden:

```
1 2 ICSCPU.DISPLAY_STACK
+ ICSCPU.DISPLAY_STACK
```

`HALT`

Mit `HALT` kann man ein iG/Script Programm mit einem DOS-Fehlerstatus <> 0 abbrechen:

```
IF 'IN.Name' EXISTS NOT THEN
   ERROR 'Die Komponenente IN.Name exisitiert nicht'
   HALT
END_IF
```

2.15. iG/Script Direktiven

Mit iG/Script Direktiven können gewisse Eigenschaften eines iG/Script Programms beinflusst werden. Im Gegensatz zu Methoden oder Prozeduren werden Direktiven nur zu Kompilationszeit durch den iG/Script Compiler ausgeführt.

2.15.1. Die |INCL Direktive

Für die bessere Übersichtlichkeit ist es erlaubt, iG/Script-Programme auf mehrere Textdateien zu verteilen. Mit der |INCL Direktive kann an einer beliebigen Stelle eines Skripts ein weiteres Skriptfile eingebunden werden. Die |INCL Direktive muss am Anfang einer neuen Zeile stehen.

Beispiel 19. Die |INCL Direktive

```
|INCL ics.lib ! hier wird die Datei ics.lib eingebunden
```

In der eingebundenen Datei dürfen ebenfalls `|INCL` Direktiven stehen.Die einzubindende Datei kann entweder durch einen absoluten Pfad oder relativ zum Installationsverzeichnis von ICS angegeben werden.

Trotz der Unterteilung der Verzeichnisstruktur in ein System- und ein User-Verzeichnis können Dateireferenzen relativ definiert werden.

Die INTERLIS Tools suchen Dateien *immer* zuerst als absoluter Pfad, dann relativ zum User-Verzeichnis, dann relativ zum System-Verzeichnis, dann relativ zum Verzeichnis `ILTOOLS_DIR`, dann im gleichen Verzeichnis wie die Konfiguration und zuletzt im gleichen Verzeichnis wie die Konfiguration aber im analogen System-Verzeichnis.

Das nachfolgende Skriptbeispiel einer `|INCL` Direktive soll dies nochmals verdeutlichen:

```
! Beispiel ICS Konfiguration: ILTOOLS_DIR\user\script\test\test.cfg
! Diese ICS Konfiguration bindet eine Datei
! relativ zu ILTOOLS_DIR\user und/oder ILTOOLS_DIR\system ein
|INCL \script\util.lib
```

1. Zuerst wird die Datei util.lib unter dem absoluten Pfad `\script\util.lib` gesucht.

   Wird util.lib unter diesem Pfad gefunden so wird sie verarbeitet und nicht weiter gesucht.

   Wird util.lib unter diesem Pfad nicht gefunden, so wird der nächste Schritt durchgeführt.

2. Danach wird die Datei util.lib unter dem Pfad `ILTOOLS_DIR\user\script\util.lib` gesucht.

   Wird util.lib unter diesem Pfad gefunden so wird diese Version verarbeitet und nicht mehr weiter gesucht.

   Wird util.lib unter diesem Pfad nicht gefunden, so wird der nächste Schritt durchgeführt.

3. Danach wird die Datei util.lib unter dem Pfad `ILTOOLS_DIR\system\script\util.lib` gesucht.

   Wird util.lib unter diesem Pfad gefunden so wird diese Version verarbeitet und nicht weiter gesucht.

   Wird util.lib unter diesem Pfad nicht gefunden, so wird der nächste Schritt durchgeführt.

4. Danach wird die Datei util.lib unter dem Pfad `ILTOOLS_DIR\script\util.lib` gesucht.

   Wird util.lib unter diesem Pfad gefunden so wird diese Version verarbeitet und nicht weiter gesucht.

   Wird die util.lib unter diesem Pfad nicht gefunden, so wird der nächste Schritt durchgeführt.

5. Danach wird util.lib unter dem gleichen Pfad wie die ICS Konfiguration unter `ILTOOLS_DIR\user\script\test\util.lib` gesucht.

   Wird util.lib unter diesem Pfad gefunden so wird sie verarbeitet und nicht weiter gesucht.

   Wird util.lib unter diesem Pfad nicht gefunden, so wird der nächste Schritt durchgeführt.

6. Danach wird util.lib unter dem gleichen Pfad wie die ICS Konfiguration aber unter dem System-Ast `ILTOOLS_DIR\system\script\test\util.lib` gesucht.

   Wird util.lib unter diesem Pfad gefunden so wird sie verarbeitet und nicht weiter gesucht.

Wird util.lib unter diesem Pfad nicht gefunden, so wird der nächste Schritt durchgeführt.

7. Es erfolgt eine Fehlermeldung, dass util.lib nicht gefunden werden konnte.

Falls man bewusst eine Datei aus dem System-Verzeichnis verwenden möchte, muss man die Datei relativ zu `ILTOOLS_DIR` definieren:

```
! Diese ICS Konfiguration bindet eine Datei
! aus ILTOOLS_DIR\system ein
|INCL \system\script\util.lib
```

Falls man eine Datei aus dem gleichen Verzeichnis wie die Hauptkonfiguration verwenden möchte, muss man keinen Pfad definieren.

```
! Diese ICS Konfiguration bindet eine Datei
! aus dem Verzeichnis der Hauptkonfiguration ein
|INCL util.lib
```

Schliesslich kann man eine Datei auch über einen absoluten Pfad ansprechen.

```
! Diese ICS Konfiguration bindet eine
! Datei über ihren absoluten Pfad ein
|INCL c:\iltools15\user\script\util.lib
```

Es wird jedoch *dringend* davon abgeraten absolute Pfadnamen in Konfigurationen zu benutzen, weil dadurch die Konfigurationen nicht mehr einfach in andere Verzeichnisse kopierbar sind.

2.15.2. Die |LOAD Direktive

Unter Windows können Input- und Outputmodule als DLL's (Dynamic Link Libraries) implementiert werden. Um die Methoden eines solchen Moduls in der Skriptsprache benutzen zu können, muss der Modul zuerst mit der `|LOAD` Direktive geladen werden.

Beispiel 20. Die |LOAD Direktive

```
|LOAD topo ! hier wird das Topologiemodul geladen
```

Die `|LOAD` Direktive muss im Skript vor der ersten ausführbaren Anweisung stehen.Es können mehrere DLL Module pro Skript geladen werden.

2.15.3. Die |LICENSE Direktive

Die meisten iG/Script Programme benötigen für die Ausführung eine gültige Lizenzdatei. Mit der Lizenzdatei wird angegeben, welche Module innerhalb eines Skripts benutzt werden können und ob mit der Lizenz nur versiegelte Scripts ausführbar sind. Die `|LICENSE` Direktive muss am Anfang einer neuen Zeile stehen.

Beispiel 21. Die |LICENSE Direktive

```
|LICENSE \license\igimp.lic ! Angabe der Lizenzdatei relativ ! zu IG_IMPORT_DIR
```

Die Lizenzdatei kann entweder durch einen absoluten Pfad oder relativ zum Installationsverzeichnis von ICS angegeben werden.

2.15.4. Die |SEALHARD |SEALSOFT Direktiven

iG/Script Programme können versiegelt werden. Die Versiegelung dient dazu, keine oder nur teilweise Veränderungen in der Codierung von iG/Script Programmen zuzulassen.

Die Versiegelung kann genutzt werden, um iG/Script Programme vor ungewollter Veränderung zu schützen. Durch einen Teil des Lizenzschlüssels und den Direktiven `|SEALHARD` und `|SEALSOFT` wird die Art der Versiegelung bestimmt.

Es gibt zwei Arten von Versiegelungen, welche in der Lizenz bestimmt werden:

`SEAL,HARD`

"harte" Versiegelung. Mit der Lizenz können nur "hart" versiegelte iG/Script-Programme ausgeführt werden. In "hart" versiegelteniG/Script-Programmen inklusive den Include-Files können keine Veränderungen vorgenommen werden.

`SEAL,SOFT`

"weiche" Versiegelung. Mit der Lizenz können "weich" und "hart" versiegelte iG/Script-Programme ausgeführt werden. In "weich" versiegelten iG/Script-Programmen inklusive den Include-Files können nur innerhalb der MAP-Definitionen Veränderungen vorgenommen werden.

Die Versiegelung **`SEAL,HARD`** wird wie folgt definiert:

1. In der Lizenzdatei

   In der Lizenzdatei muss folgender Teil enthalten sein:

   ```
   ...,SEAL,HARD,...
   ```

   Damit können mit dieser Lizenz nur "hart" versiegelte iG/Script Programme ausgeführt werden.

2. Im iG/Script Programm

   In iG/Script Programmen muss folgende Direktive enthalten sein:

   ```
   |SEALHARD <Checksum>
   ```

   Wobei `<Cecksum>` einer Check-Summe des iG/Script Programmes entspricht. Die `|SEALHARD` Direktive muss am Anfang einer neuen Zeile stehen.

Die Versiegelung **`SEAL,SOFT`** wird wie folgt definiert:

1. In der Lizenzdatei

   In der Lizenzdatei muss folgender Teil enthalten sein:

   ```
   ...,SEAL,SOFT,...
   ```

   Damit können mit dieser Lizenz "weich" versiegelte iG/Script Programme ausgeführt werden. Ebenfalls können mit dieser Lizenz "hart" versiegelte iG/Script Programme ausgeführt werden.

2. Im iG/Script Programm

   In iG/Script Programmen muss folgende Direktive enthalten sein:

   ```
   |SEALSOFT <Cecksum>
   ```


Wobei <Cecksum> einer Check-Summe des iG/Script Programmes entspricht. Die |SEALSOFT Direktive muss am Anfang einer neuen Zeile stehen.

Die Generierung des Lizenzschlüssels mit den Anteilen SEAL,HARD oder SEAL,SOFT wird durch die infoGrips durchgeführt.

Der Eintrag |SEALHARD <Cecksum> oder |SEALSOFT <Checksum> in ein iG/Script Programm wird durch die infoGrips durchgeführt.

2.16. Starten von iG/Script

iG/Script-Programme sind für sich allein nicht lauffähig. Sie müssen mit ICS, ICS for Windows, iG/Import oder iG/Export gestartet werden. Lesen Sie dazu das entsprechende Kapitel der Benutzerhandbücher.

A. Standard Klassen

1. Einleitung

In diesem Anhang sind alle für iG/Script verfügbaren Klassen und ihre Methoden beschrieben.

1.1. Konventionen

In den folgenden Abschnitten werden alle Module und deren Methoden beschrieben. Alle Methoden werden nach folgendem Muster beschrieben:

Methode	**<Name> [Stack vor Aufruf][Stack nach Aufruf]**
Beschreibung	Beschreibung der Methode.
Beispiel	`Ein kleines Beispiel`

Für jedes Argument bzw. zurückgelieferten Wert einer Methode ist zusätzlich der erlaubte Objekttyp vor dem Argument angegeben. Folgende Abkürzungen werden verwendet:

Abkürzung	Objekttyp
i	Integer
r	Real
n	Nummer d.h. Integer oder Real
s	String
m	Map
B	Blob
L	List
A	Array
p	Geometrie Aom Typ Point
l	Geometrie vom Typ Line

r	Geometrie vom Typ Rand
a	Geometrie vom Typ Area
g	beliebiger Geometrietyp
*	beliebiger Objekttyp

2. Klasse CODEC

2.1. Allgemeines

Mit der Klasse CODEC können STRING's kodiert und dekodiert werden. Die Klasse CODEC muss mit `|LOAD codec` geladen werden.

Der Basiszeichensatz für die Kodierung und die Dekodierung ist der Windows 1252 Zeichensatz. Das heisst, Strings werden vom und zum Windows 1252 Zeichensatz kodiert und dekodiert.

2.2. BASE64

Methode **`CODEC.BASE64_ENCODE [s input][s output]`**

Beschreibung Kodiert <input> nach dem BASE64 Verfahren.

Beispiel
```
'Administrator:infogrips' CODEC.BASE64_ENCODE ['QWRtaW5pc3RyYXRvcjppbmZvZ3JpcHM=']
```

Methode **`CODEC.BASE64_DECODE[s input][s output]`**

Beschreibung Dekodiert einen BAS64 kodierten STRING.

Beispiel
```
'QWRtaW5pc3RyYXRvcjppbmZvZ3JpcHM=' CODEC.BASE64_DECODE ['Administrator:infogrips']
```

2.3. UTF-8

Methode **`CODEC.UTF8_ENCODE[s input][s output]`**

Beschreibung Kodiert einen UTF-8 String.

Beispiel
```
'Beispiel: ÄÖÜ' CODEC.UTF8_ENCODE ['Beispiel: Ã„Ã–Ãœ']
```

Methode **`CODEC.UTF8_DECODE[s input][s output]`**

Beschreibung Dekodiert einen UTF-8 String.

Beispiel
```
'Beispiel: Ã„Ã–Ãœ' CODEC.UTF8_DECODE ['Beispiel: ÄÖÜ']
```

2.4. URL

Methode **`CODEC.URL_ENCODE[s input][s output]`**

Beschreibung Kodiert einen URL String. Beachten Sie, dass Sie nicht eine ganze URL kodieren, ansonsten werden URL-reservierte Zeichen wie ? oder & ebenfalls kodiert. Kodieren Sie nur die Werte, die Sie in eine URL integrieren.

Beispiel
```
'Beispiel: ÄÖÜ' CODEC.URL_ENCODE ['Beispiel%3A%20%C4%D6%DC']
```

Methode	**CODEC.URL_DECODE[s input][s output]**
Beschreibung	Dekodiert einen URL String.
Beispiel	`'Beispiel%3A%20%C4%D6%DC' CODEC.URL_DECODE ['Beispiel: ÄÖÜ']`

2.5. XML

Methode	**CODEC.XML_ENCODE[s input][s output]**
Beschreibung	Kodiert einen XML String. Beachten Sie, dass Sie nicht XML-Tags kodieren, ansonsten werden XML-reservierte Zeichen wie < oder > ebenfalls kodiert. Kodieren Sie nur die Werte, die Sie in ein XML integrieren.
Beispiel	`'Beispiel: <>&%' CODEC.XML_ENCODE ['Beispiel: <>&&#37;']`

Methode	**CODEC.XML_DECODE[s input][s output]**
Beschreibung	Dekodiert einen XML String.
Beispiel	`'Beispiel: <>&&#37;' CODEC.XML_DECODE ['Beispiel: <>&%']`

3. Klasse DB

3.1. Allgemeines

Mit den DB-Methoden kann auf eine relationale Datenbank via ODBC oder RIS (nur Microstation) zugegriffen werden. Die Klasse DB muss nicht mit |LOAD geladen werden.

3.2. Verbindung mit der Datenbank

Methode	**DB.CONNECT [s driver,s db,s user,s password] [b status]**
Beschreibung	Die Methode DB.CONNECT stellt die Verbindung mit einer Datenbank her. Es kann nur eine Datenbank gleichzeitig verbunden werden. Für <driver> sind im Moment die Werte 'MSDB' oder 'ODBC' zulässig. Mit 'MSDB' kann eine unter Microstation konfigurierte Datenbank angesprochen werden. Die weiteren Parameter haben je nach Datenbank eine andere Bedeutung. Unter RIS muss für <db> das RIS-Schema angegeben werden. Mit einer direkten Microstation-Oracle Verbindung muss für <db> der Oracle-Connectstring angegeben werden (inkl. Benutzer und Passwort z.B. 'scott/tiger@t:alfa:db1'). Mit 'ODBC' kann eine ODBC fähige Datenbank angesprochen werden. Der ODBC Treiber muss unter Windows installiert worden sein. Für den Parameter <db> muss eine ODBC-Quelle angegeben werden, für <user> und <password> ein gültiger Benutzername mit Passwort (falls die Datenbank dies verlangt). Bei einer ODBC Datenbank kann an Stelle der Datenquelle in <db> auch ein ODBC Connect String angegeben werden (s.a. Dokumentation zu "ConfigDataSource" im ODBC Treiber). Für MS-Access sieht der ODBC Connect String z.B. wie folgt aus: DRIVER={Microsoft Access Driver (*.mdb)};DBQ=e:\geoshop.mdb Wird über einen ODBC Connect String mit der ODBC Datenbank verbunden, muss unter Windows keine ODBC Datenquelle vorhanden sein. Dieses Verfahren funktioniert jedoch nicht unbedingt mit jeder ODBC Datenbank. Mindestens für Microsoft Jet Datenbanken (d.h. MS-Access oder MS-Excel) ist das beschriebene Verfahren jedoch möglich. Bemerkung: unter Microstation 95

können ODBC-Datenbanken auch über den 'MSDB' Treiber angesprochen werden

Beispiel

```
! Verbindung mit der RIS-Datenbank test4
'MSDB' 'test4' '' '' DB.CONNECT [TRUE]
```

Methode **DB.DISCONNECT [] []**

Beschreibung Baut die Verbindung zu der zuletzt mit DB.CONNECT verbundenen Datenbank wieder ab.

Beispiel

```
DB.DISCONNECT
```

Methode **DB.IS_CONNECTED [] [b state]**

Beschreibung Prüft ob eine Datenbankverbindung besteht.

Beispiel

```
DB.IS_CONNECTED [TRUE]
```

3.3. Datenbankzugriff

Methode **DB.OPEN_CURSOR [s selectstmt] [i cursor, b status]**

Beschreibung Öffnet einen Datenbankcursor <cursor>. <selectstmt> muss ein gueltiges SQL-Selectstatement sein. Das Ergebnis von DB.OPEN_CURSOR kann mit DB.FETCH_ROW abgefragt werden. Bemerkung: wegen eines Fehlers im RIS-Microstation Treiber kann unter Microstation V5.0 nur ein Cursor gleichzeitig geöffnet werden.

Beispiel

```
IF 'select * from feature' DB.OPEN_CURSOR THEN
   => VAR.C1
ELSE
   DISPLAY 'unable to open cursor'
END_IF
```

Methode **DB.FETCH_ROW [i cursor, m map] [b status]**

Beschreibung liefert einen Record von <cursor> in <map> zurück. Die Attributnamen werden als Komponentennamen eingetragen.

Beispiel

```
VAR.C1 &ROW DB.FETCH_ROW [TRUE]
```

Methode **DB.FETCH_VALUE [s selectstmt] [* value,b status]**

Beschreibung Liefert einen Wert in <value> zurück. Es wird automatisch ein Cursor für das Selectstatement geöffnet und geschlossen. Das Selectstatement <selectstmt> muss so formuliert worden sein, dass es nur einen Wert zurückliefert. Falls <status> FALSE ist, wird kein Wert zurückgeliefert.

Beispiel

```
'select max(mslink) from feature' DB.FETCH_VALUE [1027,TRUE]
```

Methode **DB.CLOSE_CURSOR [i cursor] []**

Beschreibung Schliesst einen Datenbankcursor <cursor>.

Beispiel

```
VAR.C1 DB.CLOSE_CURSOR
```

3.4. Datenbankmanipulation

Methode **DB.EXEC_SQL [s sqlstmt][b status]**

Beschreibung Führt das SQL-Statement <sqlstatement> aus.

Beispiel
```
'delete from vermpunkte' DB.EXEC_SQL [TRUE]
```

Methode **DB.INSERT_ROW [m map][b status]**

Beschreibung Speichert die Map <map> als neuen Datensatz in der Datenbank. In der Map <map> muss in der Komponente TABLE der Name der Datenbanktabelle stehen. Alle weiteren Komponentennamen müssen gültige Attributnamen sein.

Beispiel
```
MAP ROW
END_MAP
'POINTS' => ROW.TABLE
'123a' => ROW.NUMBER
600000.000 => ROW.X
200000.000 => ROW.Y
&ROW DB.INSERT_ROW [TRUE]
```

3.5. Weitere

Methode **DB.GET_ODBC_SOURCES [m map][]**

Beschreibung Liefert in der Input-Map alle zur Vefügung stehenden ODBC-Sourcen. Eine Datenbankverbindung muss nicht bestehen.

Beispiel
```
&ODBC_SOURCES DB.GET_ODBC_SOURCES
```

Methode **DB.ODBC_SOURCE_EXISTS [s source][b status]**

Beschreibung Testet ob die ODBC-Datenquelle <source> existiert.

Beispiel
```
'geoshop' DB.ODBC_SOURCE_EXISTS [TRUE]
```

Methode **DB.CREATE_ODBC_SOURCE [b system,m map][b status]**

Beschreibung Erzeugt eine ODBC-Datenquelle. Mit <system> kann angegeben werden, ob eine System Datenquelle (TRUE) oder eine Benutzer Datenquelle (FALSE) eingerichtet werden soll. In <m> wird der ODBC-Treiber (DRIVER), die Datenquelle (DSN) und die Parameter des ODBC-Treibers angegeben. Die Treiberparameter sind der Beschreibung des ODBC-Treibers dokumentiert (z.B. \windows\system32\odbcjet.hlp unter "ConfigDataSource" für MS-Access oder MS-Excel). Hinweis: Die Methode funktioniert nur, wenn der ODBC-Treiber die Funktion "ConfigDataSource" vollständig unterstützt.

Beispiel
```
MAP ODBC_SOURCE
   DRIVER => 'Microsoft Access Driver (*.mdb)'
   DSN => geoshop ! Name der Datenquelle
   DBQ => e:\geoshop\standard\server\dba\geoshop.mdb
END_MAP
TRUE &ODBC_SOURCE DB.CREATE_ODBC_SOURCE
```

Methode **DB.DELETE_ODBC_SOURCE [b system,s source] [b status]**

Beschreibung Löscht die angegebene ODBC-Datenquelle <source>. Mit <system> wird angegeben ob es sich bei der Datenquelle um eine System-Datenquelle

(TRUE) oder um eine Benutzer-Datenquelle (FALSE) handelt. Hinweis: Die Methode funktioniert nur, wenn der ODBC-Treiber die Funktion "ConfigDataSource" vollständig unterstützt.

Beispiel

```
TRUE 'geoshop' DB.DELETE_ODBC_SOURCE
```

Methode **DB.GET_PRODUCT [][s Version, s Produkt]**

Beschreibung Gibt das Produkt und die Version der Datenbank zurück. Eine Datenbankverbindung muss mit DB.CONNECT erstellt sein. Funktioniert zur Zeit nur mit dem ODBC-Treiber.

Beispiel

```
DB.GET_PRODUKT [09.00.0111,Oracle]
```

Methode **DB.GET_INFO [m map] []**

Beschreibung Liefert in der Input-Map Datenbankinformationen. Eine Datenbankverbindung muss mit DB.CONNECT erstellt sein. Funktioniert zur Zeit nur mit dem ODBC-Treiber. Der Aufbau der Informationen in der Input-Map wird wie folgt geliefert:

```
MAP DB_INFO
   PRODUKT => <Produkt>
   VERSION => <Version>
   TABLES => MAP
             :
             <Tablename> => MAP
                            :
                            COLUMNS => MAP
                                       <Colname> => MAP
                                                    TYPENAME
                                                    DATATYPE
                                                    LENGTH
                                                    PRECISION
                                                    SCALE
                                                    RADIX
                                                    REMARK
                                       :            END_MAP
                                       END_MAP
             :              END_MAP
             END_MAP
END_MAP
```

Beispiel

```
&DB_INFO DB.GET_INFO
```

Methode **DB.SET_CASE_SENSITIVE [b sensitive][]**

Beschreibung Definiert ob Tabellen- und Attributnamen Case-Sensitive - Klein- und Grossschreibung- (<sensitive> = TRUE) verarbeitet werden sollen oder nicht (<sensitive> = FALSE. Per Default ist die Verarbeitung nicht Case-sensitive.

Beispiel

```
TRUE DB.SET_CASE_SENSITIVE
```

Methode **DB.SET_SILENT [b silent][]**

Beschreibung Sollen Datenbankmeldungen unterdrückt (<silent> =TRUE) oder nicht unterdrückt (<silent> =FALSE) werden. Per Default werden die Meldungen nicht unterdrückt.

Beispiel

```
TRUE DB.SET_SILENT
```

Methode	**DB.COMMIT [][]**
Beschreibung	Führt ein sql-commit auf der Datebank aus, falls die Datenbank diesen Befehl vearbeiten kann.
Beispiel	`DB.COMMIT`

Methode	**DB.ROLLBACK [][]**
Beschreibung	Führt ein sql-rollback auf der Datebank aus, falls die Datenbank diesen Befehl vearbeiten kann.
Beispiel	`DB.ROLLBACK`

4. Klasse DIALOG

4.1. Allgemeines

Mit dem Modul Dialog können vom Benutzer interaktiv Parameter über Windows Dialogboxen abgefragt werden. Die Klasse DIALOG wird mit:

```
|LOAD dialog
```

in einem iG/Script Programm verfügbar gemacht.

4.2. Exportierte Methoden

Methode	**DIALOG.MESSAGE_BOX [s message][]**
Beschreibung	Stellt auf dem Bildschirm eine Meldungsbox mit <message> als Meldungstext dar. Der Benutzer muss die Meldung mit OK quitieren.
Beispiel	`'hello, world' DIALOG.MESSAGE_BOX`

Methode	**DIALOG.SELECT_BOX [s message][b status]**
Beschreibung	Stellt auf dem Bildschirm eine Meldungsbox mit OK und CANCEL Knopf dar. Falls der Benutzer die Meldung mit OK quitiert, wird TRUE sonst FALSE zurück geliefert.

Beispiel	`IF 'how do you feel today ?' DIALOG.SELECT_BOX THEN`
	`  DISPLAY 'ok'`
	`ELSE`
	`  DISPLAY 'bad'`
	`END_IF`

Methode	**DIALOG.GET_FILENAME [s title,s extension,b existing][s fname,b status]**
Beschreibung	Fragt vom Benutzer einen gültigen Dateinamen ab. In <title> muss der Titel des Meldungsbox angegeben werden. In <extension> wird die Dateiendung angegeben. Falls für <existing> TRUE übergeben wird, können vom Benutzer nur existierende Dateinamen angegeben werden. Falls <status> TRUE ist, wird in <fname> der vollständige Dateiname (inkl. Pfad) der selektierten Datei zurückgegeben.

Beispiel	`IF 'Select INTERLIS Input File' 'itf' TRUE DIALOG.GET_FILENAME THEN`
	`  => VAR.FNAME`
	`ELSE`

```
   DISPLAY 'script aborted by user'
   HALT
END_IF
```

Methode **DIALOG.GET_FILENAMES [s title,s extension][li fnames,b status]**

Beschreibung Fragt vom Benutzer extistierende Dateinamen ab. In <title> muss der Titel des Meldungsbox angegeben werden. In <extension> wird die Dateiendung angegeben. Falls <status> TRUE ist, wird in <fnames> eine Liste mit den vollständigen Dateinamen (inkl. Pfad) der selektierten Dateien zurückgegeben.

Beispiel

```
IF 'Select INTERLIS Input Files' 'itf'  DIALOG.GET_FILENAMES THEN
   => VAR.LIST
ELSE
   DISPLAY 'script aborted by user'
   HALT
END_IF
```

Methode **DIALOG.GET_DIRECTORY [s title][s directory,b status]**

Beschreibung Fragt vom Benutzer ein gültiges Directory ab. In <title> muss der Titel des Meldungsbox angegeben werden. Falls <status> TRUE ist, wird in <directory> der vollständige Directoryname (inkl. Pfad) des selektierten Directories zurückgegeben.

Beispiel

```
IF 'Select Input Directory' DIALOG.GET_DIRECTORY THEN
   => VAR.DIRECTORY
ELSE
   DISPLAY 'script aborted by user'
   HALT
END_IF
```

Methode **DIALOG.GET_STRING [s message][s user_input]**

Beschreibung Fragt vom Benutzer einen String über eine Dialogbox ab. <message> wird in der Dialogbox als Meldungstext dargestellt.

Beispiel

```
'Enter RIS-Schema' DIALOG.GET_STRING => DB_PARAM.CONNECT
```

Methode **DIALOG.GET_LIST [s message, m map][b status]**

Beschreibung Fragt vom Benutzer Listenelemente in einer Liste ab. <message> wird in der Dialogbox als Meldungstext dargestellt. In <map> wird eine Map mit den Listenelementen übergeben. Als Name wird in der Map der darzustellende Text des Listenelements eingetragen. Als Wert wird in der Map der Wert ON oder OFF eingetragen. Beim Wert ON ist das Listenelement vorselektioniert. Falls <status> TRUE ist, sind in <map> die selecktierten Listenelement mit einem Wert ON enthalten.

Beispiel

```
MAP MAPLIST
   Bodenbedeckung,BoFlaeche => ON
   Bodenbedeckung,BoFlaeche_Area => ON
   Bodenbedeckung,BoFlaeche_Boundary => OFF
END_MAP

IF ' Select Topics,Tables:' &MAPLIST DIALOG.GET_LIST NOT THEN
   DISPLAY 'script aborted by user'
```



```
   HALT
END_IF
```

Methode **DIALOG.GET_ODBC_SOURCE [s message, m source-map, m selected-map][b status]**

Beschreibung Fragt vom Benutzer eine ODBC-Source mit User und Password ab. <message> wird in der Dialogbox als Meldungstext dargestellt. In <source-map> werden die ODBC-Sources für die Auswahlliste übergeben. (Die ODBC-Sourcen könnten mit DB.GET_ODBC_SOURCES ermittelt werden). In <selected-map> wird die vom User selektierte ODBC-Source zurückgegeben.

Beispiel

```
MAPCREATE => VAR.ODBC_SOURCES
MAPCREATE => VAR.ODBC_SELECTED

&VAR.ODBC_SOURCES DB.GET_ODBC_SOURCES

IF 'Select ODBC Source' &VAR.ODBC_SOURCES &VAR.ODBC_SELECTED DIALOG.GET_ODBC_SOURC
ELSE
   ERROR 'script aborted by user'
   HALT
END_IF
```

Die <selected-map> beinhaltet folgende Komponenten:

```
MAP VAR.ODBC_SELECTED
   SOURCE => <ODBC-Source>
   USER => <User>
   PASSWORD => <Password>
   ORACLESPATIAL => <TRUE|FALSE>
END_MAP
```

Methode **DIALOG.GET_ODBC_FILE [s message, m source-map, s file-extension, b file-existing][s input, b status]**

Beschreibung Fragt vom Benutzer eine ODBC-Source mit User und Password oder ein Datenbank File ab. <message> wird in der Dialogbox als Meldungstext dargestellt. In `<source-map>` werden die ODBC-Sources für die Auswahlliste übergeben. (Die ODBC-Sourcen könnten mit `DB.GET_ODBC_SOURCES` ermittelt werden). In `<file-extension>` wird die Dateiendung für ein File angegeben. Falls für `<file-existing>` `TRUE` übergeben wird, können vom Benutzer nur existierende Dateinamen angegeben werden. Falls `<status>` `TRUE` ist, wird in `<input>` die ODBC-Source in der Form `<source,user,password>` oder der vollständige Dateiname (inkl. Pfad) der selektierten Datei zurückgegeben.

Beispiel

```
MAPCREATE => VAR.ODBC_SOURCES

&VAR.ODBC_SOURCES DB.GET_ODBC_SOURCES

IF 'Select ODBC or File' &VAR.ODBC_SOURCES '*' TRUE DIALOG.GET_ODBC_FILE THEN
   => OPT.input
ELSE
   ERROR 'script aborted by user'
   HALT
END_IF
```

Methode **DIALOG.GET_ARCGIS [s message, s file-extension, b file-existing][s input, b status]**

Beschreibung Fragt vom Benutzer eine ArcGIS-SDE-Connection mit Server,Instance,Database,User,Password und Version oder ein ArcGIS Datenbank File ab. <message> wird in der Dialogbox als Meldungstext dargestellt. In <file-extension> wird die Dateiendung für ein File angegeben. Falls für <file-existing> TRUE übergeben wird, können vom Benutzer nur existierende Dateinamen angegeben werden. Falls <status> TRUE ist, wird in <input> die ArcGIS-SDE-Connection in der Form <server,instance,database,user,password,version> oder der vollständige Dateiname (inkl. Pfad) der selektierten Datei zurückgegeben.

Beispiel

```
IF 'Select SDE or File' 'mdb;gdb' TRUE DIALOG.GET_ARCGIS THEN
   => OPT.input
ELSE
   ERROR 'script aborted by user'
   HALT
END_IF
```

5. Klasse DIRECTORY

5.1. Allgemeines

Mit den Methoden der Klasse DIRECORY können Directories bearbeitet werden. Die Klasse DIRECORY muss nicht mit |LOAD geladen werden.

5.2. Kreieren und Löschen von Directories

Methode **DIRECTORY.CREATE [s directory][b status]**

Beschreibung Die Methode kreiert ein Directory.

Beispiel

```
'c:\mydir' DIRECTORY.CREATE [TRUE]
```

Methode **DIRECTORY.REMOVE [s directory][b status]**

Beschreibung Die Methode löscht ein Directory. Das Directory muss leer sein. Das Directory darf nicht das aktuell aktive Directory sein.

Beispiel

```
'c:\mydir' DIRECTORY.REMOVE [TRUE]
```

Methode **DIRECTORY.EXISTS [s directory][b status]**

Beschreibung Die Methode testet, ob das Directory existiert.

Beispiel

```
'c:\mydir' DIRECTORY.EXISTS [TRUE]
```

Methode **DIRECTORY.CHANGE [s directory][b status]**

Beschreibung Die Methode wechselt das aktuell aktive Directory. Das Directory muss existieren.

Beispiel

```
'c:\mydir' DIRECTORY.CHANGE [TRUE]
```

Methode **DIRECTORY.CURRENT[][s directory]**

Beschreibung Die Methode gibt das aktuell aktive Directory zurück.

Beispiel

```
DIRECTORY.CURRENT ['c:\mydir']
```

6. Klasse ICS

6.1. Allgemeines

In diesem Abschnitt sind alle Methoden der eingebauten Klasse ICS beschrieben. Die Methoden der Klasse ICS müssen nur optional mit ICS. qualifiziert werden. Die Klasse ICS muss ausserdem nicht mit |LOAD geladen werden.

6.2. Arithmetische Methoden

Methode `ICS.+ [n a,n b][n a+b]`

Beschreibung Die Methode + addiert die ersten beiden Werte des Stacks und legt das Resultat auf dem Stack ab.

Beispiel
```
2 3 + [5]
```

Methode `ICS.- [n a,n b][n a-b]`

Beschreibung Die Methode - subtrahiert den obersten Wert des Stacks vom zweitobersten Wert des Stacks und legt das Resultat auf dem Stack ab.

Beispiel
```
7 5 - [2]
```

Methode `ICS.* [n a,n b][n a*b]`

Beschreibung Die Methode * multipliziert die ersten beiden Werte des Stacks und legt das Resultat auf dem Stack ab.

Beispiel
```
2 3 * [6]
```

Methode `ICS./ [n a,n b][n a/b]`

Beschreibung Die Methode / dividiert den zweitobersten Wert des Stacks durch den obersten Wert des Stacks und legt das Resultat auf dem Stack ab.

Beispiel
```
3 4 / [0.75]
```

Methode `ICS.INC [i a][i a+1]`

Beschreibung Erhöht die Zahl <a> auf dem Stack um 1.

Beispiel
```
5 INC [6]
```

Methode `ICS.DEC [i a][i a-1]`

Beschreibung die Zahl <a> auf dem Stack um 1.

Beispiel
```
6 DEC [5]
```

Methode `ICS.ROUND [r a,i dez][r a]`

Beschreibung Rundet die Zahl <a> auf <dez> Dezimalen. ROUND kann auch mit Stringargument für <dez> aufgerufen werden z.B. 1.567 '1' ROUND. In diesem Fall wird als Resultat ein auf <dez> Dezimalen gerundeter String (hier: '1.6') zurück gegeben.

Beispiel
```
1.567 1 ROUND [1.6]
```

Methode **ICS.FLOOR [r a][r floor(a)]**

Beschreibung Rundet die Zahl <a> auf die nächst tiefere ganze Zahl ab.

Beispiel
```
1.5 FLOOR [1.0]
```

Methode **ICS.CEIL [r a][r ceil(a)]**

Beschreibung Rundet die Zahl <a> auf die nächst höhere ganze Zahl auf.

Beispiel
```
1.5 CEIL [2.0]
```

Methode **ICS.SQRT [r a][r sqrt(a)]**

Beschreibung Berechnet die Wurzel der reellen Zahl <a>.

Beispiel
```
2.0 SQRT [1.414]
```

Methode **ICS.POWER [n x, n y][r pow(x,y)]**

Beschreibung Berechnet <x> hoch <y>.

Beispiel
```
2.0 3.0 POWER [8.0]
```

Methode **ICS.MOD [i x, i y][i x % y]**

Beschreibung Berechnet den Module von <x> und <y>, den Rest aus der Division der ganzen Zahlen <x> und <y>.

Beispiel
```
12 10 MOD [2]
```

Methode **ICS.SIN [r w][r sin(w)]**

Beschreibung Berechnet den Sinus des Winkels <w>. Der Winkel muss in Grad angegeben werden.

Beispiel
```
90.0 SIN [1.0]
```

Methode **ICS.ASIN [r a][r asin(a)]**

Beschreibung Berechnet den Arcussinus von <a>. Der Winkel wird in Grad zurückgegeben.

Beispiel
```
1.0 ASIN [90.0]
```

Methode **ICS.COS [r w][r sin(w)]**

Beschreibung Berechnet den Cosinus des Winkels <w>. Der Winkel muss in Grad angegeben werden.

Beispiel
```
90.0 COS [0.0]
```

Methode **ICS.ACOS [r a][r asin(a)]**

Beschreibung Berechnet den Arcuscosinus von <a>. Der Winkel wird in Grad zurückgegeben.

Beispiel
```
0.5 ACOS [60.0]
```

Methode **ICS.TAN [r w][r sin(w)]**

Beschreibung Berechnet den Tangens des Winkels <w>. Der Winkel muss in Grad angegeben werden.

Beispiel
```
45.0 TAN [1.0]
```

Methode `ICS.ATAN2 [r y, r x][r atan2(a)]`
Beschreibung Berechnet die ATAN2 Funktion. Der Winkel wird in Grad zurückgegeben.
Beispiel
```
1.0 3.0 SQRT ATAN2 [30.0]
```

6.3. Boolesche Methoden

Methode `ICS.AND [b x, b y][b (a AND b)]`
Beschreibung Boolesche AND-Operation von zwei booleschen Werten <x> und <y>.
Beispiel
```
TRUE TRUE AND [TRUE]
```

Methode `ICS.OR [b x, b y][b (a OR b)]`
Beschreibung Boolesche OR-Operation von zwei booleschen Werten <x> und <y>.
Beispiel
```
TRUE FALSE OR [TRUE]
```

Methode `ICS.XOR [b x, b y][b (a AND b)]`
Beschreibung Boolesche XOR-Operation von zwei booleschen Werten <x> und <y>.
Beispiel
```
TRUE TRUE XOR [FALSE]
```

Methode `ICS.NOT [b x, b y][b (a AND b)]`
Beschreibung Boolesche NOT-Operation eines booleschen Wertes <x>.
Beispiel
```
TRUE NOT [FALSE]
```

Methode `ICS.INTAND [i x, i y][i (x & y)]`
Beschreibung Boolesche Bit-AND-Operation von zwei Integer32 Werten <x> und <y>.
Beispiel
```
3 1 INTAND [1]
```

Methode `ICS.INTOR [i x, i y][i (x | y)]`
Beschreibung Boolesche Bit-OR-Operation von zwei Integer32 Werten <x> und <y>.
Beispiel
```
2 1 INTOR [3]
```

Methode `ICS.INTXOR [i x, i y][i (x ^ y)]`
Beschreibung Boolesche Bit-XOR-Operation von zwei Integer32 Werten <x> und <y>.
Beispiel
```
2 0 INTXOR [2]
```

Methode `ICS.INTSHIFTL [i x, i y][i (x << y)]`
Beschreibung Shift Left Operation eines Integer32 Wertes <x> um <y> Bits.
Beispiel
```
1 2 INTSHIFTL [4]
```

Methode `ICS.INTSHIFTR [i x, i y][i (x >> y)]`
Beschreibung Shift Right Operation eines Integer32 Wertes <x> um <y> Bits.
Beispiel
```
4 2 INTSHIFTR [1]
```

6.4. Stringmethoden

Methode `ICS.LEN [s str][i länge]`

Beschreibung Berechnet die Länge des Strings <str>. Die Länge wird als Integer zurückgegeben.

Beispiel
```
'hello, World' LEN [12]
```

Methode `ICS.EXTRL [s str][s rest,s extract]`

Beschreibung Extrahiert aus dem String <str> einen String <extract>. Es wird bis zum ersten Blank oder Komma extrahiert. Der Extraktstring und der Reststring <rest> werden auf dem Stack abgelegt. Falls der Rest die Länge 0 aufweist, wird für den Rest der Wert NULL auf dem Stack abgelegt.

Beispiel
```
'hello, World' EXTRL [' World','hello']
'hello' EXTRL [NULL,'hello']
```

Methode `ICS.EXTRL1 [s str][s extract]`

Beschreibung Wie EXTRL. Es wird jedoch nur der <extract> geliefert.

Beispiel
```
'hello, World' EXTRL1 ['hello']
```

Methode `ICS.EXTRLS [s str][s rest,s extract]`

Beschreibung Wie EXTRL. Als Trennzeichen wird aber nur das Komma berücksichtigt.

Beispiel
```
'hello   ,World' EXTRLS ['World','hello   ']
'hello' EXTRLS [NULL,'hello']
```

Methode `ICS.EXTRLN [s str,i n][s rest, s extract]`

Beschreibung Extrahiert aus <str> <n> Zeichen und legt den Reststring und den Extraktstring auf dem Stack ab.

Beispiel
```
'hello, World' 5 EXTRLN [' ,World','hello']
```

Methode `ICS.EXTRLN1 [s str,i n][s extract]`

Beschreibung Extrahiert aus <str> <n> Zeichen und legt diese als Extraktstring auf dem Stack ab.

Beispiel
```
'hello, World' 5 EXTRLN1 ['hello']
```

Methode `ICS.EXTRR [s str][s rest, s extract]`

Beschreibung Wie EXTRL mit dem Unterschied, dass von rechts extrahiert wird.

Beispiel
```
'hello,World' EXTRR ['hello','World']
```

Methode `ICS.EXTRR1 [s str][s rest, s extract]`

Beschreibung Wie EXTRL1 mit dem Unterschied, dass von rechts extrahiert wird.

Beispiel
```
'hello,World' EXTRR1 ['World']
```

Methode `ICS.EXTRRN [s str,i n][s rest, s extract]`

Beschreibung Wie EXTRLN mit dem Unterschied, dass von rechts extrahiert wird.

Beispiel
```
'hello,World' 5 EXTRRN ['hello,','World']
```

Methode `ICS.EXTRRN1 [s str,i n][s extract]`

Beschreibung EXTRLN1 mit dem Unterschied, dass von rechts extrahiert wird.

Beispiel
```
'hello,World' 5 EXTRRN1 ['World']
```

Methode `ICS.EXTRCHAR [s str,i n][s extract]`

Beschreibung Extrahiert das Zeichen des n. Index des Strings. Ist der Index n < 0 order n >= strlen dann wird NULL zurückgegeben.

Beispiel
```
'hello, World' -1 EXTRCHAR [NULL]
'hello, World' 0 EXTRCHAR ['h']
'hello, World' 11 EXTRCHAR ['d']
'hello, World' 12 EXTRCHAR [NULL]
```

Methode `ICS.APP [s s1, s s2][s str]`

Beschreibung Hängt den String <s2> an den String <s1> an und legt das Resultat <str> auf dem Stack ab.

Beispiel
```
'World' 'hello' APP ['helloWorld']
```

Methode `ICS.LOC [s str,s suchstring][s str, i pos]`

Beschreibung Sucht <suchstring> in <str>. Falls der <suchstring> in <str> enthalten ist, gibt LOC die Position <pos> des 1. Buchstabens von <suchstring> in <str> zurück. Falls <suchstring> nicht gefunden wird, wird NULL für <pos> zurückgegeben. LOC wird meist zusammen mit EXTRLN1 benutzt.

Beispiel
```
'hello, World' 'Wo' LOC ['hello, World',7]
'hello, World' 'hello' LOC ['hello, World',0]
'hello, World' 'Wu' LOC ['hello, World,NULL]
'hello, World 'o' LOC EXTRLN1 ['hell']
```

Methode `ICS.LOCR [s str, s suchstring][s str, i pos]`

Beschreibung Sucht <suchstring> in <str> von rechts anstatt von links. Falls der <suchstring> in <str> enthalten ist, gibt LOC die Position <pos> des 1. Buchstabens von <suchstring> in <str> zurück. Falls <suchstring> nicht gefunden wird, wird NULL für <pos> zurückgegeben. LOCR wird meist zusammen mit EXTRLN1 benutzt.

Beispiel
```
'hello.txt.txt' '.txt' LOCR EXTRLN1 ['hello.txt']
```

Methode `ICS.LOCCHAR [s str][s str, pos i]`

Beschreibung Wie LOC mit dem Unterschied, dass nach dem 1. Alphazeichen gesucht wird.

Beispiel
```
'1234567hello' LOCCHAR ['1234567hello',7]
'1234567hello' LOCCHAR EXTRLN1 ['1234567']
```

Methode `ICS.TO_UPPER [s value][s result]`

Beschreibung Wandelt den String <value> in einen String mit Grossbuchstaben um.

Beispiel
```
'hello, world' TO_UPPER ['HELLO, WORLD']
```

Methode `ICS.TO_LOWER [s value][s result]`

Beschreibung Wandelt den String <value> in einen String mit Kleinbuchstaben um.

Beispiel
```
'HELLO, WORLD' TO_LOWER ['hello, world']
```

Methode **ICS.TRIM [s value][s result]**

Beschreibung Eliminiert Leerzeichen am Anfang und am Ende eines Strings.

Beispiel
```
'  abc   ' TRIM ['abc']
```

Methode **ICS.LJUST [* value, i length][s result]**

Beschreibung Wandelt das Objekt <value> in einen String um und macht den String <result> linksbündig und mindestens <length> Zeichen lang.

Beispiel
```
'  abc   ' 20 LJUST ['abc                 ']
```

Methode **ICS.RJUST [* value, i length][s result]**

Beschreibung Wandelt das Objekt <value> in einen String um und macht den String <result> rechtsbündig und mindestens <length> Zeichen lang.

Beispiel
```
'  abc   ' 20 RJUST ['                 abc']
```

Methode **ICS.STARTS_WITH[s value,s prefix][b result]**

Beschreibung Prüft ob der String <value> mit dem Prefix <prefix> anfängt.

Beispiel
```
IF 'hello, world' 'hello' STARTS_WITH THEN
! do something
END_IF
```

Methode **ICS.ENDS_WITH[s value,s postfix][b result]**

Beschreibung Prüft ob der String <value> mit dem Prefix <postfix> endet.

Beispiel
```
IF 'hello, world' 'world' ENDS_WITH THEN
! do something
END_IF
```

Methode **ICS.REPLACE_CHAR[s value,s char1,s char2][s result]**

Beschreibung Ersetzt alle Vorkommen des Buchstaben <char1> durch <char2>.

Beispiel
```
'abcabc' 'a' 'u' REPLACE_CHAR ['ubcubc']
```

Methode **ICS.REPLACE_STRING[s value,s search,s replace][s result]**

Beschreibung Ersetzt alle Vorkommen des Strings <search> durch den String <replace>.

Beispiel
```
'abcabc' 'ab' '12' REPLACE_CHAR ['12c12c']
```

6.5. Stackmethoden

Methode **ICS.POP [* objekt][]**

Beschreibung Vernichtet das oberste Objekt auf dem Stack.

Beispiel
```
'hello, World' POP []
```

Methode **ICS.DUP [* objekt][* objekt,* objekt]**

Beschreibung Dupliziert das oberste Objekt auf dem Stack.

Beispiel
```
'hello, World' DUP ['hello, World','hello, World']
'hello, World' DUP POP ['hello, World']
```

Methode **ICS.RDUP [* objekt][* objekt,* objekt]**

Beschreibung Erzeugt eine zusätzliche Referenz auf das oberste Objekt des Stacks.

Beispiel
```
'hello, World' RDUP ['hello, World','hello, World']
'hello, World' RDUP POP ['hello, World']
```

Methode **ICS.SWAP [* objekt1,* objekt2][* objekt2,* objekt1]**

Beschreibung Vertauscht die obersten beiden Elemente des Stacks.

Beispiel
```
1234,'hello, World' SWAP ['hello, World',1234]
```

6.6. Konversionsmethoden

Methode **ICS.TO_STRING [* Objekt][s str]**

Beschreibung Wandelt das oberste Objekt des Stacks in einen String um.

Beispiel
```
1234 TO_STRING ['1234']
1.234 TO_STRING ['1.234']
TRUE TO_STRING ['TRUE']
```

Methode **ICS.TO_REAL [* objekt][s str]**

Beschreibung Wandelt das oberste Element des Stacks in einen Real um. Als Inputtyp ist String oder Integer zulässig.

Beispiel
```
'1.234' TO_REAL [1.234]
1234 TO_REAL [1234.0]
```

Methode **ICS.TO_INT [* objekt][s str]**

Beschreibung Wandelt das oberste Objekt des Stacks in einen Integer um. Als Inputtyp ist String oder Real zulässig. Bei reellen Zahlen wird auf die nächtste ganze Zahl gerundet.

Beispiel
```
1234.567 TO_INT [1235]
```

Methode **ICS.TO_POINT [s string][p punkt]**

Beschreibung Wandelt das oberste Element des Stacks in eine Geometrie vom Typ Punkt um.

Beispiel
```
'1.0/2.0/3.0' TO_POINT [POINT(1.0/2.0/3.0)]
```

Methode **ICS.CHR [i ascii][s ch]**

Beschreibung Wandel den ASCII-Code <ascii> in das entsprechende Zeichen um.

Beispiel
```
65 CHR ['A']
```

Methode **ICS.ASCII [s ch][i ascii]**

Beschreibung Berechnet den ASCII-Code des Zeichens <ch>.

Beispiel
```
'A' ASCII [65]
```

Methode **ICS.GET_TYPE [* objekt][s type]**

Beschreibung Gibt den Typ von <objekt> als String zurück.

Beispiel

```
1 GET_TYPE ['int']
1.2 GET_TYPE ['real']
TRUE GET_TYPE ['boolean']
'abc' GET_TYPE ['string']
&IN GET_TYPE ['map']
'1.0/2.0' TO_POINT GET_TYPE ['point']
VAR.L1 GET_TYPE ['line']
VAR.A1 GET_TYPE ['area']
```

Methode **ICS.BINSTRING_TO_INT [s string][i a]**

Beschreibung Wandelt einen String aus Binärziffern in einen 32-Bit Integer um.

Beispiel

```
'10101011' BINSTRING_TO_INT [171]
```

Methode **ICS.INT_TO_BINSTRING [i a][s string]**

Beschreibung Wandelt einen 32 Bit Integer in einen String aus Binärziffern um. Der String ist immer 32 Zeichen lang.

Beispiel

```
171 INT_TO_BINSTRING ['00000000000000000000000010101011']
```

Methode **ICS.HEXSTRING_TO_INT [s string][i a]**

Beschreibung Wandelt einen String aus Hexziffern in einen 32-Bit Integer um. Die Hexziffern 'A' - 'F' dürfen als Gross- oder Kleinbuchstaben übergeben werden.

Beispiel

```
'AB' HEXSTRING_TO_INT [171]
```

Methode **ICS.INT_TO_HEXSTRING [i a][s string]**

Beschreibung Wandelt einen 32 Bit Integer in einen String aus Hexziffern um. Die Hexziffern 'A' - 'F' werden immer als Grossbuchstaben zurück gegeben. Der String ist immer 8 Zeichen lang.

Beispiel

```
171 INT_TO_HEXSTRING ['000000AB']
```

Methode **ICS.HEXSTRING_TO_REAL [s string][r a]**

Beschreibung Wandelt einen String aus Hexziffern in einen 64-Bit Real um. Die Hexziffern 'A' - 'F' dürfen als Gross- oder Kleinbuchstaben übergeben werden.

Beispiel

```
'0F03F' HEXSTRING_TO_REAL [1.0]
```

Methode **ICS.REAL_TO_HEXSTRING [r a][s string]**

Beschreibung Wandelt einen 64 Bit Real in einen String aus Hexziffern um. Die Hexziffern 'A' - 'F' werden immer als Grossbuchstaben zurück gegeben. Der String ist immer 16 Zeichen lang.

Beispiel

```
1.0 INT_TO_HEXSTRING ['000000000000F03F']
```

Methode **ICS.HEXSTRING_TO_STRING [s string][s string]**

Beschreibung Wandelt einen String aus Hexziffern in String um. Die Hexziffern 'A' - 'F' dürfen als Gross- oder Kleinbuchstaben übergeben werden.

Beispiel

```
'68656C6C6F' HEXSTRING_TO_STRING ['hello']
```

Methode `ICS.STRING_TO_HEXSTRING [s string][s string]`

Beschreibung Wandelt einen String in einen String aus Hexziffern um. Die Hexziffern 'A' - 'F' werden immer als Grossbuchstaben zurück gegeben.

Beispiel
```
'hello' INT_TO_HEXSTRING ['68656C6C6F']
```

Methode `ICS.HEXSTRING_TO_BLOB [s string][B blob]`

Beschreibung Wandelt einen String aus Hexziffern in einen BLOB um. Die Hexziffern 'A' - 'F' dürfen als Gross- oder Kleinbuchstaben übergeben werden.

Beispiel
```
'0011AAff' HEXSTRING_TO_BLOB [BLOB]
```

Methode `ICS.BLOB_TO_HEXSTRING [B blob][s string]`

Beschreibung Wandelt einen BLOB in einen String aus Hexziffern um. Die Hexziffern 'A' - 'F' werden immer als Grossbuchstaben zurück gegeben.

Beispiel
```
VAR.BLOB BLOB_TO_HEXSTRING ['0011AAFF']
```

Methode `ICS.REAL_TO_DEGREE [r degree][r s, i m, i d]`

Beschreibung Wandelt einen Winkel in Grad vom Type `REAL` nach Grad, Minuten, Sekunden.

Beispiel
```
45.25 REAL_TO_DEGREE [0.0 15 45]
```

Methode `ICS.DEGREE_TO_REAL [i d, i m, r s] [r degree]`

Beschreibung Wandelt einen Winkel von Grad, Minuten, Sekunden nach Grad vom Type `REAL`.

Beispiel
```
45 15 0.0 DEGREE_TO_REAL [45.25]
```

Methode `ICS.REAL_TO_DEGREE_STRING [r degree][s d.m.s]`

Beschreibung Wandelt einen Winkel in Grad vom Type `REAL` nach Grad, Minuten, Sekunden vom Typ `STRING`.

Beispiel
```
45.25 REAL_TO_DEGREE_STRING ['45.15.00.00']
```

Methode `ICS.DEGREE_STRING_TO_REAL [s d.m.s] [r degree]`

Beschreibung Wandelt einen Winkel in Grad, Minuten, Sekunden vom Typ `nach` Grad vom Type `REAL` .

Beispiel
```
'45.15.00.00' DEGREE_STRING_TO_REAL [45.25]
```

6.7. Mapmethoden

Methode `ICS.VAL [s objektliste][s werteliste]`

Beschreibung Wertet eine Objektliste aus und ersetzt die Objekte durch ihre aktuellen Werte.

Beispiel
```
! Annahme: in der Mapkomponente IN.POS ist der
! Wert 5.0/10.0/0.0 und in der Mapkomponente
! IN.BLOCK ist der Wert 'PUNKT' gespeichert.
'OUT.BLOCK' VAL ['PUNKT']
'OUT.BLOCK,IN.POS' VAL ['PUNKT,5.0/10.0/0.0']
'abc,IN.POS' VAL ['abc,5.0/10.0']
```

Methode **ICS.VAL2 [s objektliste][s werteliste]**

Beschreibung Wertet eine Objektliste aus und ersetzt die Objekte durch ihre aktuellen Werte. Die Auswertung erfolgt analog zum DISPLAY-Befehl.

Beispiel
```
1234 'Wert=<$>' VAL2 ['Wert=<1234>']
```

Methode **ICS.REF [s objektname][* objekt]**

Beschreibung Liefert eine Referenz auf das Objekt mit Namen <objektname> zurück.

Beispiel
```
'ROOT.IN' REF [map(IN)]
```

Methode **ICS.INDEX [m map,s key][* objekt]**

Beschreibung Liefert die Komponente aus <m> mit Namen <key>. Falls die Komponente nicht existiert, wird der Skript abgebrochen.

Beispiel
```
&IN 'abc' INDEX ['uvw']
```

Methode **ICS.EXISTS [s komponentenname][b status]**

Beschreibung Gibt TRUE zurück falls dies Mapkomponente existiert, FALSE sonst.

Beispiel
```
'OUT.BLOCK' EXISTS [TRUE]
'OUT.POS' EXISTS [FALSE]
```

Methode **ICS.MAPCREATE [][m map]**

Beschreibung Erzeugt eine neue (unbenannte) Map <map>.

Beispiel
```
MAPCREATE [map(NONAME)]
```

Methode **ICS.MAPRESET [m map][]**

Beschreibung Setzt den internen Lesezeiger einer Map zurück (Vorbereitungsmethode für MAPSCAN).

Beispiel
```
&IN MAPRESET []
```

Methode **ICS.MAPSCAN [m map][* wert,name s,status b]**

Beschreibung Liest die nächste Komponente <name>, <wert> aus der Map <map>. Falls <status> FALSE ist konnte keine Komponente gelesen werden (<wert> und <status> werden in diesem Fall nicht auf den Stack geschoben).

Beispiel
```
&TEST MAPSCAN ['b' 'a' TRUE]
```

Methode **ICS.MAPINS [m map,s komponentenname,* wert][]**

Beschreibung Fügt den Wert <wert> in <map> unter dem Komponentennamen <komponentenname> ein.

Beispiel
```
&TEST 'W1' 'hallo' MAPINS 'TEST.W1' EXISTS [TRUE]
```

Methode **ICS.MAPREM [m map,s komponentenname][b status]**

Beschreibung Löscht den Wert unter dem Komponentennamen <komponentenname>.

Beispiel
```
&TEST 'W1' MAPREM [TRUE]
```

Methode **ICS.MAPCLEAR [m map][]**

Beschreibung Löscht alle Komponenten der Map <map>.

Beispiel
```
&TEST MAPCLEAR
```

Methode **ICS.MAP_IS_EMPTY [m map][b status]**

Beschreibung Testet ob die Map <map> keine Komponenten hat.

Beispiel
```
&TEST MAPCLEAR MAP_IS_EMPTY [TRUE]
```

Methode **ICS.MAPCOPY[m map1,m map2][]**

Beschreibung Kopiert alle Komponenten der map <map1> in die Map <map2>. Alle ursprünglichen Komponenten der Map <map2> werden gelöscht.

Beispiel
```
&TEST1 &TEST2 MAPCOPY []
```

Methode **ICS.MAPCOPY2[m map1,m map2][]**

Beschreibung Kopiert alle Komponenten der map <map1> in die Map <map2>. Alle ursprünglichen Komponenten der Map <map2> werden nicht gelöscht.

Beispiel
```
&TEST1 &TEST2 MAPCOPY []
```

Methode **ICS.MAPSORT [m map, b descending][]**

Beschreibung Sortiert die Einträge in der Map nach den Komponentennamen. descending = FALSE aufsteigend. descending = TRUE absteigend.

Beispiel
```
&TEST FALSE MAPSORT []
```

Methode **ICS.MAPSWAP [m map][m map]**

Beschreibung Kehrt die Abbildung <name> => <wert> einer Map um in <wert> => <name>. Namen werden zu Werte, Werte zu Namen. Die Werte werden in eine String umgewandelt und als Namen eingefügt. Namen werden als String belassen und als Werte eingefügt. Ein DEFAULT-Wert wird beibehalten.

Beispiel
```
&TEST MAPSWAP [map(NONAME)]
```

```
MAP TEST
   a       => 1
   b       => 2
   c       => 2
   DEFAULT => OFF
END_MAP

&TEST MAPSWAP => VAR.TEST2

! VAR.TEST2:
! --- MAP NONAME -------
!    a       => 1
!    b       => 2
!    c       => 2
!    DEFAULT => OFF
! --- END MAP NONAME ---
```

Methode **ICS.SET_LABEL[m map,s label][]**

Beschreibung Setzt das Label der Map <map>.

Beispiel
```
&USER 'USER' SET_LABEL []
```

Methode **ICS.GET_LABEL[m map][s label]**

Beschreibung Fragt das Label der Map <map> ab. Falls kein Label gesetzt wurde, wird NULL zurück gegeben.

Beispiel

```
IF &USER GET_LABEL IS_NOT_NULL THEN
   DISPLAY 'map USER has a label'
END_IF
```

6.8. Listenmethoden

Methode **ICS.CREATE_LIST [][li list]**

Beschreibung Erzeuge ein neues Objekt vom Typ list.

Beispiel

```
CREATE_LIST [list]
```

Methode **ICS.APPEND_TO_LIST [li list, * obj][li result]**

Beschreibung Hängt das Objekt <obj> an die Liste <list> an.

Beispiel

```
CREATE_LIST 'abcd' APPEND_TO_LIST [list]
```

Methode **ICS.RESET_READ [li list][]**

Beschreibung Setzt den internen Lesezeiger zurück (Vorbereitung zu READ_NEXT).

Beispiel

```
&VAR.LIST RESET_READ []
```

Methode **ICS.READ_NEXT [li list][* obj, b status]**

Beschreibung Liest das nächste Objekt <obj> aus der Liste <list>.

Beispiel

```
&VAR.LIST RESET_READ
WHILE &VAR.LIST READ_NEXT DO
   DISPLAY $
END_WHILE
```

6.9. Linkmethoden

Prozedur **ICS.CREATE_LINK [s key,i orderpos][link ref]**

Beschreibung Erzeugt einen Link für das Schreiben einer Referenz. `orderpos` wird nur gesetzt wenn `orderpos > 0`. Zum Beispiel wird in INTERLIS 2 in der Outputdatei für die Rolle `REF="<key>"` und `ORDERPOS="<orderpos>"` gesetzt.

Beispiel

```
'key1' 1 CREATE_LINK [LINK(:0::key1:1)]
```

Prozedur **ICS.SET_LINK_BID [link ref, s bid][link ref]**

Beschreibung Setzt in einem Link die Basket ID `bid`.

Beispiel

```
'bid1' SET_LINK_BID [LINK(:0:bid1:key1:1)]
```

Prozedur **ICS.GET_LINK_BID [link ref][s bid]**

Beschreibung Liest aus einem Link die Basket ID `bid`.

Beispiel

```
 [LINK(:0:bid1:key1:1)] GET_LINK_BID ['bid1']
```

Prozedur	**ICS.GET_LINK_KEY [link ref][s key]**
Beschreibung	Liest aus einem Link die Objekt ID key.
Beispiel	`[LINK(:0:bid1:key1:1)] GET_LINK_KEY ['key1']`

Prozedur	**ICS.GET_LINK_ORDERPOS [link ref][i orderpos]**
Beschreibung	Liest aus einem Link die orderpos .
Beispiel	`[LINK(:0:bid1:key1:1)] GET_LINK_ORDERPOS [1]`

6.10. Anzeigemethoden

Methode	**ICS.DISP [* objekt][]**
Beschreibung	Gibt das oberste Element des Stacks auf dem Bildschirm aus.
Beispiel	`['abc'] DUP DISP ! gibt abc auf den Bildschirm aus` `! ohne den Stack zu verändern`

Methode	**ICSCPU.DISPLAY_STACK [][]**
Beschreibung	Gibt den aktuellen Inhalt des Stacks auf den Bildschirm aus. Der Stack wird nicht verändert.
Beispiel	`[] ICSCPU.DISPLAY_STACK []`

6.11. Geometriemethoden

Für die Speicherung und Bearbeitung von Geometriedaten stellt der Kern den Geometrietyp bereit. Eine Geometrie ist entweder eine Fläche, ein Rand, eine Linie oder ein Punkt. Geometrieen sind hierarchisch aufgebaut. So besteht z.B. eine Fläche aus Rändern (ein äusserer Rand und Löcher innerhalb der Fläche). Ein Rand besteht aus Linien. Eine Linie besteht aus Punkten. In einer Linie können Punkte durch Geraden oder Kreisbögen verbunden werden. Durch Standardmethoden können Geometrieen erzeut, durchsucht oder kopiert werden.

Mit den Methoden SET_GATTR und GET_GATTR kann pro Geometrieelement ein Benutzerattribut gesetzt bzw. abgefragt werden.

Methode	**ICS.CREATE_LINE [p p1, p p2][l line]**
Beschreibung	Erzeugt aus zwei Punkten eine Linie.
Beispiel	`VAR.P1 VAR.P2 ICS.CREATE_LINE [line]`

Methode	**ICS.POINTX [p p1][r koord]**
Beschreibung	Gibt die X-Koordinate eines Punkts zurück (horizontale Achse).
Beispiel	`VAR.P1 ICS.POINTX [720340.235]`

Methode	**ICS.POINTY [p p1][r koord]**
Beschreibung	Gibt die Y-Koordinate eines Punkts zurück (vertikale Achse).
Beispiel	`VAR.P1 ICS.POINTY [250370.857]`

Methode	**ICS.POINTZ [p p1][r koord]**

Beschreibung Gibt die Z-Koordinate eines Punkts zurück (Höhe).

Beispiel
```
VAR.P1 ICS.POINTZ [320.750]
```

Methode **ICS.CREATE_ARC [p p1, p p2, p p3][l line]**

Beschreibung Erzeugt aus drei Punkten einen Kreisbogen. Der Kreisbogen führt von <p1> via <p2> nach <p3>.

Beispiel
```
VAR.P1 VAR.P2 VAR.P3 ICS.CREATE_ARC [line]
```

Methode **ICS.APPEND_LINE_POINT [l line, p p1][l line]**

Beschreibung Hängt einen Punkt an eine Linie an. Die Verbindung zwischen dem letzten Punkt der Linie <l> und dem Punkt <p1> ist eine Gerade.

Beispiel
```
VAR.L1 VAR.P1 ICS.APPEND_LINE_POINT [line]
```

Methode **ICS.APPEND_LINE_ARC [l line, p p1, p p2][l line]**

Beschreibung Hängt einen Kreisbogen via <p1>, <p2> an eine Linie an. Als erster Punkt des Kreisbogens wird der letzte Punkt der Linie <l> angenommen.

Beispiel
```
VAR.L1 VAR.P1 VAR.P2 ICS.APPEND_LINE_ARC [line]
```

Methode **ICS.CREATE_RAND [l line][r rand]**

Beschreibung Erzeugt aus der Linie <line> einen Rand.

Beispiel
```
VAR.L1 ICS.CREATE_RAND [rand]
```

Methode **ICS.APPEND_RAND_LINE [r rand,l line][r rand]**

Beschreibung Hängt an einen Rand eine Linie an.

Beispiel
```
VAR.R1 VAR.L1 ICS.APPEND_RAND_LINE [rand]
```

Methode **ICS.CREATE_AREA [r rand][a flaeche]**

Beschreibung Erzeugt aus dem Rand eine Fläche.

Beispiel
```
VAR.R1 ICS.CREATE_AREA [area]
```

Methode **ICS.APPEND_AREA_RAND [a flaeche, r rand][a flaeche]**

Beschreibung Erzeugt eine Insel in einer bestehenden Fläche.

Beispiel
```
VAR.A1 VAR.R1 ICS.APPEND_AREA_RAND [area]
```

Methode **ICS.SET_GATTR [* geometrie, i attr][* geometrie]**

Beschreibung Setzt das Geometrieattribut für die Geometrie <geometrie>.

Beispiel
```
VAR.P1 1234 ICS.SET_GATTR [point]
```

Methode **ICS.GET_GATTR [* geometrie][, i attr]**

Beschreibung Fragt das Geometrieattribut ab.

Beispiel
```
VAR.P1 1234 ICS.SET_GATTR ICS.GET_GATTR [1234]
```

Methode **ICS.ARCPOINT [p ap, p ep, r radius][p mp, b status]**

Beschreibung Berechnet aus einem durch Anfangspunkt <ap>, Endpunkt <ep> und Radius <r> gegebenen Kreisbogen den Mittelpunkt <mp> auf der Peripherie. Falls <status> TRUE ist, konnte die Berechnung erfolgreich durchgeführt werden.

Beispiel
```
VAR.P1 VAR.P2 2.5 ICS.ARCPOINT [point,TRUE]
```

Methode **ICS.GEOMRESET [g geom][]**

Beschreibung Setzt den internen Lesezeiger der Geometrie <geom> auf die erste Subgeometrie von <geom> (Vorbereitungsmethode für ICS.GEOMSCAN). ICS.GEOMRESET kann nicht auf Punkte angewendet werden.

Beispiel
```
&VAR.L1 ICS.GEOMRESET []
```

Methode **ICS.GEOMSCAN [g geom][g subgeom, b status]**

Beschreibung Liest die nächste Subgeometrie <subgeom> der Geometrie <geom>. Falls keine Subgeometrie mehr existiert, wird als <status> FALSE zurückgeliefert. ICS.GEOMSCAN kann nicht auf Punkte angewendet werden.

Beispiel
```
&VAR.L1 ICS.GEOMRESET
WHILE &VAR.L1 GEOMSCAN DO
   DISP
END_WHILE
```

Methode **ICS.GEOMSCAN_LINEARC [l geom][p p3 , p p2, p p1, i status]**

Beschreibung Liest die nächste Linie bzw. den nächsten Kreisbogen einer Liniengeometrie <l>. Bei einer Linie werden die beiden Endpunke (<p1>, <p2>) und bei einem Kreisbogen die Endpunkte (<p1>, <p3>) und ein Stützpunkt <p2> auf der Peripherie zurückgegeben. Der <status> hat folgende Bedeutung: 0: Fehler bzw. keine Linien oder Kreisbögen mehr vorhanden. 1: Es wurde eine Linie gelesen. 2: Es wurde ein Kreisbogen gelesen.

Beispiel
```
&VAR.L1 ICS.GEOMRESET
&VAR.L1 GEOMSCAN_LINEARC [point1 point2 1]
```

Methode **ICS.GEOM_HAS_ARCS [g geom][b status]**

Beschreibung Ermittelt ob die Geometrie Kreisbögen beinhaltet. Wenn die Geometrie Kreisbögen beinhaltet, wird als `status` `TRUE` zurückgegeben, sonst `FALSE` . Nur Linien, Flächen und Ränder können Kreisbögen beinhalten.

Beispiel
```
VAR.GEOM GEOM_HAS_ARCS [TRUE]
```

Methode **ICS.GEOM_HAS_HOLES [g geom][b status]**

Beschreibung Ermittelt ob die Geometrie Löcher beinhaltet. Wenn die Geometrie Löcher beinhaltet, wird als `status` `TRUE` zurückgegeben, sonst `FALSE` . Nur Flächen können Löcher beinhalten.

Beispiel
```
VAR.GEOM GEOM_HAS_HOLES [TRUE]
```

Methode **ICS.GET_POINT_INSIDE [a flaeche][p point]**

Beschreibung Berechnet einen Punkt <point> der sicher innerhalb der Fläche <flaeche> liegt. Inseln werden bei der Berechnung berücksichtigt.

Beispiel
```
&VAR.A1 GET_POINT_INSIDE [point]
```


Methode **ICS.GET_AREA [a flaeche][r flaecheninhalt]**

Beschreibung Berechnet den Flächeninhalt der Fläche <flaeche>. Inseln werden bei der Berechnung berücksichtigt.

Beispiel
```
&VAR.A1 GET_AREA [270.345]
```

Methode **ICS.IS_INSIDE_AREA [a area, p point][b status]**

Beschreibung Berechnet ob ein Punkt innerhalb einer Fläche liegt. Falls ja gibt die Methode das Argument TRUE zurück. Falls nein gibt die Methode das Argument FALSE zurück.

Beispiel
```
VAR.AREA VAR.POINT ICS.IS_INSIDE_AREA [TRUE]
```

Methode **ICS.IS_INSIDE_FENCE [a fence, b overlap, g geom][b status]**

Beschreibung Berechnet ob eine Geometrie innerhalb eines Fences liegt. Falls ja gibt die Methode das Argument `TRUE` zurück. Falls nein, gibt die Methode das Argument `FALSE` zurück. Der Fence muss eine Fläche sein. Die Geometrie kann ein Punkt, eine Linie oder eine Fläche sein. Mit `overlap` kann bestimmt werden, ob die Berechnung im Modus overlap oder inside ausgeführt werden soll. Kreisbögen werden berücksichtigt. Löcher in der Fläche des Fence oder in der Fläche einer Geometrie werden berücksichtigt. Bei der Berechnung von Flächen als Geometrie wird die Flächen getestet und nicht nur die Begrenzungslinien der Fläche. Das heisst zum Beispiel beim Modus `overlap`, dass eine Fläche die den Fence umfasst, innerhalb des Fences ist. Die Begrenzungslinien des Fences gehören ebenfalls zum Fence. Das heisst zum Beispiel, dass ein Punkt der auf der Begrenzungslinie des Fences liegt, innerhalb des Fences ist.

Beispiel
```
VAR.FENCE TRUE VAR.LINE ICS.IS_INSIDE_FENCE [TRUE]
```

Methode **ICS.IS_INSIDE_FENCE_AND_BORDER [a fence, b overlap, g geom][b status]**

Beschreibung Analog zur Methode `ICS.IS_INSIDE_FENCE` mit dem Unterschied, dass die Begrenzungslinien des Fences nicht zum Fence gehören. Das heisst zum Beispiel, dass ein Punkt der auf der Begrenzungslinie des Fences liegt, nicht innerhalb des Fences ist.

Beispiel
```
VAR.FENCE TRUE VAR.LINE ICS.IS_INSIDE_FENCE_AND_BORDER [TRUE]
```

Methode **ICS.FENCE_CUT [a fence, g geom][li list of geometries, b status]**

Beschreibung Berechnet die Verschneidung einer Geometrie mit einem Fence. Der Fence muss eine Geometrie vom Typ `area` sein. Die Geometrie kann vom Typ `point`, `line` oder `area` sein. Ist die ganze Geometrie ausserhalb des Fences, so wird der `status` mit `FALSE` ohne Liste zurückgegeben. Ist die ganze Geometrie oder Teile der Geometrie innerhalb des Fence, so wird der `status` mit `TRUE` und die Teilgeometrien in einer Liste `list` zurückgegeben. Die Geometrien werden immer als Liste zurückgegeben, auch wenn die ganze Geometrie innerhalb des Fence liegt.

Beispiel
```
VAR.FENCE VAR.GEOM ICS.FENCE_CUT [list, TRUE]
```

Methode **ICS.GEOMMOVE [g geom, p vector][g geom]**

Beschreibung Verschiebt eine Geometrie um einen Vektor.

Beispiel
```
VAR.GEOMETRIE VAR.P ICS.GEOMMOVE [geom]
```

Methode **ICS.GEOMSCALE [g geom, r scale][g geom]**

Beschreibung Skaliert eine Geometrie um den Nullpunkt.

Beispiel
```
VAR.GEOMETRIE 10.0 ICS.GEOMSCALE [geom]
```

Methode **ICS.GEOMSCALE2 [g geom, p point, r scale][g geom]**

Beschreibung Skaliert eine Geometrie um den Punkt `point`.

Beispiel
```
VAR.GEOMETRIE VAR.POINT 2.0 ICS.GEOMSCALE2 [geom]
```

Methode **ICS.GEOMROT [g geom, r rot][g geom]**

Beschreibung Rotiert eine Geometrie um den Nullpunkt.

Beispiel
```
VAR.GEOMETRIE 45.0 ICS.GEOMROT [geom]
```

Methode **ICS.GEOMROT2 [g geom, p point, r rot][g geom]**

Beschreibung Rotiert eine Geometrie um den Punkt `point`.

Beispiel
```
VAR.GEOMETRIE VAR.POINT 45.0 ICS.GEOMROT2 [geom]
```

Methode **ICS.GEOMROUND [g geom, r resolution][g geom]**

Beschreibung Rundet eine Geometrie auf die Resolution. Für Millimeter `resolution = 0.001` .

Beispiel
```
VAR.GEOMETRIE 0.001 ICS.GEOMROUND [geom]
```

Methode **ICS.STROKE [g geom, r tolerance][g geom]**

Beschreibung Löst alle Kreisbögen in einer Geometrie in Liniensegmente auf. Ein Kreisbogen wird gleichmässig in Liniensegemente aufgelöst, bis die Toleranz unterschritten ist. tolerance > 0.0 : Die Toleranz ergibt sich aus dem Verhältnis der Pfeilhöhe zum Radius eines Kreisbogens. tolerance = 0.0 : Der Kreisbogen wird in den Anfangs- und Endpunkt und dem Punkt auf dem Kreisbogen aufgelöst. tolerance < 0.0 : Die Toleranz ist ein absoluter Wert in Meter, der die maximale Pfeilhöhe definiert.

Beispiel
```
VAR.GEOMETRIE 0.1 ICS.STROKE [geom]
```

Methode **ICS.GET_LENGTH [l line][r length]**

Beschreibung Berechnet die Länge einer Linie. Die Linie kann Lines oder Arcs beinhalten. Die Länge wird nur planar -x/y- berechnet. Falls die Länge nicht berechnet werden kann, gibt die Methode NULL zurück.

Beispiel
```
VAR.LINE ICS.GET_LENGTH [123.456]
```

Methode **ICS.GET_LENGTH2 [g geometry][r length]**

Beschreibung Berechnet die Länge eines Punktes, Linie oder Fläche. Bei einem Punkt ist die Länge=0.0. Bei einer Fläche entspricht das Resultat der Summe der Längen aller Ränder. Linienelement können Lines oder Arcs beinhalten. Die Länge wird nur planar -x/y- berechnet. Falls die Länge nicht berechnet werden kann, gibt die Methode NULL zurück.

Beispiel
```
VAR.AREA ICS.GET_LENGTH2 [123.456]
```

Methode **ICS.GET_ARC_CENTER [l arc][p point]**

Beschreibung Berechnet den Zentrumspunkt eines Kreisbogens. Falls die Geometrie kein Arc ist, wird NULL zurückgegeben.

Beispiel
```
VAR.ARC ICS.GET_ARC_CENTER [point]
```

Methode **ICS.GET_ARC_RADIUS [l arc][r radius]**

Beschreibung Berechnet den Bogenradius eines Kreisbogens. Falls die Geometrie kein Arc ist, wird NULL zurückgegeben.

Beispiel
```
VAR.ARC ICS.GET_ARC_RADIUS [112.234]
```

Methode **ICS.EXTEND_LINE [l line, r distance][]**

Beschreibung Verlängert eine Linie am Anfang oder Ende. Mit einer positiven `distance` wird die Linie am Ende (letztes Segment) verlängert. Mit einer negative `distance` wird die Linie am Anfang (erstes Segment) verlängert. Die Input-Geometrie bleibt unverändert in folgenden Fällen: es ist keine Linie, das zu verlängernde Segment ist ein Kresibogen, das zu verlängernde Segment hat keine Ausdehnung.

Beispiel
```
&VAR.LINE 5.0 ICS.EXTEND_LINE []
```

Methode **ICS.CALC_POINT_AT [l line, r distance][r orientation, p point]**

Beschreibung Berechnet einen Punkt und die Richtung am Punkt entlang einer Linie vom Anfangspunkt mit der Distanz. Die Linie kann Lines oder Arcs beinhalten. Der Punkt wird nur planar -x/y- berechnet. Falls die Distanz länger als die Linienlänge ist, wird der letzte Punkt der Linie mit seiner Orientierung zurückgegeben.. Falls keine Berechnung durchgeführt werden kann, werden die Werte als NULL zurückgegeben.

Beispiel
```
VAR.LINE 10.0 ICS.CALC_POINT_AT [45.0, point]
```

Methode **ICS.CALC_POINT_AT2 [l line, r distance, r cross][r orientation, p point]**

Beschreibung Berechnet einen Punkt und die Richtung am Punkt entlang einer Linie vom Anfangspunkt mit der Distanz und der Querdistanz. Die Linie kann Lines oder Arcs beinhalten. Der Punkt wird nur planar -x/y- berechnet. Eine positive Querdistanz bedeutet rechts in Richtung der Linie. Eine negative Querdistanz bedeutet links in Richtung der Linie. Falls die Distanz länger als die Linienlänge ist, wird der letzte Punkt der Linie mit seiner Orientierung für die Berechnung herangezogen. Falls keine Berechnung durchgeführt werden kann, werden die Werte als NULL zurückgegeben.

Beispiel
```
VAR.LINE 10.0 2.0 ICS.CALC_POINT_AT2 [45.0, point]
```

Methode **ICS.PROJECT_POINT [l line, p point][r orientation, p point]**

Beschreibung Projiziert die kürzeste Distanz eines Punktes auf eine Linie und berechnet den Linienpunkt auf der Linie und dessen Richtung. Ergeben sich mehrere Lösungen mit denselben kürzesten Distanzen, so wird derjenige Linienpunkt berücksichtigt, der die kürzeste Distanz auf der Linie vom Anfangspunkt der Linie aufweist. Der Linienpunkt wird nur planar -x/y- berechnet.

Beispiel
```
VAR.LINE VAR.POINT ICS.PROJECT_POINT [45.0, point]
```

Methode **ICS.PROJECT_POINT2 [l line, p point][r distance, r length, r orientation, p point]**

Beschreibung Wie ICS.PROJECT_POINT. Zusätzlich werden als Argumente die Länge (Stationierung) auf der Linie bis zum Linienpunkt und der Querabstand vom Punkt zum Linienpunkt zurückgegeben. Eine positive Querdistanz bedeutet rechts in Richtung der Linie. Eine negative Querdistanz bedeutet links in Richtung der Linie.

Beispiel
```
VAR.LINE VAR.POINT ICS.PROJECT_POINT2 [-12.56, 123.12, 45.0, point]
```

Methode **ICS.PROJECT_POINT_PERPENDICULAR [l line, p point][r orientation, p point, b status]**

Beschreibung Projiziert einen Punkt als Lot rechtwinklig auf eine Linie und berechnet den Linienpunkt auf der Linie und dessen Richtung. Ergeben sich mehrere Lösungen mit denselben kürzesten Distanzen vom Punkt zum Linienpunkt, so wird derjenige Linienpunkt berücksichtigt, der die kürzeste Distanz auf der Linie vom Anfangspunkt der Linie aufweist. Der Linienpunkt wird nur planar -x/y- berechnet. Falls der Linienpunkt nicht berechnet werden kann, gibt die Methode nur den Status FALSE zurück.

Beispiel
```
VAR.LINE VAR.POINT ICS.PROJECT_POINT_PERPENDICULAR [45.0, point, TRUE]
```

Methode **ICS.PROJECT_POINT_PERPENDICULAR2 [l line, p point][r distance, r length, r orientation, p point, b status]**

Beschreibung Wie ICS.PROJECT_POINT_PERPENDICUALR. Zusätzlich werden als Argumente die Länge (Stationierung) auf der Linie bis zum Linienpunkt und der Querabstand vom Punkt zum Linienpunkt zurückgegeben. Eine positive Querdistanz bedeutet rechts in Richtung der Linie. Eine negative Querdistanz bedeutet links in Richtung der Linie. Falls der Linienpunkt nicht berechnet werden kann, gibt die Methode nur den Status FALSE zurück.

Beispiel
```
VAR.LINE VAR.POINT ICS.PROJECT_POINT_PERPENDICUALR2
[-12.56,123.12, 45.0, point, TRUE]
```

Methode **ICS.PROJECT_GEOM [g geometry1, g geometry2][r distance, p point2, p point1]**

Beschreibung Projiziert die kürzeste Distanz einer Geometrie 1 auf eine Geometrie2. Geometrien könne vom Typ `point`, `line` oder `area` sein. Die Methode liefert die kürzeste Distanz , den Punkt 1 auf der Geometrie 1 und den Punkt 2 auf der Geometrie. Der Berechnung erfolgt nur planar -x/y. Achtung: Wenn ein Kreisbogen der Geometrie 1 zu einem Kreisbogen der Geometrie 2 die kürzeste Distanz ergibt, so ist die Berechnung ungenau. Der Kreisbogenpunkt der einen Geometrie wird jeweils auf den Kreisbogen der anderen Geometrie projiziert.

Beispiel
```
VAR.AREA VAR.LINE ICS.PROJECT_GEOM [16.83, point2, point1]
```

Methode **ICS.GET_LINE_START_END_POINT [l line][p endpoint , p startpoint]**

Beschreibung Liefert zu einer Linie den Start-und Endpunkt der Linie.

Beispiel
```
VAR.LINE ICS.GET_LINE_START_END_POINT [point, point]
```

Methode **ICS.GET_LINE_START_END_TANGENT_ANGLE [l line][r endangle , r startangle]**

Beschreibung Liefert zu einer Linie den Start-und Endtangentwinkel der Linie.

Beispiel
```
VAR.LINE ICS.GET_LINE_START_END_POINT [45.0, 315.0]
```


Methode **ICS.COPY_LINE_PARALLEL [l line, r distance, s join, r miterlimit][l line]**

Beschreibung Kopiert eine Linie parallel um eine Distanz. Ein positiver Wert für `distance` bedeutet ein Kopieren rechts in Richtung der Linie. Ein negativer Wert für distance bedeutet ein Kopieren links in Richtung der Linie. Mit `join` wird der Verbindungstyp der Teilsegmente der kopierten Linie bestimmt. Mit `join='bevel'` werden die Teilsegmente mit einer Linie verbunden. Mit `join='miter'` werden die Teilsegmente verlängert, bis sie sich kreuzen, oder bis der Wert von `miterlimit` erreicht ist. Mit `join='round'` werden die Teilsegmente mit einem Kreisbogen verbunden.

Beispiel
```
VAR.LINE 0.25 'miter' 0.30 ICS.COPY_LINE_PARALLEL [line]
```

Methode **ICS.COPY_AREA_BOUNDARY_PARALLEL [a area, r distance, s join, r miterlimit][a area]**

Beschreibung Kopiert die Flächenbegrenzungslinien eine Fläche parallel um eine Distanz . Sind die Fächenbegrenzungslinien im Uhrzeigersinn gerichtet, so bedeutet ein positiver Wert für `distance` ein Kopieren nach innen der Fläche, ein negativer Wert für `distance` ein Kopieren nach aussen der Fläche. Sind die Fächenbegrenzungslinien gegen den Uhrzeigersinn gerichtet, so bedeutet ein positiver Wert für `distance` ein Kopieren nach aussen der Fläche, ein negativer Wert für `distance` ein Kopieren nach innen der Fläche. Die gewünschte Richtung den Fächenbegrenzungslinien kann mit den Methoden `ICS.SET_AREA_CLOCKWISE` und `ICS.SET_AREA_ANTICLOCKWISE` gesetzt werden. Mit `join` wird der Verbindungstyp der Teilsegmente der kopierten Linie bestimmt. Mit `join='bevel'` werden die Teilsegmente mit einer Linie verbunden. Mit `join='miter'` werden die Teilsegmente verlängert, bis sie sich kreuzen, oder bis der Wert von `miterlimit` erreicht ist. Mit `join='round'` werden die Teilsegmente mit einem Kreisbogen verbunden.

Beispiel
```
VAR.AREA 0.25 'miter' 0.30 ICS.COPY_AREA_BOUNDARY_PARALLEL [area]
```

Methode **ICS.CREATE_AREA_FROM_LINE[l line, r distance, s join, r miterlimit, s caps][a area]**

Beschreibung Erzeugt einen Linienbuffer (Fläche) um eine Linie durch das parallele Kopieren der Linie nach links und rechts und der Verbindung der kopierten Linien zu einer Fläche. Der Wert für distance gibt die Distanz zum parallelen Kopieren der Linien für die Fläche. Mit Join wird der Verbindungstyp der Teilsegmente der kopierten Linien bestimmt. Mit Join='bevel' werden die Teilsegmente mit einer Linie verbunden. Mit Join='miter' werden die Teilsegmente verlängert, bis sie sich kreuzen, oder bis der Wert von miterlimit erreicht ist. Mit Join='round' werden die Teilsegmente mit einem Kreisbogen verbunden. Mit Caps wird der Verbindungstyp an den Enden der kopierten Linien bestimmt. Mit Caps='bevel' werden die Enden mit einer Linie verbunden. Mit Caps='round' werden die Enden mit einem Kreisbogen verbunden.

Beispiel
```
VAR.LINE 0.25 'miter' 0.30 'round' ICS.CREATE_AREA_FROM_LINE [area]
```

Methode **ICS.EXTRACT_LINE_SEGMENT[l line, r distance, r length][l line]**

Beschreibung Extrahiert aus einer Linie ein Liniensegment ab der Distanz vom Startpunkt und mit der Länge .

Beispiel
```
VAR.LINE 100.0 10.0  ICS.EXTRACT_LINE_SEGMENT [line]
```


Methode **ICS.EXTRACT_LINE_PATTERN_SEGMENT[l line, s pattern][li geometry-list]**

Beschreibung Extrahiert aus einer Linie entsprechend dem Pattern Liniensegmente und gibt diese Geometrien in einer Liste zurück. Der 1. Eintrag im Pattern definiert die Länge des 1. Liniensegmentes. Der 2. Eintrag im Pattern definiert die Distanz vom Ende des 1. zum Anfang des 2. Lininesegment. Der 3. Eintrag im Pattern definiert die Länge des 2. Liniensegmentes. Der 4. Eintrag im Pattern definiert die Distanz vom Ende des 2. zum Anfang des 3. Lininesegment. Und so weiter bis zum Ende des Patterns, danach wiederholt sich das Pattern.

Beispiel

```
VAR.LINE '4.5/1.5/4.5/1.5/4.5/6.0'
ICS.EXTRACT_LINE_PATTERN_SEGMENT [list]
```

Methode **ICS.EXTRACT_LINE_PATTERN_POINTS [l line, s pattern, b rot][li geometry-list]**

Beschreibung Extrahiert aus einer Linie entsprechend dem Pattern Punkte und gibt diese Punkte in einer Liste zurück. Mit dem Boolean rot kann definiert werden, ob dir Rotation auf der Linie für einen Punkt in der z-Koordinate des Punktes zurückgegeben serden soll. Der 1. Eintrag im Pattern definiert die Länge bis zum 1. Punkt. Der 2. Eintrag im Pattern definiert die Länge vom vorherigen bis zum 2. Punkt. Der 3. Eintrag im Pattern definiert die Länge vom vorherigen bis zum 4. Punkt. Und so weiter bis zum Ende des Patterns, danach wiederholt sich das Pattern.

Beispiel

```
VAR.LINE '8.0/3.0/4.0/5.0' TRUE
ICS.EXTRACT_LINE_PATTERN_POINTS [list]
```

Methode **ICS.LINEARCS_TO_BEZIER [l line][li list]**

Beschreibung Approximiert Kreisbögen in einem Linienzug zu Bezier Kurven. Es wird eine Liste von Segmenten zurückgegeben. Ein Segment kann aus einer Geraden oder einer Bezier Kurve bestehen. Eine Gerade besteht aus einer Liste von 2 Punkten. Eine Bezier Kurve besteht aus einer Liste von 4 Stützpunkten. Der letzte Punkt eines Segmentes ist gleich dem ersten Punkt des nächsten Segmentes. Kreisbögen können in mehrer Bezier Kurven approximiert werden. Bezier Kurven werden zum Beispiel bei der Definition von Charctern/Symbolen in Fonts verwendet.

Beispiel

```
VAR.LINE ICS.LINEARCS_TO_BEZIER [list]
```

Methode **ICS.ARC_CIRCLE_PREPARE [l arc, r sweep-angle-limit][l line, b status]**

Beschreibung Eine Geometrie die exakt ein Arc-Segment enthält und fast einen Vollkreis bildet, wird in einen Linienzug mit zwei Arc-Segmenten umgewandelt. Mit `sweep-angle-limit` kann die Limite gesetzt werden. Überschreitet der Sweep-Winkel des Kreisbogens diese Limite, wird die Verarbeitung durchgeführt und die neue Geometrie wird zusammen mit dem `status TRUE` zurückgegeben. Diese Methode kann angewendet werden, wenn Kreisbögen in Systeme geschrieben werden sollen, die Probleme mit Kreisbögen haben die fast einen Vollkreis bilden.

Beispiel

```
VAR.LINE 359.00 ICS.ARC_CIRCLE_PREPARE [line, TRUE]
```

Methode **ICS.CREATE_CIRCLE [p point, r radius][l line]**

Beschreibung Kreiert aus einem Punkt als Kreiszentrum und einem Radius einen Kreis als Linie. Die Linie besteht aus zwei Kreisbögen. Falls der Kreis nicht kreiert werden kann, wird NULL zurückgegeben.

Beispiel
```
VAR.POINT 10.00 ICS.CREATE_CIRCLE [line]
```

Methode **ICS.HATCH_AREA [a area, s type, p origin, r angle , r space][li geometry-list]**

Beschreibung Schraffiert eine Fläche und gibt die Schraffurlinien als Liste von Linien-Geometrien zurück. Mit `type` wird der Schraffurtype definiert. Falls der Schraffurtype `NULL` oder unbekannt ist, gilt `Type='absolute'`. Mit `type 'absolute'` wird absolut vom Origin mit dem Winkel `angle` und der Distanz `space` schraffiert. Mit `type 'relative'` wird relativ zur längsten Begrenzunglinie der Fläche vom `origin` mit dem Winkel `angle` und der Distanz `space` schraffiert. `origin` definiert den Schraffurursprung. Falls `origin NULL` ist gilt der `origin 0.0/0.0`. `angle` definiert den Schraffurwinkel. `space` definiert die Schraffurdistanz zwischen den Linien.

Beispiel
```
VAR.AREA 'absolute' VAR.POINT 45.0 1.0 ICS.HATCH_AREA [list]
```

Methode **ICS.CROSSHATCH_AREA [a area, s type, p origin, r angle , r space, r angle2 , r space2][li geometry-list]**

Beschreibung Analog wie `ICS.HATCH_AREA` mit 2 zusätzlichen Argumenten `angle2` und `space2` , mit denen die Kreuzschraffur analog den Argumenten `angle` und `space` definiert wird.

Beispiel
```
VAR.AREA 'absolute' VAR.POINT 45.0 1.0 -45.0 1.0 ICS.CROSSHATCH_AREA [list]
```

Methode **ICS.PATTERN_AREA [a area, s type, p origin, r angle , r space-vertical, r offset][li geometry-list]**

Beschreibung Pattert eine Fläche und gibt die Pattern-Punkte als Liste von Punkt-Geometrien zurück. Mit `type` wird der Patterntype definiert. Falls der Patterntype `NULL` oder unbekannt ist, gilt `type 'absolute'`. Mit `type 'absolute'` wird absolut vom `origin` mit dem Winkel `angle` und der Distanz `space` gepattert. Mit `type 'relative'` wird relativ zur längsten Begrenzunglinie der Fläche vom Origin mit dem Winkel `angle` und der Distanz `space` gepattertt. `origin` definiert den Patternursprung. Falls `origin NULL` ist gilt der `origin 0.0/0.0`. `angle` definiert der Patternwinkel. `space-vertical` definiert die vertikale Patterndistanz zwischen den Punkten. `offset` definiert den horizontalen Offset eines Punktes auf jeder zweiten Pattern-Linie, relativ zum Patternwinkel.

Beispiel
```
VAR.AREA 'absolute' VAR.POINT 45.0 1.0 0.5 ICS.PATTERN_AREA [list]
```

Methode **ICS.PATTERN_AREA2 [a area, s type, p origin, r angle , r space-vertical , r space-horizontal, r offset, b random][li geometry-list]**

Beschreibung Analog wie `ICS.PATTERN_AREA` mit 2 zusätzlichen Argumenten `space-horizontal` und `random`. `space-horizontal` definiert die horizontale Patterndistanz zwischen den Punkten. `random` definiert, dass der Punkt zuffällig um den berechneten Punkt ermittelt werden soll.

Beispiel
```
VAR.AREA 'absolute' VAR.POINT 45.0 1.0 2.0 0.5 FALSE ICS.PATTERN_AREA2 [list]
```

Methode **ICS.SET_AREA_CLOCKWISE [a area][a area]**

Beschreibung Setzt den Drehsinn der Geometrien des äusseren Randes einer Fläche auf den Uhrzeigersinn. Setzt den Drehsinn der Geometrien der Ränder von Inseln der Fläche auf den Gegenuhrzeigersinn.

Beispiel
```
VAR.AREA ICS.SET_AREA_CLOCKWISE[area]
```

Methode **ICS.SET_AREA_ANTICLOCKWISE [a area][a area]**

Beschreibung Setzt den Drehsinn der Geometrien des äusseren Randes einer Fläche auf den Gegenuhrzeigersinn. Setzt den Drehsinn der Geometrien der Ränder von Inseln der Fläche auf den Uhrzeigersinn.

Beispiel

```
VAR.AREA ICS.SET_AREA_ANTICLOCKWISE[area]
```

Methode **ICS.GEOM_CLEAN [g geometry][g|li geometry or list, b status]**

Beschreibung Bereinigt eine Geometrie. Der Input kann ein Punkt, eine Linie oder eine Fläche sein. Die Anwendung der Methode ist zum Beispiel notwendig, wenn Geometrien in ein Geometrie-restrikives System geschrieben werden. Zum Beispiel zur Elimination von doppelten Punkten in einem Linienzug. Die Methode gibt einen Status zurück, ob die Geometrie berinigt wurde oder nicht. Status = FALSE , die Geometrie wurde nicht bereinigt, Status = TRUE, die Geometrie wurde bereinigt, die bereinigte Geometrie wird zusätzlich zrückgegeben. Die bereinigte Geometrie kann eine Geometrie, eine Liste von Geometrien oder NULL sein. Bei der Bereinigung erfolgt keine Umwandlung des Typs der Geometrie. Zum Beispiel wird eine nicht geschlossene Fläche nicht in eine Linie umgewandelt. Die Methode legt in der Map GEOM_CLEAN_STATUS Informationen über die Bereiningung der Geometrie ab. Mit ICS.GEOM_CLEAN_SYSTEM_SET kann das System definiert werden, nach dessen Krtiterien die Geometrie bereinigt werden soll . Mit ICS.GEOM_CLEAN_RESOLUTION_SET kann die Auflösung für die Bereiningung definiert werden. Zur funktionsweise siehe dazu mehr im Appendix.

Beispiel

```
VAR.GEOM ICS.GEOM_CLEAN [list, TRUE]
```

```
! Beispiel Anwendung
! ICS.GEOM_CLEAN
! ICS.GEOM_CLEAN_RESOLUTION_SET
! ICS.GEOM_CLEAN_STATUS_DESCR_GET

'ORACLE' GEOM_CLEAN_SYSTEM_SET
0.001 GEOM_CLEAN_RESOLUTION_SET

IF VAR.GEOM GEOM_CLEAN THEN
   => VAR.GEOM_CLEAN

   &GEOM_CLEAN_STATUS MAPRESET
   WHILE &GEOM_CLEAN_STATUS MAPSCAN DO
      TO_INT => VAR.STATUS
      => VAR.LIST

      VAR.STATUS GEOM_CLEAN_STATUS_DESCR_GET => VAR.DESCR

      DISPLAY VAR.DESCR
      &VAR.LIST RESET_READ
      WHILE &VAR.LIST READ_NEXT DO
         DISP
      END_WHILE
   END_WHILE
END_IF
```

Methode **ICS.GEOM_CLEAN_SYSTEM_SET [s system][]**

Beschreibung Setzt für die Methode ICS.GEOM_CLEAN das System, nach dessen Kriterien die Geometrie bereiningt werden soll. Werte sind ORACLE oder ESRI.

Beispiel
```
'ORACLE' ICS.GEOM_CLEAN_SYSTEM_SET []
```

Methode **ICS.GEOM_CLEAN_RESOLUTION_SET [r resolution][]**

Beschreibung Setzt für die Methode ICS.GEOM_CLEAN die Auflösung für die Bereinigung der Geometrie. Der Default ist 0.001.

Beispiel
```
0.001 ICS.GEOM_CLEAN_RESOLUTION_SET []
```

Methode **ICS.GEOM_CLEAN_STATUS_DESCR_GET [i status][s description]**

Beschreibung Die Methode ICS.GEOM_CLEAN legt in der Map GEOM_CLEAN_STATUS Informationen über die Bereiningung der Geometrie ab. Die Map beinhaltet diverse Status-Codes mit jeweils einer Liste von Geometrien, die Bereinigungen betreffen. Mit dieser Funktion kann die Beschreibung zu einem Status ermittelt werden.

Beispiel
```
23 ICS.GEOM_CLEAN_STATUS_DESCR_GET ["STRAIGHT_ARC"]
```

Methode **ICS.LINE_INTERSECTION [l line1, l line2][li list of points, b status]**

Beschreibung Berechnet den Linienverschnitt zweier Linien. Konnten Punkte berechnet werden, bringt die Methode den Status TRUE und einen Liste der Punkte zurück. Konnten keine Punkte berechnet werden, bringt die Methode den Status FALSE ohne Liste zurück. Die Liste der Punkte beinhaltet Punkte, in denen beide Input-Linien sich schneiden oder berühren.

Beispiel
```
VAR.LINE1 VAR.LINE2 ICS.LINE_INTERSECTION [list, TRUE]
```

Methode **ICS.AREA_INTERSECTION [a area1, a area2][li list of areas, b status]**

Beschreibung Berechnet den Flächenverschnitt zweier Flächen mit dem boolschen Operator AND. Konnten Flächen berechnet werden, bringt die Methode den Status TRUE und einen Liste der Flächen zurück. Konnten keine Flächen berechnet werden, bringt die Methode den Status FALSE ohne Liste zurück. Die Liste der Flächen beinhaltet Flächen, in denen beide Input-Flächen vorkommen.

Beispiel
```
VAR.AREA1 VAR.AREA2 ICS.AREA_INTERSECTION [list, TRUE]
```

Methode **ICS.AREA_DIFFERENCE [a area1, a area2][li list of areas, b status]**

Beschreibung Berechnet den Flächenverschnitt zweier Flächen mit dem boolschen Operator XOR. Konnten Flächen berechnet werden, bringt die Methode den Status TRUE und einen Liste der Flächen zurück. Konnten keine Flächen berechnet werden, bringt die Methode den Status FALSE ohne Liste zurück. Die Liste der Flächen beinhaltet Flächen, in denen jeweils nur eine der beiden Input-Flächen vorkommt.

Beispiel
```
VAR.AREA1 VAR.AREA2 ICS.AREA_DIFFERENCE [list, TRUE]
```

Methode **ICS.AREA_UNION [a area1, a area2][li list of areas, b status]**

Beschreibung Berechnet den Flächenverschnitt zweier Flächen mit dem boolschen Operator OR. Konnten Flächen berechnet werden, bringt die Methode den Status TRUE und einen Liste der Flächen zurück. Konnten keine Flächen berechnet werden, bringt die Methode den Status FALSE ohne Liste zurück. Die Liste der Flächen beinhaltet Flächen, in denen eine oder beide der Input-Flächen vorkommen. Eine erfolgreiche Berechnung ergibt immer nur

eine Fläche. Diese Fläche wird wie in den anderen boolschen Flächenverschnitten in einer Liste zurückgegeben.

Beispiel
```
VAR.AREA1 VAR.AREA2 ICS.AREA_UNION [list, TRUE]
```

Methode **ICS.GET_AREA_LINE_LIST [a area][li list]**

Beschreibung Liest alle Linien aller Ränder einer Fläche und bringt diese Linien als Liste zurück.

Beispiel
```
VAR.AREA ICS.GET_AREA_LINE_LIST[list]
```

Methode **ICS.CALC_BOUNDING_BOX [g geometrie][p pointmax, p pointmin]**

Beschreibung Berechnet zu einer Geometrie point, line oder area die maximale 2D-Ausdehung. Die Methode liefert auf dem Stack die minimalen ind maximalen Koordinaten der Ausdehnung je als point.

Beispiel
```
VAR.AREA ICS.CALC_BOUNDING_BOX [pointmax, pointmin]
```

Methode **ICS.CALC_BOUNDING_BOX3D [g geometrie][p pointmax, p pointmin]**

Beschreibung Berechnet zu einer Geometrie point, line oder area die maximale 3D-Ausdehung. Die Methode liefert auf dem Stack die minimalen ind maximalen Koordinaten der Ausdehnung je als point.

Beispiel
```
VAR.AREA ICS.CALC_BOUNDING_BOX3D [pointmax, pointmin]
```

Methode **ICS.VADD [g geometrie, p vector][g geometrie]**

Beschreibung Verschiebt eine Geometrie um einen Vektor.

Beispiel
```
VAR.GEOM 10.0 10.0 10.0 TO_POINT ICS.VADD [geometrie]
```

Methode **ICS.VDIFF [p point1, p point2][p vector]**

Beschreibung Berechnet den Vektor als Punkt von `point2` zu `point1`.

Beispiel
```
VAR.POINT1 VAR.POINT2 ICS.VDIFF [point]
```

Methode **ICS.VSCAL [g geometrie, r scale][g geometrie]**

Beschreibung Skaliert eine Geometrie um den Nullpunkt.

Beispiel
```
VAR.GEOM 2.0 ICS.VSCALE [geometrie]
```

Methode **ICS.AZI [g point|line][r azimut]**

Beschreibung Berechnet für einen Punkt oder eine Linie das Azimut. Bei einem Punkt relativ zum Nullpunkt. Bei einer Linien vom Endpunkt zum Startpunkt.

Beispiel
```
VAR.GEOM ICS.AZI [real]
```

Methode **ICS.GEOMSET2D [g geometrie][g geometrie]**

Beschreibung Transferiert eine 3D-Geometrie in eine 2D-Geometrie. Ist die Geometrie bereits eine 2D-Geometrie, bleibt diese unverändert.

Beispiel
```
VAR.GEOM ICS.GEOMSET2D [geometrie]
```

Methode **ICS.GEOMSET3D [g geometrie, r z, b zkeep][g geometrie]**

Beschreibung Transferiert eine 2D-Geometrie in eine 3D-Geometrie mit der Z-Koordinate z. Ist die Geometrie bereits eine 3D-Geometrie, so kann mit zkeep=TRUE definiert werden, dass die z-Koordinate beibehalten wird, oder mit zkeep=FALSE, dass die z-Koordinate mit z überschrieben wird

Beispiel
```
VAR.GEOM 100.0 TRUE ICS.GEOMSET3D [geometrie]
```

Methode **ICS.GEOMIS2D [g geometrie][b is2D]**

Beschreibung Ermittelt ob alle Punkte einer Geometrie 2D-Koordinaten aufweisen. Das heisst auch, es kommen keine 3D-Koordinaten vor.

Beispiel
```
VAR.GEOM ICS.GEOMIS2D [TRUE]
```

Methode **ICS.GEOMIS3D [g geometrie][b is3D]**

Beschreibung Ermittelt ob alle Punkte einer Geometrie 3D-Koordinaten aufweisen.

Beispiel
```
VAR.GEOM ICS.GEOMIS3D [TRUE]
```

6.12. Datum/Zeit Methoden

Methode **ICS.GET_TIME [][i <HHMMSS>]**

Beschreibung Liefert die aktuelle Systemzeit im Format HHMMSS als Integer.

Beispiel
```
GET_TIME [200503] ! 20 Uhr 5 Minuten und 3 Sekunden
```

Methode **ICS.GET_DATE [][i <YYYYMMDD>]**

Beschreibung Liefert das aktuelle Systemdatum im Format YYYYMMDD als Integer .

Beispiel
```
GET_DATE [19970929] ! 29. September 1997
```

Methode **ICS.GET_DAYNAME [s <language>][s <day>]**

Beschreibung Liefert den Namen des Tages des aktuellen Systemdatum. Als <language> werden folgende Codes unterstützt: DE|FR|IT|EN .

Beispiel
```
'DE' GET_DAYNAME ['Freitag']
```

Methode **ICS.DATE_TO_DAY [i <yyyymmdd>][i <day>]**

Beschreibung Liefert von einem Date den Wochentag als Integer 0..6 (Sonntag..Samstag).

Beispiel
```
19690721 DATE_TO_DAY [1]
```

Methode **ICS.INT_TO_DAYNAME [i <day>,s <language>][s <day>]**

Beschreibung Konvertiert einen Tag als Integer in den Namen des Tages. Als <day> als Integer werden folgende Werte berücksichtigt 0..6 (Sonntag..Samstag). Als <language> werden folgende Codes unterstützt: DE|FR|IT|EN .

Beispiel
```
1 'DE' INT_TO_DAYNAME ['Montag']
```

Methode **ICS.GET_MONTHNAME [s <language>][s <month>]**

Beschreibung Liefert den Namen des Monats des aktuellen Systemdatum. Als <language> werden folgende Codes unterstützt: DE|FR|IT|EN .

Beispiel
```
'DE' GET_MONTHNAME ['August']
```

Methode **ICS.DATE_TO_MONTH [i <yyyymmdd>][i <month>]**

Beschreibung Liefert von einem Date den Monat als Integer 1..12 (Januar..Dezember).

Beispiel
```
 19690721 DATE_TO_MONTH [9]
```

Methode **ICS.INT_TO_MONTHNAME [i <month>,s <language>][s <month>]**

Beschreibung Konvertiert einen Monat als Integer in den Namen des Monats. Als <month> als Integer werden folgende Werte berücksichtigt 1..12 (Januar..Dezember). Als <language> werden folgende Codes unterstützt: DE|FR|IT|EN .

Beispiel
```
7 'DE' INT_TO_MONTHNAME ['Juli']
```

Methode **ICS.GET_SECONDS [][i <seconds>]**

Beschreibung Liefert die Sekunden seit dem Datum 1.1.1970 .

Beispiel
```
GET_SECONDS [1259672203]
```

Methode **ICS.SECONDS_TO_HOURS [i <seconds>][i <HHMMSS>]**

Beschreibung Rechnet Sekunden in Stunden im Format HHMMSS als Integer um.

Beispiel
```
131088 SECONDS_TO_HOURS [362448]
```

Methode **ICS.HOURS_TO_SECONDS [i <HHMMSS>][i <seconds>]**

Beschreibung Rechnet Stunden im Format HHMMSS als Integer in Sekunden um.

Beispiel
```
362448 SECONDS_TO_HOURS [131088]
```

Methode **ICS.DATE_DIFF [i <YYYYMMDD>, i <YYYYMMDD>][i <days>]**

Beschreibung Berechnet die Anzahl Tage zwischen zwei Dati im Format YYYYMMDD als Integer.

Beispiel
```
20090101 20091231 DATE_DIFF [364]
```

Methode **ICS.TIME_DIFF [i <HHMMSS>, i <HHMMSS>][i <HHMMSS>]**

Beschreibung Berechnet die Zeit zwischen zwei Zeiten. Alle Zeiten im Format HHMMSS als Integer.

Beispiel
```
080000 173045 TIME_DIFF [93045]
```

Methode **ICS.DATE_TIME_DIFF [i <YYYYMMDD> i <HHMMSS>, i <YYYYMMDD> i <HHMMSS>][i <HHMMSS>]**

Beschreibung Berechnet die Zeit zwischen zwei Dati mit Zeiten. Alle Dati im Format YYYYMMDD als Integer.. Alle Zeiten im Format HHMMSS als Integer.

Beispiel
```
20080228 180000 20080301 183045 DATE_TIME_DIFF [483045]
```

Methode **ICS.TIME_ADD [i <HHMMSS>, i <HHMMSS>][i <HHMMSS>]**

Beschreibung Addiert zwei Zeiten. Alle Zeiten im Format HHMMSS als Integer.

Beispiel
```
183030 023030 TIME_ADD [210100]
```

Methode **ICS.TIME_MINUS [i <HHMMSS>, i <HHMMSS>][i <HHMMSS>]**

Beschreibung Subtrahiert zwei Zeiten. Alle Zeiten im Format HHMMSS als Integer.

Beispiel
```
210100 023030 TIME_MINUS [183030]
```

Methode **ICS.SECONDS_TO_DATE_TIME [i <seconds>][i <HHMMSS>, i <YYYYMMSS>]**

Beschreibung Konvertiert Sekunden in ein Datum im Format YYYYMMDD als Integer und in eine Zeit im Format HHMMSS als Integer. Die Sekunden entsprechen den Anzahl Sekunden seit dem 1.1.1970 ermittelt mit `ICS.GET_SECONDS`.

Beispiel
```
GET_SECONDS SECONDS_TO_DATE_TIME [152329, 20091201]
```

Methode **ICS.DATE_TIME_TO_SECONDS [i <YYYYMMSS>, i <HHMMSS>][i <seconds>]**

Beschreibung Konvertiert ein Datum im Format YYYYMMDD als Integer und eine Zeit im Format HHMMSS als Integer in Sekunden. Die Sekunden entsprechen den Anzahl Sekunden seit dem 1.1.1970 ermittelt wie mit `ICS.GET_SECONDS`.

Beispiel
```
20091201 152329 DATE_TIME_TO_SECONDS [1259677409]
```

Methode **ICS.TIME_TO_STRING [i <HHMMSS>][s <HH.MM.SS>]**

Beschreibung Konvertiert eine Zeit im Format HHMMSS als Integer in einen String im Format 'HH:MM:SS'.

Beispiel
```
183045 TIME_TO_STRING ['18:30:45']
```

Methode **ICS.STRING_TO_TIME [s <'HH:MM:SS'>][i <HHMMSS>]**

Beschreibung Konvertiert eine Zeit im Format 'HH:MM:SS' als String in einen Integer im Format HHMMSS.

Beispiel
```
'18:30:45' STRING_TO_TIME [183045]
```

Methode **ICS.DATE_TO_STRING [i <YYYYMMSS>][s <DD.MM.YYYY>]**

Beschreibung Konvertiert ein Datum im Format YYYYMMDD als Integer in einen String im Format 'DD.MM.YYYY'.

Beispiel
```
20091231 TIME_TO_STRING ['31.12.2009']
```

Methode **ICS.STRING_TO_DATE [s <'DD.MM.YYYY'>][i <YYYYMMDD>]**

Beschreibung Konvertiert ein dATUM im Format 'DD.MM.YYYYY' als String in einen Integer im Format YYYYMMDD.

Beispiel
```
'31.12.2009' STRING_TO_DATE [20091231]
```

Methode **ICS.DATE_TO_STRING2 [i <YYYYMMSS> s <language>][s <DD. month YYYY>]**

Beschreibung Konvertiert ein Datum im Format YYYYMMDD als Integer in einen String im Format 'DD. month YYYY'. Als `<language>` werden folgende Codes unterstützt: `DE|FR|IT|EN`

Beispiel
```
20091231 'DE' TIME_TO_STRING2 ['31. Dezember 2009']
```

6.13. Spezielle Methoden

Methode **ICS.HALT [][]**

Beschreibung Bricht die Ausführung des aktuellen Skripts sofort ab.

Beispiel

```
ICS.HALT []
```

Methode **ICS.CALL [s prozedurname][]**

Beschreibung Ruft die Prozedur <prozedurname> auf. Der Prozedurname muss als String übergeben werden. Falls die Prozedur nicht existiert, wird der Skript abgebrochen.

Beispiel

```
'MYPROC' ICS.CALL []
```

Methode **ICS.CALLED_BY [s prozedurname][b status]**

Beschreibung Testet ob eine Prozedur von einer anderen Prozedur <prozedurename> aufgerufen wurde.

Beispiel

```
PROCEDURE TEST
   IF 'A' CALLED_BY THEN
      DISPLAY 'TEST was called by A'
   ELSIF 'B' CALLED_BY THEN
      DISPLAY 'TEST was called by B'
   END_IF
END_PROCEDURE

PROCEDURE A
  TEST
END_PROCEDURE

PROCEDURE B
   TEST
END_PROCEDURE

A B
```

Methode **ICS.OSCALL [s befehl][i status]**

Beschreibung Führt den Betriebsystembefehl <befehl> aus. Der Skriptinterpreter wartet auf die Beendigung des Befehls. In <status> wird der Beendigungsstatus des Betriebssystembefehls zurückgegeben (0 = erfolgreiche Durchführung).

Beispiel

```
IF 'dir c:\' ICS.OSCALL <> 0 THEN
   DISPLAY 'der Befehl konnte nicht durchgeführt werden'
END_IF
```

Methode **ICS.GETENV [s <var>][s <value>,b <status>]**

Beschreibung Liefert den Wert der Umgebungsvariablen <var> in <value> zurück. Falls die Umgebungsvariable existiert, wird für den Status TRUE zurückgegeben.

Beispiel

```
IF 'SystemRoot' ICS.GETENV THEN
   DISPLAY 'SystemRoot=',$
ELSE
   DISPLAY 'SystemRoot existiert nicht'
END_IF
```

Methode **ICS.GET_WORKING_DIRECTORY [][s <dir>]**

Beschreibung Liefert das aktuelle Arbeitsverzeichnis. (Siehe auch Methoden der Klasse DIRECTORY).

Beispiel

```
GET_WORKING_DIRECTORY ['c:\iltools\data']
```

Methode **ICS.GET_COMPUTERNAME [][s <name>]**

Beschreibung Liefert den aktuellen Computernamen.

Beispiel
```
GET_COMPUTERNAME ['server']
```

Methode **ICS.GET_SYSTEMLANGUAGE [][s <language>]**

Beschreibung Liefert die aktuelle Systemsprache.

Beispiel
```
GET_SYSTEMLANGUAGE ['Deutsch (Schweiz)']
```

Methode **ICS.GET_SYSTEMLANGUAGECODE [][s <languagecode>]**

Beschreibung Liefert den 2-stelligen ISO-Code der aktuellen Systemsprache.

Beispiel
```
GET_SYSTEMLANGUAGECODE ['DE']
```

Methode **ICS.GET_USERNAME [][s <name>]**

Beschreibung Liefert den aktuellen Benutzernamen.

Beispiel
```
GET_USERNAME ['Administrator']
```

Methode **ICS.INCR_ERROR_COUNT [][]**

Beschreibung Erhöht den aktuellen Fehlerzähler um 1. Bemerkung: Von `ERROR` wird automatisch `ICS.INCR_ERROR_COUNT` aufgefrufen.

Beispiel
```
IF IN.Name IS_NULL THEN
   DISPLAY 'name is not defined'
   INCR_ERROR_COUNT
END_IF
```

Methode **ICS.GET_ERROR_COUNT [][i count]**

Beschreibung Fragt den aktuellen Fehlerzähler ab.

Beispiel
```
IF GET_ERROR_COUNT = 0 THEN
   DISPLAY 'script terminated with no errors'
END_IF
```

Methode **ICS.SET_ERROR_COUNT [i count][]**

Beschreibung Setzt den aktuellen Fehlerzähler auf `<count>`.

Beispiel
```
99 SET_ERROR_COUNT
```

Methode **ICS.GET_GUID [][s guid]**

Beschreibung Liefert einen GUID (Globally Unique Identifier), eine eindeutige 128-Bit Nummer welche vom Windows-System generiert wird. Der GUID wird als Hexadezimal-String mit der Länge von 38 Zeichen inklusive 6 Format-Zeichen zurückgegeben.

Beispiel
```
GET_GUID ['{C771D8CB-E6D5-4DC2-9B92-76F8ED982799}']
```

Methode **ICS.GET_GUID_RAW [][s guid]**

Beschreibung Liefert einen GUID (Globally Unique Identifier), eine eindeutige 128-Bit Nummer welche vom Windows-System generiert wird. Der GUID wird als Hexadezimal-String mit der Länge von 32 Zeichen ohne Format-Zeichen zurückgegeben.

Beispiel
```
GET_GUID_RAW ['C771D8CBE6D54DC29B9276F8ED982799']
```

Methode **ICS.GET_STRING_MD5 [s string][s md5]**

Beschreibung Liefert für einen String einen eindeutigen Schlüssel als String. Der Schlüssel entspricht dem MD5-128-Bit-Hashwert in Hexadezimaler Schreibform (MD5 = Message-Digest Algorithm 5). Falls der Schlüssel nicht berechnet werden kann, liefert die Methode NULL zurück (z.B. bein einem Input <> string).

Beispiel
```
'Hello World' GET_STRING_MD5 ['B10A8DB164E0754105B7A99BE72E3FE5']
```

Methode **ICS.GET_FILE_MD5 [s file][s md5]**

Beschreibung Liefert für ein File einen eindeutigen Schlüssel als String. Der Schlüssel entspricht dem MD5-128-Bit-Hashwert in Hexadezimaler Schreibform (MD5 = Message-Digest Algorithm 5). Falls der Schlüssel nicht berechnet werden kann, liefert die Methode NULL zurück (z.B. das File existiert nicht). Die Methode kann eingesetzt werden, um festzustellen, ob zwei Files identisch sind. Liefert die Methode unterschiedliche Schlüssel, so sind die Files nicht identisch.

Beispiel
```
'c:\iltools\data\examples\DM01AVCH24D.itf' GET_FILE_MD5 ['6100C2FA5842BDE3BCEF16B0
```

Methode **ICS.EXPAND_ICS_PATH [s file][s file, b status]**

Beschreibung Expandiert einen relativen Pfad eines Files zum vollen Pfad in der ICS-Struktur. Sucht das File nach folgender Reihenfolge: 1. absoluter Pfad, 2. Pfad unter `ILTOOLS_DIR\user`, 3. Pfad unter `ILTOOLS_DIR\system`, 4. Pfad unter `ILTOOLS_DIR`, 5. Pfad wie Hauptkonfiguration, 6. Pfad wie Hauptkonfiguration aber unter `ILTOOLS_DIR\system`. Wird das File nicht gefunden, wird als Status `FALSE` zurückgegeben.

Beispiel
```
'\models\Grunddatensatz.ili' EXPAND_ICS_PATH ['c:\iltools\system\models\Grunddater
```

Methode **ICS.INCLUDE [s script][b status]**

Beschreibung Inkludiert ein Script-File. Analg der Direktive `|INCL` . Mit dieser Methode können dynamisch Scripts geladen werden.

Beispiel
```
'\script\util.lib' INCLUDE [TRUE]
```

Methode **ICS.INCLUDES_DISPLAY [][]**

Beschreibung Displays alle inkludierten Scripts.

Beispiel
```
INCLUDES_DISPLAY
```

Methode **ICS.INCLUDES_GET [][li list]**

Beschreibung Gibt alle inkludierten Scripts als Liste zurück.

Beispiel
```
INCLUDES_GET [list]
```

Methode **ICS.SCRIPT_GET [][s script]**

Beschreibung Liefert das aktuelle Script.

Beispiel
```
SCRIPT_GET [script]
```

Methode **ICS.SET_NEXT_SCRIPT [s script][]**

Beschreibung Setzt ein Skript, das nach dem Ausführen des aktuellen Skripts ausgeführt werden soll. Durch den mehrmaligen Aufruf dieser Methode können auch weitere Skripts zur Ausführung gesetzt werden. Die Skripts werden in der Reihenfolge ausgeführt, wie dieses gesetzt wurden. Das zuerst gesetzte Script wird zuerst ausgeführt. Das zuletzt gesetzte Script wird zuletzt ausgeführt.

Beispiel
```
'\script\myscripts\myscript.cfg' SET_NEXT_SCRIPT
```

Methode **ICS.SET_NEXT_SCRIPT_OPTION [s option, s value][]**

Beschreibung Zu einem mit ICS.SET_NEXT_SCRIPT gesetzten Script können nachfolgend mit dieser Methode Optionen für das Script gesetzt werden.

Beispiel
```
'input' OPT.output SET_NEXT_SCRIPT_OPTION
```

Methode **ICS.GET_MAIN_PROGRAM [][s program]**

Beschreibung Gibt das aktuelle Hauptprogramm mit dem Pfad zurück, ..\icsw.exe oder ..\ics.exe .

Beispiel
```
GET_MAIN_PROGRAM ['c:\iltools\system\bin\icsw.exe']
```

Methode **ICS.IS_BATCH_PROGRAM [][b status]**

Beschreibung Gibt zurück, ob das aktuelle Hauptprogram ics.exe ist. ics.exe wird für Batch-Prozesse eingesetzt.

Beispiel
```
IS_BATCH_PROGRAM [TRUE]
```

7. Klasse ICSCPU

7.1. Allgemeines

ICSCPU ist eine spezielle Klasse, welche Operationen auf der eingebauten Software CPU erlaubt. Die Software CPU ist die "Maschine", welche die Instruktionen (Bytecode) eines ICS Skripts ausführt.

7.2. Stack

Methode **ICSCPU.ASSERT_STACK_EMPTY[][]**

Beschreibung Stellt sicher, dass keine Operanden mehr auf dem Stack vorhanden sind. Falls dies dennoch der Falls sein sollte, wird der Skript mit einer Fehlermeldung abgebrochen.

Beispiel
```
'hello, world' ICSCPU.ASSERT_STACK_EMPTY ! Programmabbruch !!!
```

Methode **ICSCPU.DISPLAY_STACK[][]**

Beschreibung Zeigt den aktuellen Ihnalt des Stacks an.

Beispiel
```
'hello, world' ICSCPU.DISPLAY_STACK
```

Methode **ICSCPU.DISPLAY_PSTACK[][]**

Beschreibung Zeigt den aktuellen Ihnalt des Prozedurstacks an.

Beispiel	`ICSCPU.DISPLAY_PSTACK`

7.3. Diverses

Methode	**`ICSCPU.DISPLAY_IMEM[][]`**
Beschreibung	Zeigt den Inhalt des Instruktionsspeichers an (Bytecode).
Beispiel	`ICSCPU.DISPLAY_IMEM`

Methode	**`ICSCPU.DISPLAY_SOURCE_LINE[][]`**
Beschreibung	Zeigt die Zeilnenummer der aktuellen Instruktion an.
Beispiel	`ICSCPU.DISPLAY_SOURCE_LINE`

8. Klasse ICSIO

8.1. Allgemeines

Mit der Klasse ICSIO können BLOB's (= Binary Large Object) in eine binäre Datei geschrieben werden.

8.2. Erzeugen und Schliessen

Methode	**`ICSCIO.CREATE_BINARY [s fname][i handle,b state]`**
Beschreibung	Erzeugt eine Binärdatei mit Namen `<fname>`. Falls die Datei erzeugt werden kann, wird `TRUE` und ein Dateihandle sonst `FALSE` zurück gegeben.

Beispiel	`IF 'test.dat' ICSIO.CREATE_BINARY NOT THEN`
	`  ERROR 'unable to create binary file'`
	`  HALT`
	`END_IF => VAR.F`

Methode	**`ICSIO.CLOSE[i handle][]`**
Beschreibung	Schliesst eine mit `ICSIO.CREATE_BINARY` erzeugte Datei.
Beispiel	`VAR.F ICSIO.CLOSE ! []`

8.3. Schreiben

Methode	**`ICSIO.WRITE_BLOB [i handle,bl blob][b state]`**
Beschreibung	Schreibt den BLOB `<blob>` in die binäre Datei.
Beispiel	`VAR.F &VAR.BLOB ICSIO.WRITE_BLOB ! [TRUE]`

9. Klasse ICSRUN

9.1. Allgemeines

Die Klasse ICSRUN implementiert den allgemeinen INTERLIS Tools Datenfluss (s.a. INTERLIS Tools Konfigurationshandbuch). Die Klasse ICSRUN muss nicht mit |LOAD geladen werden.

9.2. RUN Methoden

Methode `ICSRUN.RUN1[][i status]`

Beschreibung Führt den durch die INOUT Map definierten Abbildungsalgorithmus aus. Der Algorithmus ist im Detail im INTERLIS Tools Benutzerhandbuch beschrieben. Falls keine Fehler in der INOUT Map vorhanden sind, gibt ICSRUN.RUN 0 als Wert zurück, sonst einen Wert <> 0.

Beispiel

```
! Abbildungsfunktion starten
IF ICSRUN.RUN1 <> 0 THEN
   ERROR 'INOUT contains errors'
   HALT
END_IF
```

9.3. Parameterübergabe

Methode `ICSRUN.GET_PARAM[][o object]`

Beschreibung ICSRUN.GET_PARAM kann in Abbildungsprozeduren für die Übernahme von Parameterwerten benutzt werden.

Beispiel

```
ICSRUN.GET_PARAM ! ['hello']
```

Methode `ICSRUN.GET_SPARAM[][s string]`

Beschreibung Analog wie ICSRUN.GET_PARAM. Es wird aber zusätzlich auf Datentyp String getestet.

Beispiel

```
ICSRUN.GET_SPARAM ! ['hello']
```

Methode `ICSRUN.GET_IPARAM[][i integer]`

Beschreibung Analog wie ICSRUN.GET_PARAM. Es wird aber zusätzlich auf Datentyp Integer getestet.

Beispiel

```
ICSRUN.GET_IPARAM ! [25]
```

Methode `ICSRUN.GET_RPARAM[][r real]`

Beschreibung Analog wie ICSRUN.GET_PARAM. Es wird aber zusätzlich auf Datentyp Real getestet.

Beispiel

```
ICSRUN.GET_RPARAM ! [0.25]
```

Methode `ICSRUN.GET_PPARAM[][p point]`

Beschreibung Analog wie ICSRUN.GET_PARAM. Es wird aber zusätzlich auf Datentyp Point getestet.

Beispiel

```
ICSRUN.GET_PPARAM ! [15.0/5.0]
```

Methode	**ICSRUN.GET_LPARAM[][l line]**
Beschreibung	Analog wie ICSRUN.GET_PARAM. Es wird aber zusätzlich auf Datentyp Line getestet.
Beispiel	`ICSRUN.GET_LPARAM ! [line]`

Methode	**ICSRUN.GET_APARAM[][s area]**
Beschreibung	Analog wie ICSRUN.GET_PARAM. Es wird aber zusätzlich auf Datentyp Area (Fläche) getestet.
Beispiel	`ICSRUN.GET_APARAM ! [area]`

Methode	**ICSRUN.SET_PARAM[o object][]**
Beschreibung	Setzt einen Parameter für den späteren Aufruf einer Abbildungsprozedur.

Beispiel	`'hello' ICSRUN.SET_PARAM`
	`'world' ICSRUN.SET_PARAM`
	`EXAMPLEMODULE_CONCAT2`

9.4. Diverses

Methode	**ICSSRUN.ABORT_RULE[][]**
Beschreibung	Bricht die aktuelle Abbildungsregel ab. Der Skript wird jedoch nicht abgebochen.
Beispiel	`ICSRUN.ABORT_RULE`

Methode	**ICSRUN.GET_RULE[][s rule]**
Beschreibung	Liefert die aktuelle Abbildungsregel.
Beispiel	`ICSRUN.GET_RULE ['EXAMPLEMODULE_CONCAT2,hello,world']`

Methode	**ICSRUN.SET_RULE[s rule][]**
Beschreibung	Macht <rule> zur aktuellen Abbildungsregel.
Beispiel	`'EXAMPLEMODULE_CONCAT2,hello,world' ICSRUN.SET_RULE`

Methode	**ICSSRUN.APPEND_TO_RULE[s rule][]**
Beschreibung	Erweitert die aktuelle Abbildungsregel um <rule>.
Beispiel	`'IN.Art' ICSRUN.APPEND_TO_RULE`

10. Klasse MESSAGE

10.1. Allgemeines

Mit den Methoden der Klasse MESSAGE können Meldungen in eine Logdatei oder auf die Konsole geschrieben werden. Die Klasse MESSAGE muss nicht mit |LOAD geladen werden. Es werden folgende Meldungen unterschieden:

DISPLAY

Normale Meldungen. Die Meldungen werden normalerweise in die Datei IL-TOOLS_DIR\temp\ics.log geschrieben.

ERROR

Fehlermeldungen. Die Meldungen werden normerweise in die Datei IL-TOOLS_DIR\temp\ics.log geschrieben. Jede Meldung erhält den Prefix *** ERROR ***.

STATUS

Statusmeldungen. Statusmeldungen sollen über den Fortschritt der aktuellen Bearbeitung orientieren. Statusmeldungen werden in die Logdatei und in die Statuszeile des Dialogfensters (falls vorhanden) geschrieben.

10.2. Meldungen ausgeben

Methode	**`MESSAGE.DISPLAY[s message][]`**
Beschreibung	Gibt die Meldung <message> in die Logdatei aus. MESSAGE.DISPLAY wird auch vom Skriptinterpreter für jegliche Ausgaben benutzt. Wenn diese Methode durch eine andere ICS Methode oder Prozedur überschrieben wird kann der Output umgelenkt oder anderst verarbeitet werden.
Beispiel	`'hello, world' MESSAGE.DISPLAY`

Methode	**`MESSAGE.ERROR[s error][]`**
Beschreibung	Gibt die Meldung <error> als Fehlermeldung aus. MESSAGE.ERROR wird auch vom Skriptinterpreter für jegliche Fehlermeldungen benutzt. Wenn diese Methode durch eine andere ICS Methode oder Prozedur überschrieben wird, können Fehlermeldungen umgelenkt, unterdrückt oder speziell verarbeitet werden.
Beispiel	`'unable to find file' MESSAGE.ERROR`

Methode	**`MESSAGE.STATUS[s status][]`**
Beschreibung	Gibt eine Statusmeldung aus. MESSAGE.STATUS wird auch vom Skriptinterpreter für jegliche Statusmeldungen benutzt. Wenn diese Methode durch eine andere ICS Methode oder Prozedur überschrieben wird, können Statusmeldungen umgelenkt, unterdrückt oder speziell verarbeitet werden.
Beispiel	`'pricessing file input.txt ...' MESSAGE.STATUS`

10.3. Logdatei umlenken / Output unterdücken

Methode	**`MESSAGE.SET_LOG[s file][]`**
Beschreibung	Setzt die Logdatei auf die Datei mit Namen <file>.
Beispiel	`OPT.ics_dir . '\temp\my.log' MESSAGE.SET_LOG`

Methode	**`MESSAGE.GET_LOG[][s file]`**
Beschreibung	Gibt den Namen der aktuellen Logdatei zurück.
Beispiel	`MESSAGE.GET_LOG ['c:\iltools\temp\my.log']`

Methode	**`MESSAGE.RENAME_LOG[s file][]`**

Beschreibung Gibt der aktuellen Logdatei einen neuen Namen. Der Inhalt der alten Logdatei wird zuerst in die neue Logdatei umkopiert und danach die alte Logdatei geloescht. Danach werden die weiteren Meldungen an die neue Logdatei angehängt.

Beispiel
```
OPT.data_dir . '\myfile.log' MESSAGE.RENAME_LOG
```

Methode **MESSAGE.SET_SILENT[b silent][]**

Beschreibung Falls <silent> = TRUE ist, wird keinen Meldungen mehr in die Logdatei geschrieben. Falls <silent> = FALSE (Default) ist, werden Meldungen normal in die Logdatei geschrieben. <silent> kann auch über die Kommandozeile mit dem Switch -silent gesetzt werden (z.B. ics.exe -silent ...).

Beispiel
```
TRUE MESSAGE.SET_SILENT
```

Methode **MESSAGE.GET_SILENT[][b <silent>]**

Beschreibung Fragt den <silent> Status ab.

Beispiel
```
TRUE MESSAGE.SET_SILENT MESSAGE.GET_SILENT [TRUE]
```

11. Klasse OGC

11.1. Allgemeines

Mit den Methoden der Klasse OGC werden Methoden gemäss Open GIS Spezifikation zur Verfügung gestellt. Die Klasse OGC muss mit |LOAD geladen werden.

11.2. Well Known Text (WKT)

Methode **OGC.GEOM2WKT [o geometry][s wkt]**

Beschreibung Diese Methode wandelt eine Geometrie oder eine Liste von Geometrien in einen WKT String um.

Beispiel
```
'1.0/2.0' TO_POINT OGC.GEOM2WKT ['POINT(1.0 2.0)']
```

Methode **OGC.WTK2GEOM [s wkt][o geometry]**

Beschreibung Wandelt einen String im WKT Format in eine Geometrie oder eine Liste von Geometrien um.

Beispiel
```
'POINT(1.0/2.0)' OGC.WKT2GEOM [point(1.0,2.0)]
```

11.3. Well Known Binary (WKB)

Methode **OGC.GEOM2WKB [o geometry][B wkb]**

Beschreibung Diese Methode wandelt eine Geometrie oder eine Liste von Geometrien in einen WKB Blob um.

Beispiel
```
'1.0/2.0' TO_POINT OGC.GEOM2WKB [blob]
```

Methode **OGC.WTB2GEOM [B wkb][o geometry]**

Beschreibung Wandelt einen Blob im WKB Format in eine Geometrie oder eine Liste von Geometrien um.

Beispiel
```
VAR.POINT_BLOB OGC.WKB2GEOM [point(1.0,2.0)]
```

12. Klasse SERIAL

12.1. Allgemeines

Mit der Klasse SERIAL können Objekte im GeoShop Serial Format gelesen und gespeichert werden. Im Serial-Format werden z.B. alle Benutzerdefinitionen (.usr) und sonstigen Konfigurationsdateien des GeoShop gespeichert. Die Klasse SERIAL muss nicht mit |LOAD geladen werden.

12.2. Lesen und Schreiben

Methode **SERIAL.READ_OBJECT [i file][o object, b status]**

Beschreibung Liest ein Serial-Objekt von der Textdatei <file>. <file> muss vorgängig mit TEXTFILE.OPEN geöffnet worden sein.

Beispiel
```
IF 'test.usr' TEXTFILE.OPEN NOT THEN
   ERROR 'test.usr not found'
   HALT
END_IF => VAR.F
IF VAR.F SERIAL.READ_OBJECT THEN
   DISP ! [m user]
END_IF
VAR.F TEXTFILE.CLOSE
```

Methode **SERIAL.LOAD_OBJECT[s file][o object,b status]**

Beschreibung Kurzversion von SERIAL.READ_OBJECT ohne die Notwedigkeit die Textdatei zuerst mit TEXTFILE.OPEN zu öffnen. Es kann nur ein einziges Objekt aus der Textdatei <file> gelesen werden.

Beispiel
```
IF 'test.usr' SERIAL.LOAD_OBJECT THEN
   DISP ! [m user]
END_IF
```

Methode **SERIAL.WRITE_OBJECT[i file,o object][]**

Beschreibung Schreibt das Objekt <o> in die Textdatei <file>. <file> ist ein Texthandle, welches von TEXTFILE.CREATE zurückgegeben wurde.

Beispiel
```
'test.usr' TEXTFILE.CREATE POP => VAR.F
VAR.F &USER SERIAL.WRITE_OBJECT
VAR.F TEXTFILE.CLOSE
```

Methode **SERIAL.SAVE_OBJECT[s file,o object][b status]**

Beschreibung Kurzversion von SERIAL.WRITE_OBJECT ohne die Notwendigkeit zuerst eine Textdatei mit TEXTFILE.CREATE zu erzeugen. Es kann nur ein einziges Objekt in die Textdatei <file> geschrieben werden.

Beispiel
```
'test.usr' &USER SERIAL.SAVE_OBJECT => VAR.STATUS
```

13. Klasse REGEX

13.1. Allgemeines

Mit der Klasse REGEX werden Methoden für die Erkennung von Mustern in Zeichenketten bereit gestellt. Die Klasse REGEX basiert auf der freien Bibilothek PCRE2. Für eine detailierte Beschreibung der möglichen Muster wird hier daher auf die Webseite http://www.pcre.org/current/doc/html/pcre2syntax.html verwiesen. Die Klasse REGEX muss mit |LOAD geladen werden.

13.2. Methoden

Methode	**REGEX.MATCH [s subject,s pattern][i pos, b status]**
Beschreibung	Erkennt das Muster <pattern> in der Zeichenkette <subject>. Falls das Muster gefunden wurde, wird TRUE und die 1. Position des Muster in der Zeichenkette zurück gegeben, sonst FALSE. Die Syntax für <pattern> muss einem regulären Ausdruck in PCRE2 Schreibweise entsprechen.
Beispiele	

```
! erkennt die erste Zeichenfolge aus 3 Zahlen
'abc123uvw789' '\d\d\d' REGEX.MATCH [3,TRUE]

! erkennt IP-Adressen der Form ddd.ddd.ddd.ddd
'255.255.255.000' '\b\d{3}\.\d{3}\.\d{3}\.\d{3}\b'
REGEX.MATCH [0,TRUE]

! erkennt NBIdent der Form ZHddddddddddd
'ZH0000000000A' '\bZH\d{11}\b'
REGEX.MATCH [FALSE]
```

14. Klasse REGISTRY

14.1. Allgemeines

Mit den Methoden der Klasse REGISTRY können Windows Registry Einträge gelesen werden. Die Klasse REGISTRY muss mit |LOAD geladen werden.

14.2. REGISTRY

Methode	**REGISTRY.SET_32 [] []**
Beschreibung	Auf Windows 64 Systemen wird mit dieser Methode definiert, dass die System 32 Registry gelesen wird. Auf Windows 64 Systemen ist der Default die System 64 Registry. Auf Windows 32 Systemen hat diese Methode keine Auswirkung. System 32 Registry: %winddir%\syswow64\regedit.exe. System 64 Registry: %winddir%\regedit.exe.
Beispiel	

```
REGISTRY.SET_32 []
```

Methode	**REGISTRY.SET_64 [] []**
Beschreibung	Auf Windows 64 Systemen wird mit dieser Methode definiert, dass die System 64 Registry gelesen wird. Auf Windows 64 Systemen ist der Default

	die System 64 Registry. Auf Windows 32 Systemen hat diese Methode keine Auswirkung. System 32 Registry: %winddir%\syswow64\regedit.exe. System 64 Registry: %winddir%\regedit.exe.
Beispiel	`REGISTRY.SET_32 []`

Methode	**REGISTRY.READ_KEY [s key] [m values]**
Beschreibung	Diese Methode liest einen Windows Registry Key. Die Methode bring eine Map mit den Namen und Werten für den Key zurück. Subkeys sind als als Submaps in der Map enthalten.
Beispiel	`'HKEY_LOCAL_MACHINE\SOFTWARE\Windows' REGISTRY.READ_KEY [m]`

Methode	**REGISTRY.READ_KEY _VALUE[s key] [* value]**
Beschreibung	Diese Methode liest einen Windows Registry Key. Die Methode bring eine Map mit den Namen und Werten für den Key zurück. Subkeys sind als als Submaps in der Map enthalten. Kann der Key nicht gelesen werden, wird `NULL` zurückgegeben.
Beispiel	`'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\ProgramFilesDir' REGISTRY.READ_KEY_VALUE [*]`

14.3. Skriptbeispiel

```
|LOAD REGISTRY

'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion' => VAR.KEY
VAR.KEY REGISTRY.READ_KEY => VAR.VALUE
DISPLAY ''
DISPLAY VAR.KEY
DISPLAY VAR.VALUE                    ! Displays map of key
DISPLAY VAR.VALUE.ProgramFilesDir    ! Displays value of ProgramFilesDir of map

'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\ProgramFilesDir' => VAR.NAME
VAR.NAME REGISTRY.READ_KEY_VALUE => VAR.VALUE
DISPLAY ''
DISPLAY VAR.NAME
DISPLAY VAR.VALUE                    ! Displays value
```

15. Klasse SOCKET

15.1. Allgemeines

Mit den Methoden der Klasse SOCKET kann man via TCP/IP auf Resourcen im Netzwerk (Intranet oder Internet) zugreifen. Die Klasse SOCKET muss nicht mit |LOAD geladen werden.

15.2. Verbindung Aufbauen / Abbrechen

Methode	**`SOCKET.OPEN` [s address,i port][s socket,b status]**
Beschreibung	Die Methode SOCKET.OPEN öffnet einen Socket <s> für den späteren Zugriff (Lesen oder Schreiben). In <address> muss eine gültige Netwerk-adresse angegeben werden.

Beispiel

```
! Verbindung mit dem infoGrips Server aufnehmen
IF 'www.infogrips.ch' 80 SOCKET.OPEN NOT THEN
   ERROR 'es konnte keine Verbinung hergestellt werden.'
   HALT
END_IF => VAR.SOCKET
```

Methode **SOCKET.CLOSE [i socket][]**

Beschreibung Schliesst den mit SOCKET.OPEN geöffneten Socket <socket>.

Beispiel

```
VAR.SOCKET SOCKET.CLOSE
```

15.3. Lesen / Schreiben

Methode **SOCKET.READ_BYTE [i socket][i byte,b status]**

Beschreibung Die Methode SOCKET.READ_BYTE liest ein Byte von dem angegebenen Socket.

Beispiel

```
VAR.SOCKET SOCKET.READ_BYTE [65,TRUE]
```

Methode **SOCKET.READLN_STRING [i socket][s string,b status]**

Beschreibung Liest einen String vom angegebenen Socket. Es wird bis zum nächsten Zeilenumbruch (ASCII = 10) gelesen.

Beispiel

```
VAR.SOCKET SOCKET.READLN_STRING ['infoGrips GmbH',TRUE]
```

Methode **SOCKET.WRITE_BYTE [i socket,i byte][b status]**

Beschreibung Übermittelt ein Byte an den Socket.

Beispiel

```
VAR.SOCKET 99 SOCKET.WRITE_BYTE [TRUE]
```

Methode **SOCKET.WRITE_STRING [i socket,s string][b status]**

Beschreibung Übermittelt einen String an den Socket.

Beispiel

```
VAR.SOCKET 'hello, world' SOCKET.WRITE_STRING [TRUE]
```

Methode **SOCKET.WRITELN_STRING [i socket,s string][b status]**

Beschreibung Übermittelt den String <string> an den Socket. Anschliessend wird ein Zeilenumbruch (ASCII = 10) an den Socket übermittelt.

Beispiel

```
VAR.SOCKET 'hello, world' SOCKET.WRITELN_STRING [TRUE]
```

16. Klasse TEXTFILE

16.1. Allgemeines

Mit den Methoden der Klasse TEXTFILE kann auf eine oder mehrere Textdateien zugegriffen werden. Die Klasse TEXTFILE muss nicht mit |LOAD geladen werden.

16.2. Lesen von Textfiles

Methode **TEXTFILE.OPEN [s filename][i fileid, b status]**

Beschreibung Die Methode TEXTFILE.OPEN öffnet ein bestehendes Textfile für den Lesezugriff. In <fileid> wird, falls das Textfile geöffnet werden konnte, ein Filehandle zurückgeliefert.

Beispiel

```
IF 'test.dat' TEXTFILE.OPEN THEN
   => VAR.F1
ELSE
   MESSAGE 'unable to open textfile'
   HALT
END_IF
```

Methode **TEXTFILE.READLN [i filedid][s line, b status]**

Beschreibung Liest eine Zeile aus File <fileid> und gibt das Resultat in <line> zurück.

Beispiel

```
WHILE VAR.F1 TEXTFILE.READLN DO
   DISPLAY $
END_WHILE
```

Methode **TEXTFILE.READ_OBJECT [i filedid][* obj, b status]**

Beschreibung Liest ein Objekt aus der Textdatei. Die Objekte müssen vorher mit TEXTFILE.WRITE_OBJECT in die Textdatei geschrieben worden sein.

Beispiel

```
WHILE VAR.F1 TEXTFILE.READ_OBJECT DO
   DISP
END_WHILE
```

16.3. Erzeugen von Textfiles

Methode **TEXTFILE.CREATE [s filename][i fileid, b status]**

Beschreibung Erzeugt ein neues Textfile mit Namen <filename>. In <fileid> wird ein Filehandle auf die erzeugte Datei zurückgegeben.

Beispiel

```
IF 'test.dat' TEXTFILE.CREATE THEN
   => VAR.F1
ELSE
   DISPLAY ‘unable to create textfile’
END_IF
```

Methode **TEXTFILE.WRITE [i fileid,s buffer][]**

Beschreibung Schreibt den Buffer <buffer> auf das File <fileid>. Das File muss mit TEXTFILE.CREATE erzeugt worden sein.

Beispiel

```
VAR.F1 'hello, World' TEXTFILE.WRITE
```

Methode **TEXTFILE.WRITELN [i fileid,s buffer][]**

Beschreibung Schreibt den Buffer <buffer> auf das File <fileid>. Am Ende von <buffer> wird ein Newline Zeichen in das File geschrieben. Das File muss mit TEXTFILE.CREATE erzeugt worden sein.

Beispiel

```
VAR.F1 'hello, World' TEXTFILE.WRITELN
```

Methode `TEXTFILE.WRITE_OBJECT [i fileid,* obj][]`

Beschreibung Schreibt das Objekt <obj> in das File <fileid>. Die Objekte werden in einem speziellen Format in die Textdatei geschrieben, so dass sie mit TEXTFILE.READ_OBJECT wieder gelesen werden können. Achtung: Die Methode TEXTFILE.WRITE_OBJECT sollte nicht für die längerfristige Archivierung von Objekten benutzt werden, da das Speicherformat in Zukunft ändern könnte.

Beispiel

```
VAR.F1 &IN TEXTFILE.WRITE_OBJECT
```

Methode `SERIAL.SAVE_OBJECT [s filename,* obj][b status]`

Beschreibung Schreibt das Objekt <obj> in das Textfile <filename> im GeoShop Serial-Format. Es kann nur ein einziges Objekt in die Datei <filename> geschrieben werden.

Beispiel

```
IF 'test.usr' &USER SERIAL.WRITE_OBJECT NOT THEN
   ERROR 'unable to save object to file test.usr'
END_IF
```

Methode `SERIAL.WRITE_OBJECT [i fileid,* obj][b status]`

Beschreibung Schreibt das Objekt <obj> in das Textfile <fileid> im GeoShop Serial-Format.

Beispiel

```
IF VAR.F1 &USER SERIAL.WRITE_OBJECT NOT THEN
   ERROR 'unable to save object to file test.usr'
END_IF
```

Methode `TEXTFILE.OPEN_APPEND [s filename][i fileid, b status]`

Beschreibung Die Methode TEXTFILE.OPEN_APPEND öffnet ein bestehendes Textfile für den Schreibzugriff am Ende des Files. In <fileid> wird, falls das Textfile geöffnet werden konnte, ein Filehandle zurückgeliefert.

Beispiel

```
IF 'test.dat' TEXTFILE.OPEN_APPEND THEN
   => VAR.F1
ELSE
   DISPLAY 'unable to open to append textfile'
   HALT
END_IF
```

16.4. Schliessen von Textfiles

Methode `TEXTFILE.CLOSE [i fileid][]`

Beschreibung Schliesst ein mit TEXTFILE.CREATE oder TEXTFILE.OPEN geöffnetes File.

Beispiel

```
VAR.F1 TEXTFILE.CLOSE
```

16.5. Weitere

Methode `TEXTFILE.GET_FILES [s File-Pattern][li list]`

Beschreibung Liest die Filenamen aufgrund eines File-Pattern und liefert diese Filenamen in einer Liste zurück.

Beispiel

```
'c:\temp\*.txt' TEXTFILE.GET_FILES [list]
```

Methode	**TEXTFILE.IS_DIRECTORY [s File][b status]**
Beschreibung	Testet, ob ein File ein Directory ist oder nicht. Liefert den Status TRUE oder FALSE zurück.
Beispiel	`'c:\temp' TEXTFILE.IS_DIRECTORY [TRUE]`

Methode	**TEXTFILE.REMOVE [s File][b status]**
Beschreibung	Löscht ein File. Liefert den Status TRUE oder FALSE zurück.
Beispiel	`'c:\temp\test.txt' TEXTFILE.REMOVE [TRUE]`

Methode	**TEXTFILE.GET_FILEDATE [s File][i date,i time]**
Beschreibung	Gibt das Erstelldatum und die Erstellzeit (hhmmss) einer Datei <File> zurück.
Beispiel	`'c:\temp\test.txt' TEXTFILE.GET_FILEDATE [20050327,100855]`

Methode	**TEXTFILE.GET_FILESIZE [s File][i size]**
Beschreibung	Gibt die Grösse einer Datei in Byte zurück.
Beispiel	`'c:\temp\test.txt' TEXTFILE.GET_FILESIZE [800857]`

Methode	**TEXTFILE.GET_POS [i fileid][i pos,b state]**
Beschreibung	Gibt die aktuelle Schreibposition einer geöffneten Textdatei zurück.
Beispiel	`VAR.F1 TEXTFILE.GET_POS [TRUE,7897]`

Methode	**TEXTFILE.SET_POS [i fileid,i pos][b state]**
Beschreibung	Setzt die aktuelle Leseposition einer geöffneten Textdatei.
Beispiel	`VAR.F1 7897 TEXTFILE.SET_POS [TRUE]`

17. Klasse TRANSFORM

17.1. Allgemeines

Mit den Methoden der Klasse TRANSFORM können Geometrien von einem Koordinationsystem in ein anderes Koordinatensystem transformiert werden. Die Klasse TRANSFORM muss mit |LOAD geladen werden.

17.2. TRANSFORM

Methode	**TRANSFORM.EPSG [i EPSGIN, i EPSGOUT, g geometry] [g geometry]**
Beschreibung	Diese Methode transformiert eine Geometrie vom EPSGIN-Koordinatensystem in das EPSGOUT-Koordinatensystem. Unterstützte EPSG-Koordinationsysteme sind: 3785 (Google), 4326 (WGS84), 2056 (CH1903+ / LV95), 21781 (CH1903), 21780 (LI1903). Es sind nicht alle Wege der möglichen Transformationen implementiert.
Beispiel	`21781 4326 VAR.GEOM TRANSFORM.EPSG [geometry]`

17.3. Spezielles Dreiecksvermaschung 2056 <> 21781

Für die Transformation 21781 <> 2056 wird eine Dreiecksvermaschung benötigt. Standardmässig wird die schweizerische Dreiecksvermaschung `ILTOOLS_DIR\system\db\grid\CHENyx06.bin` verwendet.

Methode	**TRANSFORM.SET_SWISS_GRID_FILE [s file] []**
Beschreibung	Mit dieser Methode kann ein zusätzliches File mit einer Dreieckvermaschung für die Transformation 21781 <> 2056 definiert werden. Wird für eine Koordinate ein Dreieck innerhalb dieser Dreieckvermaschung gefunden, wird dieses Dreieck für die Transformation verwendet. Ansonsten wird die Dreiecksvermaschung aus `CHENyx06.bin` verwendet. Das File der Dreiecksvermaschung kann ein ASCII- oder Binär File sein. Das ASCII-Format ist in der Dokumentation zu FINELTRA der swisstopo beschrieben .
Beispiel	`'\db\grid\BEENyx15.bin' TRANSFORM.SET_SWISS_GRID_FILE`

Methode	**TRANSFORM.WRITE_SWISS_GRID_FILE_TO_INTERLIS [s GRID-file, s INTERLIS-File, b reverse] []**
Beschreibung	Mit dieser Methode kann ein File mit einer Dreieckvermaschung in ein INTERLIS-itf-File transferiert werden. Das File der Dreiecksvermaschung kann ein ASCII- oder Binär File sein. Das ASCII-Format ist in der Dokumentation zu FINELTRA der swisstopo beschrieben . Die Daten im INTERLIS-itf-File entsprechen dem Modell `ILTOOLS_DIR\system\models\Geometry.ili`. Mit dem BOOLEAN `reverse` kann definiert werden, ob die LV03-Koordinaten - `reverse=FALSE` - oder die LV95-Koordinaten - `reverse=TRUE` - nach INTERLIS geschrieben werden sollen. Aus dem INTERLIS-File kann die Dreiecksvermaschung nach DGN/DXF/DWG zur Visualisierung geschrieben werden.
Beispiel	`'\db\grid\BEENyx15.bin' 'C:\BEENyx15_lv03.itf' FALSE TRANSFORM.WRITE_SWISS_GRID_FI`

18. Klasse TTF

18.1. Allgemeines

Mit den Methoden der Klasse TFF können Geometrien von True Type Fonts gelesen werden. Die Klasse TTF muss mit |LOAD geladen werden. Die Methoden eignen sich insbesonder um Geometrien von True Type Fonts vom Typ Symbol zu lesen.

18.2. Spezielles

Die True Type Fonts werden durch die Klasse TTF über folgendes Java-Programm gelesen.

`ICS_DIR\system\bin\ttf2itf.jar`

Informationen zu diesen Java-Program werden angezeigt mit:

```
java -jar ttf2itf.jar
```

18.3. TTF.FONTS

Gelesene Fonts werden in der Map TTF.FONTS abgelegt. Die Map weist folgende Struktur auf.

Komponente	Typ	Beschreibung
TTF.FONTS.<fontname>	map	Pro gelesenen Font fontname ein Eintrag.
TTF.FONTS.<fontname>.NAME	string	Name des Fonts.
TTF.FONTS.<fontname>.STYLE	string	Style des Fonts.
TTF.FONTS.<fontname>.TYPE	string	Type des Fonts. TEXT oder SYMBOL .
TTF.FONTS.<fontname>.SIZE	int	Pointgrösse des Fonts.
TTF.FONTS.<fontname>.CHARACTERS	map	Map mit Characters des Fonts als ASCII-Code.
TTF.FONTS.<fontname>.CHARACTERS.<char>	list	Liste der Geometrien des Characters.

18.4. TTF

Methode **TTF.READ_FONT [s fontname, s style, i pointsize] [b status]**

Beschreibung Liest den True Type Font fontname im Style style mit der Pointgrösse pointsize. Kann der Font gelesen werden, gibt die Methode TRUE zurück, ansonsten FALSE. Als style werden folgende Werte unterstützt: plain|italic|bold|bold,italic|symbol. Für Symbol-Fonts sollte symbol verwendet werden.

Beispiel
```
'Wingdings_Standard' 'symbol' 10 TTF.READ_FONT [TRUE]
```

Methode **TTF.READ_FONT2 [s fontname, s style, i pointsize, r offsetx, r offsety] [b status]**

Beschreibung Wie TTF.READ_FONT . Zusätzlich kann ein Offset offsetx offsety definert werden, um den das Zentrum - der Origin - der Geometrien verschoben wird. offsetx offsety sind Faktoren bezogen auf die Fontbox des Fonts.

Beispiel
```
'Wingdings_Standard' 'symbol' 10 -0.5 -0.5 TTF.READ_FONT2 [TRUE]
```

Methode **TTF.READ_CHARACTER_GEOMETRY [s fontname, s style, i|s character] [li geometries]**

Beschreibung Diese Methode liest die Geometry eines Characters character eines Fonts. Der Character kann als String oder als Integer des ASCII-Codes übergeben werden. Die Methode bring eine Liste der Geometrien des Characters zurück oder NULL, falls der Character nicht gelsen werden kann. Der Font muss vorgängig einmal mit TTF.READ_FONT oder TTF.READ_FONT2 gelesen werden.

Beispiel
```
'Wingdings_Standard' 'symbol' 'A' TTF.READ_CHARACTER_GEOMETRY [list]
```

Methode **TTF.WRITE_FONT_OBJECT_FILE [s fontname, s style, s file] [b status]**

Beschreibung Schreibt einen mit TTF.READ_FONT oder TTF_READ_FONT2 gelesenen Font als Object in ein File.

Beispiel
```
'Wingdings_Standard' 'symbol' '\iltools\user\font\myfont.obj' TTF.WRITE_FONT_OBJECT_FI
```

Methode	**TTF.READ_FONT_OBJECT_FILE [s file] [s style, s fontname, b status]**
Beschreibung	Liest einen mit `TTF.WRITE_FONT_OBJECT_FILE` geschriebenen Font. Wenn der Status `TRUE` ist, wird zusätzlich der Fontname und der Style zurückgegeben. `TTF.READ_FONT_OBJECT_FILE` ist schneller als `TTF.READ_FONT` oder `TTF.READ_FONT2` . Zusammen mit `TTF.WRITE_FONT_OBJECT_FILE` eignet sich die Methode um einmal gelesene Fonts in einem File abzuspeichern und wiederholt schneller zu lesen.
Beispiel	`'\iltools\user\font\myfont.obj' TTF.READ_FONT_OBJECT_FILE ['symbol','Wingdings_Sta`

Methode	**TTF.DISPLAY_FONT [] []**
Beschreibung	Zeigt alle gelesenen Fonts an.
Beispiel	`TTF.DISPLAY_FONTS []`

18.5. Skriptbeispiel

```
|LOAD TTF

! read font
!----------
'Wingdings_Standard' 'symbol' 10 -0.5 -0.5 TTF.READ_FONT2 => VAR.STATUS
IF VAR.STATUS NOT THEN
   ERROR 'could not read font'
   HALT
END_IF

! display font
!-------------

! display fonts
DISPLAY TTF.FONTS

! display font Wingdings_Standard
DISPLAY TTF.FONTS.Wingdings_Standard

! display font Wingdings_Standard characters
DISPLAY TTF.FONTS.Wingdings_Standard.CHARACTERS

! display font Wingdings_Standard character geometries
&TTF.FONTS.Wingdings_Standard.CHARACTERS MAPRESET
WHILE &TTF.FONTS.Wingdings_Standard.CHARACTERS MAPSCAN DO
   => VAR.CHAR
   => VAR.LIST

   DISPLAY 'Geometry if character:,VAR.CHAR

   &VAR.LIST RESET_READ
   WHILE &VAR.LIST READ_NEXT DO
      => VAR.GEOM
      VAR.GEOM DISP
   END_WHILE
END_WHILE
```

```
! reading character A by character
!--------------------------------
'Wingdings_Standard' 'symbol' 'A' TTF.READ_CHARACTER_GEOMETRY => VAR.LIST
IF &VAR.LIST IS_NOT_NULL THEN
   VAR.LIST DISP
END_IF

! reading character A by character ASCII code 65
!---------------------------------------------
'Wingdings_Standard' 'symbol' 65 TTF.READ_CHARACTER_GEOMETRY => VAR.LIST
IF &VAR.LIST IS_NOT_NULL THEN
   VAR.LIST DISP
END_IF
```

B. Vordefinierte Maps

In iG/Script sind einige Maps bereits standardmässig vorhanden (z.B. `VAR`). Der Benutzer muss diese Maps nicht explizit deklarieren. Die Bedeutung bzw. der Ihnhalt dieser Maps ist in der folgende Tabelle beschrieben.

Name der Map	**Beschreibung**
`CLASSES`	Map welche alle ICS Klassen als Map enthält. Die Klassenmaps ihrerseits enthalten als Komponenten ihre Methoden. `DISPLAY CLASSES.TEXTFILE` zeigt z.B. alle Methoden der Klasse `TEXTFILE` an.
`PROCEDURES`	Map welche alle vom Benutzer definierte Prozeduren enthält. Mit `DISPLAY PROCEDURES` können z.B. alle in einem Skript definierten Procedure angezeigt werden.
`OPT`	Map welche alle an den Skript übergebenen Kommandozeilenparameter enthält. Wenn z.B. ein Skript mit `ics.exe -script test.cfg -input test.dat` aufgerufen wurde, kann auf den `-input` Parameter im Skript mit `OPT.input` zugegriffen werden.
`LOCAL`	Map für die Abspeicherung von Zwischenergebnissen in Prozeduren.
`IN`	Map in welcher ein Inputmodul normalerweise das nächste Inputobjekt liefert.
`OUT`	Map in welcher ein Outputmodul normalerweise das Outputobjekt erwartet.
`VAR`	Map für die Speicherung von temporären Variablen.
`SOURCES`	MAP welche die Namen aller mit `|INCL` eingebunden Skriptdateien und den Namen des Hauptskripts enthält.

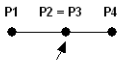
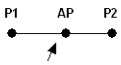
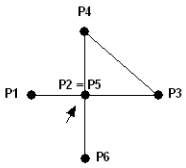
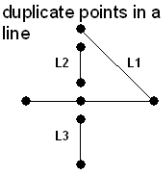
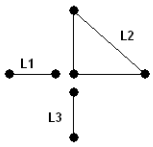
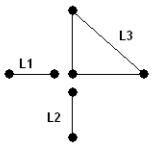
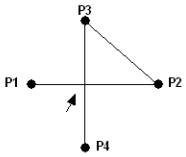
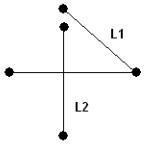
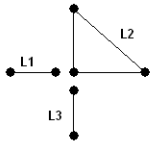
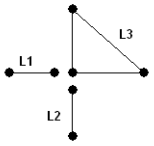
C. Methode ICS.GEOM_CLEAN Details

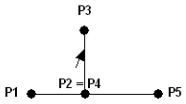
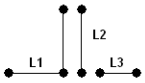
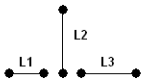
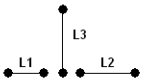
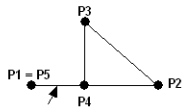
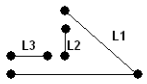
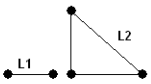
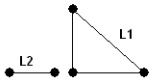
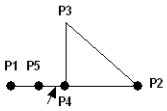
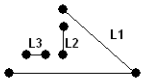
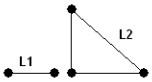
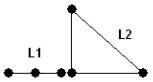
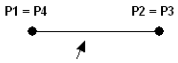
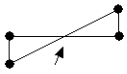
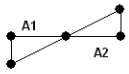
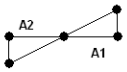
Das Geometrie-Bereinigungsverfahren der Methode `ICS.GEOM_CLEAN` der Klasse `ICS` wird an dieser Stelle näher beschrieben.

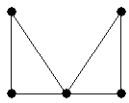
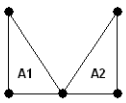
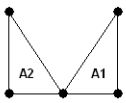
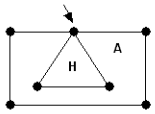
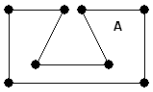
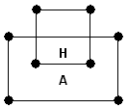
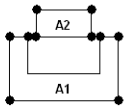
Bei den ESRI-Schnittstellen kann bei Bedarf auch eine Bereinigung der Geometrien mit ESRI-Methoden aus ArcObjects angewendet werden. Die Bereinigung erfolgt dabei mit der Methode `ESRI.GEOM_SIMPLIFY`. Zur Anwendung dieser Methode siehe mehr bei den Output-Modulen der ESRI-Schnittstellen.

Grafische Übersicht über das Bereinigunsverfahren der Geometrien.

Tabelle C.1.

Geometry Original	Geometry cleaned ICS.GEOM_CLEAN ORACLE	Geometry cleaned ICS.GEOM_CLEAN ESRI	Geometry cleaned ESRI.GEOM_SIMPLIFY
line/arc no extend 	eliminate NULL	like ORACLE	like ORACLE
line/arc duplicate point 	eliminate duplicate point	like ORACLE	like ORACLE
line straight arc 	eliminate arc point convert to line	like ORACLE	like ORACLE
line intersection by point 	eliminate intersections split into list of lines with no intersection and duplicate points in a single line 	eliminate intersections split into list of lines with no intersection at all 	like ESRI but geometries and the order of the lines can differ 
line intersection by cut 	eliminate intersections split into list of lines with no intersection in a single line 	eliminate intersections split into list of lines with no intersection at all 	like ESRI but geometries and the order of the lines can differ 

line duplicate 	eliminate duplicates split into list of lines with no duplicates in a single line 	eliminate duplicates split into list of lines with no duplicates at all 	like ESRI but geometries and the order of the lines can differ 
line overlap full 	eliminate overlaps split into list of lines with no overlaps in a single line 	eliminate overlaps split into list of lines with no overlaps at all 	like ESRI but geometries and the order of the lines can differ 
line overlap partial 	eliminate overlaps split into list of lines with no overlaps in a single line 	eliminate overlaps split into list of lines with no overlaps at all 	like ESRI but geometries and the order of the lines can differ 
area no extension 	eliminate NULL	like ORACLE	like ORACLE
area self intersection 	eliminate intersections split into list of areas with no intersection in a single area 	like ORACLE	like ORACLE but geometries and the order of the lines can differ 

area self touch 	eliminate touches split into list of areas with no touch in a single area 	like ORACLE	like ORACLE but geometries and the order of the lines can differ 
area border/hole self touch 	eliminate touches 	like ORACLE	like ORACLE but geometry (order of points) can differ
area border/hole intersection 	eliminate intersections split into list of areas with no intersection in a single area 	like ORACLE	like ORACLE but geometries (order of points) and the order of the areas can differ

Textuelle Übersicht über die von den Systemen akzeptierten Geometrien

Tabelle C.2.

Geometry	Case	Oracle	ESRI
line	no extension	not valid	not valid
line	duplicate point	not valid	not valid
line	straight arc	not valid	not valid
line	self intersecition	valid	not valid
line	line segment duplicate	valid	not valid
line	line segment overlap	valid	not valid
list of lines	single line, see cases of line	see line	see line
list of lines	intersection of two single lines	valid	not valid
list of lines	duplicates of two single lines	valid	not valid
list of lines	overlaps of two single lines	valid	not valid
area	no extension	not valid	not valid
area	self intersection	not valid	not valid
area	self touch	not valid	not valid
area	border/hole touch	not valid	not valid
area	border/hole intersection	not valid	not valid
list of area	single area, see cases of area	see area	see area
list of area	intersection of two single areas	valid	not valid
list of area	duplicates of two single areas	valid	not valid
list of area	overlaps of two single areas	valid	not valid

Anmerkungen zu den Methoden

Methode `ICS.GEOM_CLEAN System ORACLE`

Die Methode ist ein Implementation der infoGrips GmbH. Die Methode bereinigt eine Geometrie so, dass die Oracle Geometrie Validierungsfunktion SDO_GEOM.VALIDATE_GEOMETRY die Geometrie als gültig interpretiert. Die Bereinigung führt auch Bereinigungen durch, die für Oracle nicht zwingend sind, zum Beispiel Selfintersections von Linien. Es kann Sonderfälle von Geometrien geben, die gar nicht bereinigt werden können, weil die Art der Bereinigung nicht definiert ist oder bei denen die Bereinigung nicht die erwarteten Resultate liefert.

Methode `ICS.GEOM_CLEAN System ESRI`

Die Methode ist ein Implementation der infoGrips GmbH. Die Methode bereinigt eine Geometrie so, dass die ESRI Geometrie Validierungsfunktion Check Geometry die Geometrie als gültig interpretiert. Eine genaue Spezifikationen, ob eine Geometrie gültig ist oder nicht, existiert nicht. Zum Teil verlangt Check Geometry die Einhaltung einer genauen Reihenfolge von

Punkten, Linien oder Flächen. Diese Vorgaben kann die Funktion nicht alle erfüllen, weil diese unbekannt sind. Darum empfehlen wir die Daten mit der Funktion `ESRI.GEOM_SIMPLIFY` zu bereinigen, falls die Daten für die Anwendung unter einem ESRI-Produkt bestimmt sind. Es kann Sonderfälle von Geometrien geben, die gar nicht bereinigt werden können, weil die Art der Bereinigung nicht definiert ist oder bei denen die Bereinigung nicht die erwarteten Resultate liefert.

Methode **`ESRI.GEOM_SIMPLIFY`**

Die Methode ist im ICS implementiert und basiert auf der Funktion Simplify von ESRI-ArcObjects. Das heisst, die Bereining der Geometrie erfolgt über eine Funktion von ESRI. Die Methode bedingt, dass ein lizenziertes ESRI-Produkt auf der Maschine existiert auf der die Methode angewendet wird.

D. Syntax der iG/Script Sprache

Nachfolgend ist die Syntax von iG/Script in Backus Naur Form angegeben. Die Metazeichen haben folgende Bedeutung:

<s>	Schlüsselwort s
[a]	a ist optional
{a}+	a kann 1 - N-mal wiederholt werden
{a}*	a kann 0 - N-mal wiederholt werden
a | b	a oder b

iG/Script Syntax:

```
<Trennzeichen> := Newline, Return, Space oder Tabulator
<Kommentar>    := ! gefolgt von beliebiger Zeichenfolge bis Zeilenende
<Zeichenkette> := beliebige Zeichenfolge ohne <Trennzeichen>

<iG/Script-Programm> := {<Deklarationen>}*
                         <Ausdruck>

<Deklarationen> := <Mapdeklaration>
                := <Prozedurdeklaration>

<Mapdeklaration> := MAP <Mapname>
                       {<Komponentenname> => <Komponentenwert>}*
                    END_MAP

<Prozdurdeklaration> := PROCEDURE <Prozedurname>
                           <Ausdruck>
                        END_PROCEDURE

<Ausdruck> := <Konstante>
           := <Methodenaufruf>
           := <Prozeduraufruf>
           := <Objekt>
           := <Referenz>
```



```
           := <Zuweisung>
           := <Abbildung>
           := <Bedingung>
           := <Displayausdruck>
           := <WHILE-Ausdruck>
           := <Breakanweisung>
           := <Continueanweisung>
           := <Returnanweisung>

           := <IF-Ausdruck>
           := {<Ausdruck>}+

Konstante := ganze Zahl (z.B. 0)
          := reelle Zahl (z.B. 1.0)
          := String
          := TRUE
          := FALSE

String   := '<Zeichenkette>'

Methodenaufruf := <Klasse>.<Methodenname>

Prozeduraufruf := <Prozedurname>

Returnanweisung := RETURN

<Objekt>     := [ROOT.]<Komponente>
<Komponente> := <Name> | <Komponente> . <Name>
<Name>       := <Zeichenkette>

Zuweisung := => <Objekt>
             -> <Objekt>

Abbildung := <Mapname>

Bedingung := <Ausdruck> = <Basistyp>
          := <Ausdruck> <> <Basistyp>
          := <Ausdruck> > <Basistyp>
          := <Ausdruck> < <Basistyp>
          := <boolscher Ausdruck>

Displayausdruck := DISPLAY <Displayliste>
                := ERROR <Displayliste>
                := STATUS <Displayliste>

<Displayliste> := {<Mapname>|<Objekt>|<String>|$}
                  {,<Mapname>|<Objekt>|<String>|$}*

WHILE-Ausdruck := WHILE <Bedingung> DO
                     <Ausdruck>
                  END_WHILE

Breakanweisung := BREAK

Continueanweisung := CONTINUE

IF-Ausdruck := IF <Bedingung> THEN
               {ELSIF <Bedingung> THEN <Ausdruck>}*
```



```
[ELSE  <Bedingung> THEN <Ausdruck>]
END_IF
```