ExLibris Alephino

PSI – Skripte

Inhaltsverzeichnis

Die PSI-Programmiersprache

Alephino 5.0 bietet mit PSI (Program Script Interpreter) ein universelle Programmierschnittstelle mit mächtigem Funktionsumfang. Diese richtet sich an ambitionierte Anwender, die bereits Programmierkenntnisse in einer syntaktisch ähnlichen Sprache wie etwa C/C++ oder Perl sammeln konnten. PSI kann genutzt werden, um spezielle Reports zu erstellen, Daten in der Alephino-Datenbank Datei-übergreifend nach selbstdefinierten Regeln auszuwerten und zu bearbeiten u.v.m.

PSI-Programmsyntax

Die Syntax eines PSI-Programms (list) ist im BNF Format beschrieben. Buchstaben, Zeichen und Wörter, die hier in Apostroph eingeschlossen werden, sind Terminal-Symbole und müssen in PSI Programmen in dieser Schreibweise ohne Apostroph benutzt werden. Das Terminal '\n' bedeutet Zeilenvorschub und entspricht der RETURN- oder ENTER Taste der Tastatur.)

```
list:
        | list '\n'
        | list defn '\n'
        | list asgn '\n'
        | list stmt '\n'
        | list expr '\n'

asgn:   VAR '=' expr
        | VAR '[' expr ']' '=' expr
        | ARG '=' expr
        | ARG '[' expr ']' '=' expr

stmt:   expr
        | 'return'
        | 'return' expr
        | 'exit'
        | 'exit' expr
        | PROCEDURE '(' arglist ')'
        | BLTINPROC '(' arglist ')'
        | 'auto' locals
        | 'while' '(' expr ')' stmt
        | 'if' '(' expr ')' stmt
        | 'if' '(' expr ')' stmt 'else' stmt
        | '{' stmtlist '}'

stmtlist:
        | stmtlist '\n'
        | stmtlist stmt
```

```
expr:   NUMBER
        | STRING
        | VAR
        | VAR '[' expr ']'
        | 'type' '(' VAR ')'
        | 'dim' '(' VAR ')'
        | ARG
        | ARG '[' expr ']'
        | asgn
        | FUNCTION '(' arglist ')'

        | BLTINFUNC '(' arglist ')'
        | '(' expr ')'
        | expr '+' expr
        | expr '-' expr
        | expr '*' expr
        | expr '/' expr
        | expr '%' expr
        | expr '^' expr
        | '-' expr
        | '+' expr
        | expr '>' expr
        | expr '>=' expr
        | expr '<' expr
        | expr '<=' expr
        | expr '==' expr
        | expr '!=' expr
        | expr '&&' expr
        | expr '||' expr
        | '!' expr

defn:   'func' FUNCTION '(' locals ')' stmt
        | 'proc' PROCEDURE '(' locals ')' stmt

arglist:
        | expr
        | arglist ',' expr

locals:
        | VAR
        | locals ',' VAR
```

Zwischen Terminal-Symbolen ist, zur Erhöhung der Lesbarkeit, eine beliebige Anzahl von Leerzeichen erlaubt. Wenn '\n' direkt auf einen Backslash (\) folgt, dann wird der Zeilenvorschub ignoriert. Der Text nach zwei Schrägstrichen wird bis zum Zeilenende als Kommentar gewertet. Das Schlüsselwort 'quit' führt zum sofortigen Abbruch des PSI-Programms.

Syntax Beschreibung der Terminal-Symbole, die oben nicht erläutert wurden:

NUMBER: Numerische Konstante nach folgendem Schema
(Ausdrücke in eckigen Klammern sind optional):

digits ['.'] [digits] [exp] [sign] [digits]
oder
'.' [digits] [exp] [sign] [digits]

wobei:	digits	Eine oder mehrere Dezimalzahlen ('0','1','2',...,'9')
	exp	ist 'e' oder 'E'
	sign	ist '+' oder '-'

Beispiel: 1
3.12
.12
5.678e10
0.123E-6

Es gibt einige ***vordefinierte Konstanten*** (Sie können diese mit ihrem Namen ansprechen):

Name:	Wert:	Bedeutung:
PI	3.14159265358979323846	pi
E	2.71828182845904523536	e
GAMMA	0.57721566490153286060	Euler-Mascheroni
DEG	57.29577951308232087680	deg/radian (== 180/PI)
PHI	1.61803398874989484820	golden ratio
UNDEF	0	nicht definierter Typ(VAR)
NUM	1	numerischer Typ(VAR)
STR	2	Zeichenkette Typ(VAR)
ANUM	3	numerisches Feld Typ(VAR)
ASTR	4	Zeichenk. Feld Typ(VAR)
IN	0	fopen() Lesen
OUT	1	fopen() Schreiben
EXT	2	fopen() Lesen/Schreiben
UPD	3	fopen() Überschreiben

STRING: In Anführungszeichen (") eingeschlossene Zeichen. Einige spezielle, mit Backslash versehene Zeichen, können in die Zeichenkette eingesetzt werden:

\b	Ein Zeichen zurück
\f	Seitenvorschub
\n	Zeilenvorschub
\r	Wagenrücklauf
\t	TAB

\\	Backslash
\"	Anführungszeichen (")
\ddd	Oktaler Zeichenkode ddd (d kann die Werte '0','1',...,'7' annehmen)

Eine Zeichenkette kann in der Folgezeile fortgesetzt werden, wenn dies mit einem Backslash (\) eingeleitet wird.

VAR:
FUNCTION:
PROCEDURE:

Ausdrücke, die mit einem Buchstaben beginnen und Buchstaben oder Ziffern enthalten. Reservierte Ausdücke dürfen nicht verwendet werden (if, else, while, return, auto, exit, proc, func, type, dim) und Funktions- oder Prozedurnamen aus BLTINPROC oder BLTINFUNC (siehe unten)). Es wird zwischen Groß- und Kleinschreibung unterschieden ("groß" ist nicht gleich "Groß"). Sie sollten vermeiden, einen globalen Ausdruck in verschiedenen VAR, PROCEDURE oder FUNCTION zu benutzen. Lokale Variablen (definiert als 'auto' im stmt 'func' oder 'proc'), die den gleichen Namen wie eine globale Variable haben, sind vor der globalen Variablen "versteckt".

Beispiel:

```
zahl = 2
proc xyz() {
  auto zahl
  zahl = 1
}
xyz()
printl(zahl)
```

Ausgabe: 2

ARG: Wird benutzt um einen namenlosen Parameter einer 'func' oder 'proc' anzusprechen.

Syntax: '$'digits

Beispiel: $1
$12

ARG $n bezeichnet den n-th Parameter des Funktions/Prozeduraufrufs. Der erste Parameter ist $1.

Beispiel:

```
func subtr() {
  return $1-$2
}                              // Definition
a = subtr(2,5)                 // a wird -3
printl(a)                      // Ausgabe von a
```

BLTINFUNC:
BLTINPROC:

Eine der "eingebauten" Funktionen aus der PSI-Funktionsbibliothek.
Die verfügbaren Bibliotheksfunktionen werden im Anhang beschrieben.
Man unterscheidet zwischen *Funktionen*, die einen Wert als Funktionsergebnis zurückliefern, und *Prozeduren*, die keinen Wert liefern.

Erläuterungen zur Syntaxbeschreibung

Was bedeutet nun diese doch sehr formale Beschreibung eines PSI-Programms, die wohl nur Informatikern unmittelbar verständlich sein dürfte?

Schauen wir uns zunächst an, wie ein ***PSI-Programm*** (eine "Liste") aufgebaut sein kann:

- leer (/* nichts */)
  oder
- eine Liste, gefolgt von einem Zeilenvorschub (list '\n')
  oder
- eine Liste, gefolgt von einer Definition und einem Zeilenvorschub (list defn '\n')
  oder
- eine Liste, gefolgt von einer Zuweisung und einem Zeilenvorschub (list asgn '\n')
  oder
- eine Liste, gefolgt von einem Statement und einem Zeilenvorschub (list stmt '\n')
  oder
- eine Liste, gefolgt von einem Ausdruck und einem Zeilenvorschub (list expr '\n').

Vielleicht ist Ihnen schon aufgefallen, daß eine Liste – abgesehen von der ersten Regel – immer auch teilweise durch "sich selbst" erklärt wird. Dadurch kann man für "list" dann immer wieder eine der anderen Beschreibungen einsetzen und gelangt so zu einer Folge von Statements, Zuweisungen usw., die schließlich das PSI-Programm ausmachen.
Nun muß man schauen, wodurch sich die anderen Begriffe wie Statement, Zuweisung usw. ersetzen lassen, und wodurch dann deren Bausteine wieder usw. So gelangt an schließlich zu einem PSI-Programm.

Eine ***Zuweisung*** (asgn) kann sein:

- eine Variable bekommt einen neuen Wert (VAR '=' expr) oder
- eine indizierte Feldvariable ("Array") bekommt einen neuen Wert (VAR '[' expr ']' '=' expr)
  oder
- ein Argument bekommt einen neuen Wert (ARG '=' expr) oder
- ein indiziertes Argument-Feld bekommt einen neuen Wert (ARG '[' expr ']' '=' expr).

Der Index eines Feldes beginnt bei 0. Es sind nur eindimensionale Felder zulässig.

Ein ***Statement*** (stmt) kann sein:

- ein Ausdruck (expr)

oder
- eine Rückkehranweisung aus einer Prozedur ('return')
  oder
- eine Rückkehranweisung mit einem Wert, aus einer Funktion ('return' expr)
  oder
- der Ausstieg aus einem PSI Programm ('exit')
  oder
- ein Ausstieg mit einem Wert, aus einem PSI Programm ('exit' expr)
  oder
- der Aufruf einer vom Anwender definierten PROCEDURE mit möglichen Argumenten (PROCEDURE '(' arglist ')')
  oder
- der Aufruf einer Bibliotheksprozedur mit möglichen Argumenten (BLTINPROC '(' arglist ')')
  oder
- die Definition lokaler Variablen innerhalb einer, vom Anwender definierten PROCEDURE/FUNCTION ('auto' locals)
  oder
- ein 'while' Kommando, welches ein stmt ausführt, solange ein Ausdruck wahr (dh.nicht null) ist ('while' '(' expr ')' stmt)
  oder
- ein 'if' Kommando, welches ein stmt ausführt, wenn ein Ausdruck nicht null ist ('if' '(' expr ')' stmt)
  oder
- ein 'if' Kommando, welches das erste stmt ausführt, wenn ein Ausdruck nicht null ist und das zweite stmt ausführt, wenn der Ausdruck null ist ('if' '(' expr ')' stmt 'else' stmt)
  oder
- eine Statementliste, eingeschlossen in geschweifte Klammern ('{' stmtlist '}').

Eine ***Statementliste*** (stmtlist) kann sein:
- leer (/* nichts */)
  oder
- eine Statementliste gefolgt von einem Zeilenvorschub (stmtlist '\n')
  oder
- eine Statementliste gefolgt von einem Statement (stmtlist stmt).

Ein ***Ausdruck*** (expr) kann sein:
- eine numerische Konstante (NUMBER)
  oder
- eine Zeichenketten-Konstante (STRING)
  oder
- eine Variable mit einem Wert (VAR)
  oder
- ein indiziertes Variablenfeld mit einem Wert (VAR '[' expr ']')
  oder
- eine Funktion, die den Typ einer Variablen bestimmt und UNDEF, NUM, STR, NUM,

ASTR wiedergibt ('type' '(' VAR ')')
oder
- eine Funktion, die die Größe eines Feldes bestimmt und die Anzahl der Feldeinträge liefert ('dim' '(' VAR ')')
oder
- ein Argument mit dem Wert (ARG)
oder
- das indizierte Element eines Argument-Feldes mit einem Wert (ARG '[' expr ']')
oder
- eine Zuweisung (asgn)
oder
- eine, durch den Anwender definierte Funktion, die mit Argumenten aufgerufen wird (FUNCTION '(' arglist ')')
oder
- eine PSI-Bibliotheksfunktion, die mit Argumenten aufgerufen wird (BLTINFUNC '(' arglist ')')
oder
- ein, in Klammern eingeschlossener, Ausdruck ('(' expr ')')
oder
- ein arithmetischer Ausdruck (siehe unten)
oder
- ein relationaler Ausdruck (siehe unten)
oder
- ein logischer Ausdruck (siehe unten).

Ein ***arithmetischer Ausdruck*** kann sein:
- Zahlen addieren (expr '+' expr)
oder
- Zeichenketten verknüpfen (expr '+' expr)
oder
- subtrahieren (expr '-' expr)
oder
- multiplizieren (expr '*' expr)
oder
- dividieren (expr '/' expr)
oder
- modulo (expr '%' expr)
oder
- potenzieren (expr '^' expr)
oder
- negieren, z.B. UNARYMINUS ('-' expr)
oder
- ohne Auswirkung, z.B. UNARYPLUS ('+' expr).

Achtung:
Die Ausdrücke "S1 + S2" und "strcat(S1,S2)" haben die gleiche Auswirkung wenn S1 und S2

beides Zeichenketten sind. Andere arithmetische Operatoren akzeptieren nur numerische Ausdrücke.

Ein ***relationaler Ausdruck*** ist 1 (WAHR) wenn:

- der Erste größer ist als der Zweite (expr '>' expr)
  oder
- der Erste größer oder gleich dem Zweien ist (expr '>=' expr)
  oder
- der Erste kleiner ist als der Zweite (expr '<' expr)
  oder
- der Erste kleine oder gleich dem Zweiten ist (expr '<=' expr)
  oder
- der Erste gleich dem Zweiten ist (expr '==' expr)
  oder
- der Erste nicht gleich dem Zweiten ist (expr '!=' expr)

und 0 (UNWAHR) wenn nicht.

Achtung:
Die Ausdrücke "S1 RELOP S2" und "strcmp(S1,S2) RELOP 0" ergeben identische Ergebnisse wenn S1 und S2 beide Zeichenketten sind, und der RELOP einer dieser relationalen Operatoren ist:

'>', '>=', '<', '<=', '==', oder '!='.

Ein ***logischer Ausdruck*** ist 1 (WAHR) wenn:

- der Erste und der Zweite nicht 0 ist (expr '&&' expr)
  oder
- der Erste oder der Zweite nicht 0 ist (expr '||' expr)
  oder
- der Ausdruck 0 ist ('!' expr).

und 0 (UNWAHR) wenn nicht.

Achtung: Logische Operatoren erlauben nur numerische Ausdrücke als Argument.

Die ***Rangfolge*** der arithmetischen, relationalen und logischen ***Operatoren*** ist:

Assoziativität:	Operator:
rechts nach links	^
links nach rechts	UNARYMINUS UNARYPLUS !
links nach rechts	* / %
links nach rechts	+ -
links nach rechts	> >= < <= == !=
links nach rechts	&&
links nach rechts	||

rechts nach links = (assignment)

Assoziativität "rechts nach links" bedeutet,
daß "a ^ 2 ^ 3 ^ 4" gleichbedeutend mit "a ^ (2 ^ (3 ^ 4))" ist.

Assoziativität "links nach rechts" bedeutet,
daß "a * 2 * 3 * 4" gleichbedeutend mit "((a * 2) * 3) * 4" ist.

Definitionen (defn) von FUNCTION oder PROCEDURE beginnen mit dem Wort 'func' oder 'proc' gefolgt von einem Funktions/Prozedur Namen, optional mit einer Liste von benannten Argumenten in runde Klammern eingeschlossen, und schließen mit einem Statement ab. Benannte Argumente sind eine Referenz zum Typ ARG. Die folgenden, unterschiedlich definierten Funktionen, sind von der Funktionalität her gleich:

```
proc myproc(a,b,c) { print(a,b,c) }     // Benanntes Argument
proc myproc(a,b,c) { print($1,$2,$3) }  // ARG Typ
proc myproc()      { print($1,$2,$3) }  // ARG Typ
proc myproc(a,b,c) { print(a,$2,c) }    // gemischt
```

Wenn keine Argumentreferenzen benutzt werden, sind nur Argumenttypen $n erlaubt

Variablen in PSI

Zunächst gibt es in PSI einfache Variablen und Variablenfelder, sog. Arrays. Es sind nur eindimensionale Variablenfelder erlaubt. Auf die einzelnen Elemente eines Variablenfeldes greifen Sie über die Angabe eines "Indexwertes" zu. Das erste Element hat immer den Index 0, das letzte den Index n-1, wenn n die Anzahl der Elemente in dem Variablenfeld ist.

Beispiel: a[3] liefert das 4. Element des Variablenfeldes a.

Die Anzahl der Elemente eines Variablenfeldes können Sie mit dem "Dim"-Befehl feststellen:

Beispiel: anz = dim(a) belegt die Variable anz mit dem Wert der Anzahl der Elemente von a.

Variablen haben in PSI prinzipiell globale Gültigkeit, d.h. sobald Sie einer Variablen einen Wert zugewiesen haben, ist dieser im gesamten Programm verfügbar. Ausnahmen gibt es jedoch in Form lokaler Variablen, die nur innerhalb einer Prozedur/Funktion gültig sind:
wird in dem Unterprogramm eine Variable mit "***auto***" deklariert, so beziehen sich Zuweisungen und Referenzen auf die lokale Variable, auch wenn es "außerhalb" eine Variablen gleichen Namens gibt.
Variablen werden in PSI nicht deklariert; Daten welchen Typs sie beherbergen, ergibt sich aus dem Gebrauch der Variablen: wird ihr ein Zahlenwert zugewiesen, wird sie eine numerische Variable, wird ihr eine Zeichenkette zugewiesen, wird sie eine Zeichenkettenvariable. Diese Eigenschaft einer Variablen kann sich sogar während des Programmlaufs dynamisch ändern. Mit dem PSI-Befehl "***type***" können Sie herausbekommen, von welchem Typ eine Variable gerade ist. (s. Beispiele).
Eine einfache Variable kann zu einem Variablenfeld werden und umgekehrt. Alle Elemente eines Variablenfeldes müssen aber von einem Typ sein. Deshalb können Sie einem Element eines numerischen Variablenfeldes keine Zeichenkette zuweisen.

Beispiele:

```
type(n)                 // Ausgabe 0 (UNDEF)
n = 1                   // Variable wird numerisch
type(n)                 // Ausgabe 1 (NUM)
s = "strg"              // Variable wird Zeichenkette
type(s)                 // Ausgabe 2 (STR)
b[0] = n                // b wird ein numerisches Feld (n = 1)
type(b)                 // Ausgabe 3 (ANUM)
b[1] = 3                // OK, b ist ein numerisches Feld
dim(b)                  // Ausgabe 2 (2 Elemente in b)
c = b                   // kopiert Feld b in Feld c
type(c)                 // Ausgabe 3 (ANUM)
b[2] = s                // FEHLER! b ist ein numerisches Feld
b = 1                   // b ist nicht länger ein Feld
type(b)                 // Ausgabe 1 (NUM)
b[2] = s                // b wird ein Zeichenkettenfeld
type(b)                 // Ausgabe 4 (ASTR)
dim(b)                  // Ausgabe 3 (3 Elemente in b)
c = b                   // c wird ein Zeichenkettenfeld
type(c)                 // Ausgabe 4 (ASTR)
n                       // Ausgabe 1 (Wert von n)
s                       // Ausgabe "strg" (Wert von s)
func iseven(s) {        // liefert 1 wenn s gerade ist;
```

```
                        // Argument s ist für globales s
                        // unsichtbar!
     printl("glob ",n)  // Ausgabe des Wertes des globalen n
     auto n             // lokales n ist ab hier unsichtbar
                           für globales n
     n = s ^ 2 + 1      // Beispiel einer lokalen Variablen
                        // und ein benanntes Argument.
     printl("lok ",n)   // druckt den Wert der lokalen
                        // Variablen.
     if(($1%2) == 0) {  // ist s gerade ?? ($1 entspricht dem
                        // Argument)
         return 1       // ja -> iseven() liefert 1
         } else {       // dies muß in einer Zeile stehen!
         return 0       // nein -> iseven liefert 0
         }
     }                  // Ende der 'func' Definition
iseven(13)              // Ausgabe 0 (ungerade Zahl)
iseven(18)              // Ausgabe 1 (gerade Zahl)

n                              // Ausgabe 1 (globales n nicht
                               // verändert)
s                              // Ausgabe "strg" (globales s
                               //  nicht verändert)


// Definition der Fakultät (n!) mit rekursivem Aufruf
func fac(n) if (n <= 0) return 1 else return n * fac(n-1)
i = 0                           // Initialisieren einer
                                // Variablen
while(i < 10) {                 // solang i kleiner als 10
    printl(i,":\t",fac(i))      // Ausgabe der Fakultät von i
    i = i + 1                   // i erhöhen.
}
```

Funktionen der PSI Bibliothek

Zur Beschreibung der PSI-Bibliotheksfunktionen werden folgende Ausdrücke verwendet:

Xi	Typ beliebiger (i ist eine Ziffer oder nicht belegt)
Ni	Typ numerischer (i ist eine Ziffer oder nicht belegt)
Si	Typ Zeichenkette (i ist eine Ziffer oder nicht belegt)
Ai	Typ Zahlenfeld (i ist eine Ziffer oder nicht belegt)
Li	Typ Zeichenkettenfeld (i ist eine Ziffer oder nicht belegt)

Der Ausdruck "N func(N1,N2)" bedeutet beispielsweise, daß die Funktion 'func' als Parameter zwei numerische Typen erwartet und selber einen Wert vom Typ numerisch liefert. Ein "-"-Zeichen vor dem Namen der Bibliotheksfunktion bedeutet, daß kein Ergebniswert zurückgeliefert wird, es sich also um eine Prozedur handelt.

Der Ausdruck "stdin" bezeichnet die Standardeingabe des PSI-Interpreters. Zur Laufzeit eines PSI Programms ist dies die Tastatur.
Nachfolgend finden Sie eine Auflistung der in den PSI-Interpreter eingebauten Funktionen, gruppiert nach Funktionsklassen. Sie können diese Funktionen für Ihre eigenen PSI-Scripte verwenden.

Ein-/Ausgabe in Datei

Sie können maximal 10 Dateien gleichzeitig benutzen. Aus PSI-Ebene werden die verschiedenen Dateien über eine Indexnummer (Handle) angesprochen, die einen Wert von 0...9 annehmen kann. Diese Indexnummer ist gemeint, wenn im folgenden Text von "handle" gesprochen wird.

`- fclose(N1)`	Schließen der Datei mit dem `handle` N1.
`- fdelete(S1)`	Löscht die Datei `S1`.
`N feof(N1)`	Liefert 1 bei Erreichen des Dateiendes der Datei mit dem `handle N1` (Liefert 0 wenn das Dateiende nicht erreicht wurde).
`N ferror(N1)`	Liefert 1 wenn ein Fehler während des Schreibens/ Lesens der Datei mit dem `handle N1` auftrat (0 wenn kein Fehler auftrat).
`N fok(N1)`	Liefert 1 wenn das Dateiende noch nicht erreicht wurde und kein Fehler während des Schreibens/ Lesens der Datei mit dem `handle N1` auftrat (0 wenn ein Fehler auftrat). Die Funktion entspricht dem Kommando "`(!feof(N1) && !ferror(N1))`", ist aber schneller. Beispiel: Listen der Datei TEST.TXT (im Verzeichnis „bin“)

	`fp = fopen("TEST.TXT",IN)`
	`while(fok(fp)) print(fgets(fp))`
	`fclose(fp)`

`N fopen(S1,N1)`	Öffnen der Datei `S1` mit dem Zugriffstyp `N1`. Gültige Werte für `N1` sind hier: IN zum Einlesen OUT zum Schreiben EXT zum Erweitern (Anhängen an das Dateiende) UPD zum Lesen und Schreiben Der Rückgabewert (Datei-Index, kurz `handle`) muß in Funktionen, die diesen Index auswerten, benutzt werden. Die Datei muß spätestens zum Programmende geschlossen werden. Es können maximal 10 Dateien gleichzeitig benutzt werden.
`- fprint(X1,...,N1)`	Schreibt Argumente X1,... in die Datei mit dem `handle` N1.
`- fprintf (S1,X1,N1)`	Schreibt Argument X1 formatiert in die Datei mit dem `handle` N1. Die Formatbeschreibung zu S1 lesen Sie bitte im Abschnitt "Format Zeichenkette" nach.
`- fprintln(X1,...,N1)`	Wie `fprint()`, aber mit Zeilenvorschub am Ende.
`- rename(S1,S2)`	Umbenennen Datei `S1` nach `S2`
`S fgets (N1)`	Liest eine Zeichenkette aus der Datei mit dem `handle` N1.
`N fexist(S1)`	Liefert 1 (Wahr) falls die Datei mit dem Namen `S1` existiert, anderfalls 0.
`- fcopy (S1,S2)`	Kopieren Datei `S1` nach `S2`
`- mkdir (S1)`	Erzeugen Verzeichnis mit Namen `S1`
`L dir (S1)`	Listen Inhalt des Verzeichnisses `S1`

Ein-/Ausgaben (Konsole)

Mit diesen Bibliotheksfunktionen regeln Sie die Ausgaben auf den Bildschirm und die Eingaben von der Tastatur.

`- print(X1,...)`	Ausgabe der Argumente `X1,...` auf dem Bildschirm bzw. der Protokolldatei des Jobs.
`- printf(S1,X1)`	Formatierte Ausgabe des Arguments `X1` auf dem Bildschirm bzw. der Protokolldatei des Jobs. Die Formatbeschreibung zu `S1` lesen Sie bitte im Abschnitt "Format Zeichenkette" nach.
`- printl(X1,...) N`	Wie print(), aber mit Zeilenvorschub am Ende.
`N scan()`	Liest eine Zahl von stdin.
`S scans()`	Liest eine Zeichenkette von stdin.

Konvertierungsfunktionen

`N num(S1)`	ASCII Code des ersten Zeichens von `S1`.
`S str(N1)`	Liefert ASCII `N1` als Zeichenkette.
`N strtod(S1)`	Konvertiert eine Zeichenkette in einen Zahlenwert. Zulässiger Inhalt von `S1` entstammt der Zahlenmenge "0123456789".

Mathematische Funktionen

`N abs(N1)`	Absolutwert von `N1`.
`N acos(N1)`	Arc Cosinus von `N1` (mit: -1 <= N1 <= 1).
`N asin(N1)`	Arc Sinus von `N1` (mit: -1 <= N1 <= 1).
`N atan(N1)`	Arc Tangens von `N1`.
`N atan2(N1,N2)`	Arc Tangens von `N1/N2` (mit: N1 und N2 nicht 0).
`N cos(N1)`	Cosinus von `N1` (in Bogenmaß)
`N cosh(N1)`	Cosinus Hyperbolikus von `N1`.
`N exp(N1)`	Exponentialfunktion der Variable `N1`. (entspricht "E ^ N1", E ist global definiert.)
`N int(N1)`	Ganzzahliger Teil von `N1` (z.B. int(3.14) ist 3).
`N log(N1)`	Natürlicher Logarithmus von `N1` (mit: N1 >= 0).
`N log10(N1)`	Logarithmus zur Basis 10 von `N1` (mit: N1 >= 0).
`N sin(N1)`	Sinus von `N1` (im Bogenmaß).
`N sinh(N1)`	Sinus Hyperbolikus von `N1`.
`N sqrt(N1)`	Wurzel von `N1`.
`N tan(N1)`	Tangens von `N1` (im Bogenmaß).
`N tanh(N1)`	Tangens Hyperbolikus von `N1`.

Systemfunktionen

`- writeparm(S1,S2,S3)`	Ändert den Parameter `S2` im Block `S1` der Konfigurationsdatei `etc/alephino.cfg` auf den Wert `S3`. Falls der Parameter bereits existiert, wird der vorhandene Wert ersetzt; andernfalls wird ein neuer Eintrag angelegt. Beispiel: `writeparm("Communication","Port",4711)`
`S readparm(S1,S2)`	Gibt den Inhalt des Eintrages des Parameters `S2` im Block `S1` der Konfigurationsdatei `etc/alephino.cfg` zurück.
`- setup(N1,N2,N3)`	Einstellen der PSI Umgebung: Größe stack (N1),

	maximale Programmgröße (N2), max. Aufrufverschachtelung (N3). Wenn N1, N2, oder N3 den Wert 0 haben, werden voreingestellte Werte genommen. Diese sind: setup(256,2000,100)
`A sort(X1)`	Schreibt den Index des Feldes `X1` sortiert in `A`.
`N index(X1,X2)`	Gibt den Index des Feldes mit dem Schlüssel `X2` im Feld `X1` zurück. Wird der Schlüssel nicht gefunden, wird -1 zurückgeliefert.
`S getparm(S1)`	Liefert den Inhalt des Batch-Parameters `S1`.

Zeichenketten-Funktionen

`S sprint(X1,...)`	Schreibt ein oder mehrere Argumente `X1,..` in `S`.
`S sprintf(S1,X1)`	Schreibt ein Argument `X1` formatiert in Zeichenkette `S1` die in `S` zurückgegeben wird. Die Formatbeschreibung zu `S1` lesen Sie bitte im Abschnitt "Format-Zeichenkette" nach.
`S sprintl(X1,...)`	Wie `sprint()`, aber mit Zeilenvorschub am Ende.
`S strcat(S1,S2)`	Hintereinanderfügen von S1 und S2.
`N strcmp(S1,S2)`	Vergleichen zweier Zeichenketten. Liefert eine Zahl größer, gleich oder kleiner 0 in Abhängigkeit ob S1 größer, gleich oder kleiner S2 ist.
`N strlen(S1)`	Länge der Zeichenkette S1. Beispiel: `strlen("asdf")` liefert 4.
`S tolower(S1)`	Wandelt die Buchstaben der Zeichenkette S1 in Kleinbuchstaben und liefert das Ergebnis in der Zeichenkette S
`S toupper(S1)`	Wandelt die Buchstaben der Zeichenkette S1 in Großbuchstaben und liefert das Ergebnis in der Zeichenkette S.
`S strrpl(S1,S2,S3)`	Ersetzt in `S1` das erste Vorkommen von `S2` durch `S3` ersetzt wird. Liefert die modifizierte Zeichenkette zurück. Beispiel: `strrpl("Pi: &1","&1",sprint(PI))` liefert Zeichenkette `"Pi: 3.1415927".`
`N strstr(S1,S2)`	Die Position, an der `S2` als Teilstring von `S1` auftaucht. Liefert –1 wenn S2 nicht in S1 enthalten ist und einen Wert größer oder gleich 0 als Position des ersten Zeichens von S2 in S1. Beispiel: `strstr("asdfas","as")` liefert 0, `strstr("asdfas","df")` liefert 2, und `strstr("asdfas","qw")` liefert -1.

`S strsub(S1,N1,N2)`	Liefert einen Teilstring von `S1` beginnend an Position `N1` mit der Länge von `N2`. Wenn `N2` länger ist als `S1`, wird automatisch die Länge von `S1` angenommen. Beispiel: `strsub("asdfas",1,3)` liefert `"sdf"`, `strsub("asdfas",2,100)` liefert `"dfas"`.
`L splitline(S1,S2)`	Spaltet den String `S1` anhand des Separator-Strings `S2` auf. Liefert ein Feld mit den extrahierten Teilen zurück.
`L splitregex(S1,S2)`	Spaltet den String `S1` anhand des regulären Ausdrucks `S2` auf. Liefert ein Feld mit den extrahierten Teilen zurück.
`S transl(S1,S2)`	Konvertiert den Zeichensatz des Strings `S1` entsprechend der durch `S2` definierten Translate-Tabelle. Diese muss einem mit Label TRANSL= definierten Eintrag in der Server-Generierung entsprechen. Beispiel: `transl("Übung", "ISOTOEXT")` liefert „Übung“ in UTF-8-Kodierung (z.B. zum Schreiben in die Datenbank).

Zeit- und Datumsfunktionen

`N clock()`	Systemzeit in Sekunden.
`S date([S1])`	Systemdatum im Format "DD.MM.YYYY HH:MI:SS" zurück. Falls ein Parameter `S1` angegeben wurde, wird dieser als Formatzeichenkette bewertet.
`S datadd(S1,N1)`	Addiert (oder subtrahiert) die Anzahl von Tagen `N1` zu (von) Datum `S1`. `S1` muss im Format „YYYYMMDD“ angegeben werden.

Alephino Funktionen (Datenbank-Zugriff)

`N find(S1)`	Suche in der Alephino-Datenbank. Liefert die Nummer N eines Ergebnis-Sets zurück. `S1` besteht aus den Komponenten:
	`POOL=<Kürzel des Datenpools> (optional)`
	`FILE=<Kürzel der Stammdatei>`
	`QUERY=<Suchanfrage in CCL-Syntax>`
	
	Beispiele:
	`POOL=B FILE=BEN QUERY=NAM=Meier*`
	`Liefert alle Benutzer mit Namen Meier`
	
	`FILE=TIT QUERY=SWT=Wirtschaft and JHR=1995`

	Liefert alle Titel mit Schlagwort Wirtschaft und Erscheinungsjahr 1995
N get(S1)	Suche in der Alephino-Datenbank über Verknüpfung. Liefert die Nummer N eines Ergebnis-Sets zurück. S1 besteht aus den Komponenten: POOL=<Kürzel des Datenpools> (optional) FILE=<Kürzel der Stammdatei> IDN=<Identnummer des Ausgangssatzes der Verknüpfung> **oder:** SET=<Nummer eines Ergebnis-Sets NUMBER=<Index des Ausgangssatzes der Verknüpfung im Set (beginnend bei 1)> und: LINK=<Verknüpfungsaspekt, der vom Ausgangssatz zur gewünschten Ziel-Satzart verweist> Beispiele: POOL=B FILE=BEN IDN=123 LINK=VBU Liefert alle Verbuchungen des Benutzers mit der Identnummer 123. SET=5 NUMBER=1 LINK=TIT Angenommen, das Set Nr. 5 enthält Autorensätze, werden alle Titel des 1. Autors zurückgeliefert.
A getset(N1)	Liefert die Liste A der im Set N1 enthaltenen Identnummern.
N getrec(S1)	Lesen Datensatz. Liefert die Nummer N eines Datensatz-Handles oder einen (negativen) Fehlercode zurück. Bis zu 10 Datensätze können zugleich genutzt werden. S1 besteht aus den Komponenten: POOL=<Kürzel des Datenpools> (optional) FILE=<Kürzel der Stammdatei> IDN=<Identnummer des Datensatzes> **oder:** SET=<Nummer eines Ergebnis-Sets NUMBER=<Index des gewünschten Datensatzes im Set (beginnend bei 1)> Beispiele: POOL=B FILE=ORD IDN=8719 Liefert den Bestellsatz mit der Identnummer 8719. SET=8 NUMBER=7 Liefert den 7. Satz aus dem Ergebnis-Set Nr. 8. (Die Informationen über Datenpool und Stammdatei sind im Set enthalten.)
N imprec()	Liest den aktuellen Datensatz aus dem globalen Satzpuffer des Alephino Export/Import. Liefert die Nummer N eines Satz-Handles oder einen (negativen) Fehlercode zurück.
S getfld(N1,S1)	Liefert den Inhalt des Feldes mit dem Namen S1 aus dem Datensatz mit dem Handle N1.
- putfld(N1,S1,X1)	Schreibt die Zahl oder den String X1 in das Feld S1 des Satzes mit dem Handle N1.

- delfld(N1,S1)	Löscht das Feld S1 aus dem Satz mit dem Handle N1.
N addrec(N1,S1)	Legt einen neuen Datensatz mit dem Handle N1 an. Liefert die Identnummer des Satzes zurück. S1 enthält die Komponenten: POOL=<Kürzel des Datenpools> (optional) FILE=<Kürzel der Stammdatei>
- updrec(N1)	Schreibt den Datensatz mit dem Handle N1 in die Datenbank.
- delrec(N1)	Löscht den Datensatz mit dem Handle N1 aus der Datenbank.
- exprec(N1)	Schreibt den Datensatz mit dem Handle N1 in den globalen Satzpuffer für den Export/Import.
- freerec(N1)	Gibt den Datensatz (das Satz-Handle) N1 frei.
N initrec()	Gibt Satz-Handle N (für einen neuen Datensatz) zurück.

Die Format-Zeichenkette

Eine Format-Zeichenkette für sprintf(), fprintf() und printf() ist eine Zeichenkette, die druckbare Zeichen und eine Formatbeschreibung, bestehend aus verschiedenen Feldern - wie nachfolgend beschrieben enthält. (Felder in eckigen Klammern sind optional):

```
%[Marke][Min][.[Max]]Typ
```

Beschreibung der Felder in der Formatbeschreibung:

Feld: Bedeutung:

Marke Eins oder mehr dieser Zeichen -, +, # oder ein Leerzeichen (siehe unten).
Min Eine Zahl, die die Mindestanzahl der zum Ausdruck eines Wertes verwendeten Zeichen bestimmt.
Max Eine Zahl, die die Maximalanzahl der zum Ausdruck eines Wertes verwendeten Zeichen bestimmt.
Typ Ein Buchstabe, der den Typ der gedruckten Variablen beschreibt. (siehe unten)

Das Feld **Marke** der Formatbeschreibung:

Marke:	**Bedeutung**	**Voreinstellung**
-	Linksbündige Ausgabe	Rechtsbündige Ausgabe
+	Wenn die Ausgabe ein Zahlenwert ist, wird das Vorzeichen + oder - vor der Zahl ausgegeben.	Nur für eine negative Zahl wird ein - vor der Zahl ausgegeben.
<Leerzeichen>	Positive Zahlen werden linksbündig mit Leerzeichen	Es werden keine Leerzeichen ausgegeben.

	aufgefüllt.	
#	Wenn der Typ %e, %E oder %f ausgegeben wird, wird ein Dezimalpunkt mit ausgegeben. Der Typ %g oder %G wird mit Dezimalpunkt und abschließenden Nullen ausgegeben.	Ein Dezimalpunkt wird nur aus- gegeben wenn Zahlen folgen. Abschließende Nullen werden unterdrückt, ein Dezimalpunkt nur mit anschließenden Zahlen ausgegeben.

Das Feld **Typ** der Formatbeschreibung:

Typ	**Argument**	**Resultierendes Ausgabeformat**
e	Zahl	Vorzeichenbehafteter Wert im wissenschaftlichen Format Beispiel: -123.4567 wird als -1.234567e+002 ausgegeben wenn %e benutzt wird
f	Zahl	Vorzeichenbehafteter Wert, (VZ)(Zahlen).(Zahlen) Beispiel: -123.4567 wird als -123.456700 ausgegeben wenn %f benutzt wird
g	Zahl	Vorzeichenbehafteter Wert im Format %e oder %f. Es wird eine möglichst kurze Ausgabe nach dem mitgegebenen Maximum "Max" und dem Inhalt ausgegeben.
s	Zeichenkette	Ausgabe einer Zeichenkette

In allen Ausgabe Funktionen/Prozeduren, in denen das Ausgabeformat nicht explizit angegeben wurde, wird für Zahlen das Format "%.8g" und für Zeichenketten des Format "%s" benutzt.

Programmerstellung

Include-Mechanismus

Um häufig benötigte Funktionen zur Wiederverwendung in separaten Dateien vorhalten zu können, anstelle deren Quelltext in Ihr aktuelles PSI-Programm kopieren, enthält PSI eine Anweisung der Form:

```
#include <PSIsource.psi>
```

PSIsource.psi steht für den Namen der Include-Datei. Dieser versteht sich einschließlich ggfs. vorangestelltem Pfad, ist wahlfrei, darf jedoch nicht mehr als 100 Zeichen umfassen.
Das Label **#include** muß am Anfang der Zeile stehen, der Dateiname ist in Spitzklammern einzuschließen.

Programmstart

Als Startpunkt wird die Angabe eines Programmnamens (Prozedur oder Funktion) ohne Funktionskörper angenommen.
Üblicherweise existiert ein Funktionsaufruf **main()** am Ende des Quelltextes, der auf eine zuvor definierte gleichnamige Funktion verweist. Der Name main() für die Startprozedur ist jedoch nicht zwingend.

Programmieren von Alephino-Datenbankdiensten

Beispiel 1:

Ihnen liegt eine Excel-Datei (Format .csv) vor, die mit Semikolon separierte Personendaten enthält. Jede Zeile enthält jeweils die Personalnummer, den Vornamen, Nachnamen und das Geburtsdatum einer Person im Format TT.MM.JJJJ. Daraus sind Alephino Benutzerstammdaten zu erzeugen. Die Daten sind im Zeichensatz ISO Latin1 kodiert.

```
proc main() {

  print ("Wir lesen unsere csv-Datei ein...")

  fp = fopen("personen.csv", IN)

  if (!fok(fp)) {
    printl ("Datei konnte nicht gelesen werden")
  }

  recno = 1
  while (fok(fp)) {
     line = fgets(fp)
     columns = splitline(line, ";")
     maxcols = dim(columns)
     if (maxcols > 2) {
       printf("=============== Satz Nr.=%.f ==================", recno)
       recno = recno + 1
       name = transl(columns[2], "ISOTOEXT") + ", " + transl(columns[1],
"ISOTOEXT")
       record = initrec()
       putfld(record, "100", columns[0])
       putfld(record, "102", name)
       putfld(record, "105", strsub(columns[3], 6, 4) + strsub(columns[3], 3, 2)
+ strsub(columns[3], 0, 2))
      idn = addrec(record, "POOL=B FILE=BEN")
      print("Neuer Benutzer " +  name + sprintf(" mit IDN %.f", idn))
      freerec(record)
    }
  }
  fclose(fp)
}

main()
```

Unser Programm liefert mittels print() Ausgaben, die Sie anschließend im Job-Protokoll finden, beispielsweise:

```
JOB 000035 RUN_PSI 2013/10/16 15:56:14 START
2013/10/16 15:56:14 SCRIPT=beispiel1.psi SAVE=Y

Wir lesen unsere csv-Datei ein...
===============Satz Nr.=1==================
Neuer Benutzer Graca, Maria mit IDN 2
```

```
==============Satz Nr.=2=================
Neuer Benutzer Wurst, Hans mit IDN 3
==============Satz Nr.=3=================
Neuer Benutzer Blaubär, Käpt'n mit IDN 4
==============Satz Nr.=4=================
Neuer Benutzer Sorglos, Susi mit IDN 5
...

JOB 000035 RUN_PSI 2013/10/16 15:56:14 END
```

Beispiel 2:

Sie möchten ermitteln, wieviele Titel mit Schlagworten verknüpft sind, die das Wort „Wirtschaft“ beinhalten:

```
proc swtlink() {

  query = "FILE=SWT QUERY=SWT=Wirtschaft*"
  printl ("Ich suche in Alephino... " + query)
  setno = find(query)
  results = getset(setno)
  ix = dim(results)
  os = sprintf ("Set-Nummer = %.f", setno) + sprintf (" (%.f Saetze)", ix)
  printl(os)
  x = 1
  while (x <= ix) {
    query = sprintf("SET=%.f", setno) + sprintf(" NUMBER=%.f", x)
    handle = getrec(query)
    if (handle < 0) {
      printl("Error reading record - Stop!")
      x = ix + 1
    } else {
      linkedset = get(query + " LINK=TIT")
      lres = getset(linkedset)
      ax = dim(lres)
      print(sprintf("%.f)", x) + getfld(handle, "800") + sprintf(" (%.f Titel)",
ax))
    }
    freerec(handle)
    x = x + 1
  }
}

swtlink()
```

Die Ausgaben unseres Programms in das Job-Protokoll sind beispielsweise:

```
JOB 000039 RUN_PSI 2013/10/16 16:21:24 START
2013/10/16 16:21:24 SCRIPT=beispiel2.psi SAVE=Y

Ich suche in Alephino... FILE=SWT QUERY=SWT=Wirtschaft*
```

```
Set-Nummer = 99 (19 Saetze)

1)Wirtschaftlichkeitsrechnung (1 Titel)
2)Wirtschaftsprüfer (1 Titel)
3)Wirtschaftliche Lage (1 Titel)
4)Wirtschaftswissenschaftliches Studium (1 Titel)
5)Wirtschaftlichkeit (1 Titel)
6)Wirtschaftsstrafrecht (4 Titel)
7)Wirtschaftsrecht (4 Titel)
8)Wirtschaftsverwaltungsrecht (4 Titel)
9)Wirtschaftsinformatik (1 Titel)
10)Wirtschaftsmathematik (4 Titel)
11)Wirtschaftliches Verhalten (1 Titel)
12)Wirtschaftsethik (6 Titel)
13)Wirtschaft (3 Titel)
14)Wirtschaftsprüfung (6 Titel)
15)Wirtschaftspsychologie (5 Titel)
16)Wirtschaftswissenschaften (4 Titel)
17)Internationale Wirtschaftspolitik (1 Titel)
18)Wirtschaftspolitik (2 Titel)
19)Wirtschaftstheorie (4 Titel)

JOB 000039 RUN_PSI 2013/10/16 16:21:24 END
```