ExLibris Alephino

PSI – Scripts

Table of contents

The PSI programming language

With PSI (Program Script Interpreter), Alephino 5.0 offers a universal programming interface with a powerful range of functions. This is aimed at ambitious users who have already gained programming knowledge in a syntactically similar language such as C / C ++ or Perl. PSI can be used to create special reports, to analyze and edit data in the Alephino database file-wide according to self-defined rules and much more

PSI program syntax

The syntax of a PSI program (list) is described in the BNF format. Letters, characters, and words enclosed in apostrophe are terminal symbols and must be used in PSI programs in this spelling without an apostrophe. The terminal '\ n' means line feed and corresponds to the RETURN or ENTER key on the keyboard.)

```
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
```

An arbitrary number of spaces is permitted between terminal symbols to increase readability. If '\ n' is followed directly by a backslash (\), the line feed is ignored. The text after two slashes is evaluated as a comment until the end of the line. The keyword 'quit' leads to the immediate termination of the PSI program.

Syntax Description of terminal symbols not explained above:

NUMBER: Numeric constant according to the following scheme
(Expressions in square brackets are optional):

digits ['.'] [digits] [exp] [sign] [digits]
or
'.' [digits] [exp] [sign] [digits]

where:	digits	one or more decimal numbers ('0','1','2',...,'9')
	exp	exponent initiated with 'e' or 'E'
	sign	'+' or '-'

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

There are some ***predefined constants*** (you can respond with their name):

Name:	Value:	Meaning:
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	undefined type (VAR)
NUM	1	numeric type (VAR)
STR	2	string type (VAR)
ANUM	3	numeric array type (VAR)
ASTR	4	string array type (VAR)
IN	0	fopen() file descriptor read
OUT	1	fopen() file descriptor write
EXT	2	fopen() file descriptor extend(append)
UPD	3	fopen() file descriptor update

STRING: Characters enclosed in quotation marks ("). Some special characters with a leading backslash may be used in the string:

\b	Backspace
\f	Page feed
\n	Line feed
\r	Carriage return
\t	TAB

\\	Backslash
\"	Quotation marks (")
\ddd	Octal character code ddd (d can take the values '0','1',...,'7')

A string can be continued in the following line if this is initiated with a backslash (\).

VAR:
FUNCTION:
PROCEDURE:

Expressions beginning with a letter and containing letters or digits. Reserved expressions must not be used (if, else, while, return, auto, exit, proc, func, type, dim) and BLTINPROC or BLTINFUNC function or procedure names. It is case-sensitive ("big" is not equal to "Big"). You should avoid using a global expression in various VAR, PROCEDURE, or FUNCTION. Local variables (defined as 'auto' in stmt 'func' or 'proc') that have the same name as a global variable are hidden from the global variable.

Example:

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

Output: 2

ARG: Used to an unnamed parameter a 'func' or 'proc' to address.

Syntax: '$'digits

Example: $1
$12

ARG $n denotes the n-th parameter of the function / procedure call. The first parameter is $1.

Example:

```
func subtr() {
  return $1-$2
}                                  // Definition
a = subtr(2,5)                     // a becomes -3
printl(a)                          // Output from a
```

BLTINFUNC:
BLTINPROC:

One of the "built-in" functions from the PSI function library.
The available library functions are described in the appendix.
One distinguishes between *functions* that return a value as a function result, and *procedures* that return no value.

Explanation of the syntax description

What is the meaning of this very formal description of a PSI program that should only be immediately intelligible to computer scientists?

Let's look at first on how a ***PSI program*** (a "list") may be constructed:

- Empty (/ * nothing * /)
  or
- A list, followed by a line feed (list '\ n')
  or
- A list, followed by a definition and a line feed (list defn '\ n')
  or
- A list, followed by an assignment and a line feed (list asgn '\ n')
  or
- A list, followed by a statement and a line feed (list stmt '\ n')
  or
- A list, followed by an expression and a line feed (list expr '\ n').

Perhaps you have already noticed that a list - apart from the first rule - is always partly explained by "itself". This allows you to use one of the other descriptions for "list", which leads to a sequence of statements, assignments, etc., which finally make up the PSI program.

Now one has to look, whereby the other terms such as statement, allocation etc. are replaced, and then their building blocks again etc. Thus finally makes a PSI program.

An ***assignment*** (ASGN) can be:

- A variable gets a new value (VAR '=' expr) or
- An indexed array variable ("array") gets a new value (VAR '[' expr ']' '=' expr) or
- An argument gets a new value (ARG '=' expr) or
- An indexed argument field gets a new value (ARG '[' expr ']' '=' expr).

The index of a field begins at 0. Only one-dimensional fields are allowed.

A ***statement*** (stmt) can be:

- An expression (expr)
  or
- A return statement from a procedure ('return')
  or

- A return statement with a value, a function ('return' expr)
  or
- The exit from a PSI program ('exit')
  or
- An exit with a value, from a PSI program ('exit' expr)
  or
- The call of a user-defined PROCEDURE with possible arguments (PROCEDURE '(' arglist ')')
  or
- The calling of a library procedure with possible arguments (BLTINPROC '(' arglist ')') or
- The definition of local variables within a user defined PROCEDURE / FUNCTION ('auto' locals)
  or
- A 'while' command that executes a stmt as long as an expression is true (ie, not null) ('while' 'expr') stmt)
  or
- An 'if' command that executes a stmt if an expression is not null ('if' ('expr') 'stmt)
  or
- An 'if' command that executes the first stmt if an expression is not null and executes the second stmt if the expression is null ('if' 'expr' stmt 'else' stmt)
  or
- A statement list enclosed in braces ('{' stmtlist '}').

A ***statement list*** (stmtlist) can be:

- Empty (/ * nothing * /)
  or
- A statement list followed by a line feed (stmtlist '\ n')
  or
- A statement list followed by a statement (stmtlist stmt).

An ***expression*** (expr) may be:

- A numeric constant (NUMBER)
  or
- A string constant (STRING)
  or
- A variable with a value (VAR)
  or
- An indexed variable field with a value (VAR '[' expr ']')
  or
- A function that determines the type of a variable and returns UNDEF, NUM, STR, NUM, and ASTR ('type' ('VAR'))
  or
- A function that determines the size of a field and returns the number of field entries ('dim' '(' VAR ')')
  or
- An argument with the value (ARG)

or
- The indexed element of an argument field with a value (ARG '[' expr ']')
  or
- An assignment (asgn)
  or
- A function defined by the user which is called with arguments (FUNCTION '(' arglist ')')
  or
- A PSI library function called with arguments (BLTINFUNC '(' arglist ')')
  or
- An expression enclosed in brackets ('(' expr ')')
  or
- An arithmetic expression (see below)
  or
- A relational expression (see below)
  or
- A logical expression (see below).

An ***arithmetic expression*** may be:

- Add numbers (expr '+' expr)
  or
- Linking string strings (expr '+' expr)
  or
- Subtract (expr '-' expr)
  or
- Multiply (expr '*' expr)
  or
- Dividing (expr '/' expr)
  or
- Modulo (expr '%' expr)
  or
- (Expr 'expr)
  or
- For example, UNARYMINUS ('-' expr)
  or
- Without effect, eg UNARYPLUS ('+' expr).

Note:
The expressions "S1 + S2" and "strcat (S1, S2)" have the same effect when S1 and S2 are both character strings. Other arithmetic operators accept only numeric expressions.

A ***relational expression*** is true (1) if:

- The first is greater than the second (expr 'expr)
  or
- The first is greater than or equal to the two (expr '> =' expr)
  or
- The first is less than the second (expr 'expr)

or
- The first is small or equal to the second (expr '<=' expr)
  or
- The first is equal to the second (expr '==' expr)
  or
- The first is not equal to the second (expr '! =' Expr)

And 0 (FALSE) if not.

Note:
The expressions "S1 RELOP S2" and "strcmp (S1, S2) RELOP 0" produce identical results when S1 and S2 are both string strings, and the RELOP is one of these relational operators:

'>', '>=', '<', '<=', '==', or '!='.

A ***logical expression*** is 1 (TRUE) if:
- The first and second is not 0 (expr '&&' expr)
  or
- The first or the second is not 0 (expr '||' expr)
  or
- The expression 0 is ('!' Expr).

And 0 (FALSE) if not.

Note: Logical operators only allow numeric expressions as arguments.

The ***ranking*** of arithmetic, relational and logical ***operators:***

Associativity:	Operator:
Right to left	^
Left to right	UNARYMINUS UNARYPLUS!
from left to right	* / %
Left to right	+ -
Left to right	> >= < <= == !=
from left to right	&&
Left to right	||
Right to left	= (assignment)

Associativity "right to left" means,
That "a ^ 2 ^ 3 ^ 4" equals "a ^ (2 ^ (3 ^ 4))".

Associativity "left to right" means,
That "a * 2 * 3 * 4" is equal to "((a * 2) * 3) * 4".

Definitions (defn) FUNCTION or PROCEDURE begin with the word 'func' or 'proc' followed by a function / procedure name, optionally included with a list of named arguments in parentheses, and close with a statement from. Named arguments are a reference to type ARG. The following, differently defined functions are the same from the functionality:

```
(a, b, c)} // Named argument
Proc myproc (a, b, c) {print ($ 1, $ 2, $ 3)} // ARG type
Proc myproc () {print ($ 1, $ 2, $ 3)} // ARG type
(a, b, c) {print (a, $ 2, c)} //
```

If no argument references are used, only argument types $ n are allowed

Variables in PSI

First, in PSI, there are simple variables and variable fields, so-called arrays. Only one-dimensional variable fields are allowed. You access the individual elements of a variable field by specifying an "index value". The first element always has the index 0, the last the index n-1 if n is the number of elements in the variable field.

Example: a[3] returns the 4th element of the variable field a.

You can determine the number of elements of a variable field with the "Dim" command:

Example: anz = dim(a) uses the variable anz with the value of the number of elements of a.

Variables are generally global in PSI, so if you assign a value to a variable, it is available in the entire program. Exceptions are, however, in the form of local variables that are valid only within a procedure / function:
is in the subroutine a variable declared using ***"auto",*** then assignments and references refer to the local variable, even if it is "outside" is a variable of the same name.
Variables are not declared in PSI; Data of which type they are accommodated results from the use of the variable: if it is assigned a numerical value, it becomes a numeric variable; if it is assigned a string, it becomes a string variable. This property of a variable can change dynamically even during program execution. With the PSI-command ***"type"*** you can find out, what type a variable is just. (See examples).
A simple variable can become a variable field and vice versa. However, all elements of a variable field must be of one type. Therefore, you can not assign a string to an element of a numeric variable field.

Examples:

```
type(n)                 // output 0 (UNDEF)
n = 1                   // variable becomes numeric
type(n)                 // output 1 (NUM)
s = "strg"              // variable becomes string
type(s)                 // output 2 (STR)
b[0] = n                // b becomes numeric array (n = 1)
type(b)                 // output 3 (ANUM)
b[1] = 3                // OK, b is numeric array
dim(b)                  // Output 2 (2 elements in b)
c = b                   // copies array b into array c
type(c)                 // output 3 (ANUM)
b[2] = s                // ERROR! b is numeric array
b = 1                   // b is not longer an array
type(b)                 // output 1 (NUM)
b[2] = s                // b becomes string array
type(b)                 // output 4 (ASTR)
dim(b)                  // output 3 (3 elements in b)
c = b                   // c becomes string array
type(c)                 // output 4 (ASTR)
n                       // output 1 (value of n)
s                       // output "strg" (value of s)
func iseven(s) {        // output 1 if s is even;
                        // argument s is invisible for global s!
```

```
     printl("glob ",n)  // output of value of the global n
     auto n             // local n is from here invisible for global n
     n = s ^ 2 + 1      // example for local variable and named argument
     printl("lok ",n)   // print value of local variable
     if(($1%2) == 0) {  // is s even ?? ($1 corresponds to the argument)
         return 1       // yes -> iseven() returns 1
         } else {       // this must be in the same line!
         return 0       // no -> iseven returns 0
         }
     }                  // end of 'func' definition
iseven(13)              // output 0 (uneven number)
iseven(18)              // output 1 (even number)

n                              // output 1 (global n not modified)
s                              // output "strg" (global s not modified)


// Definition of the faculty (n!) With recursive call
func fac(n) if (n <= 0) return 1 else return n * fac(n-1)
i = 0                           // initialization of a variable

while(i < 10) {                 // as long as i is less than 10
    printl(i,":\t",fac(i))      // print faculty of i
    i = i + 1                   // increment i by 1.
}
```

Functions of the PSI library

The following expressions are used to describe the PSI library functions:

Xi	Type of any (i is a digit or not used)
Ni	Type numeric (i is a digit or not used)
Si	Type string (i is a digit or not used)
Ai	Type number array (i is a digit or not used)
Li	Type string array (i is a digit or not used)

For example, the expression "N func (N1, N2)" means that the func function expects two numeric types as parameters, and provides a value of the type numerically. A "-" sign before the name of the library function means that no result value is returned, ie it is a procedure.

The term "stdin" denotes the standard input of the PSI interpreter. At runtime of a PSI program, this is the keyboard.
The following is a list of the functions built into the PSI interpreter, grouped by function classes. You can use these features for your own PSI scripts.

I / O to file

You can use up to 10 files at a time. From the PSI level, the various files are accessed via an index number (handle), which can have a value of 0 ... 9. This index number is meant when "handle" is used in the following text.

`- fclose (N1)`	Close the file with the `handle` N1.
`- fdelete (S1)`	Deletes the `S1` file.
`N feof (N1)`	Returns 1 at end of the file of the file with the `handle N1` (Returns 0 if the end of file has not been reached).
`N ferror (N1)`	Returns 1 if an error occurs during the writing / reading of the file occurred with the `handle N1` (0 if no error occurred).

`N fok (N1)`	Returns 1 if the end of file has not been reached and no error during the writing / reading of the file with the `handle N1` occurred (0 if an error occurred). The function corresponds to the command "(! `feof (N1) &&! ferror` (N1))", but is faster.
	
	Example: Lists of the file TEST.TXT (in the directory "bin")
	`fp = fopen ("TEST.TXT", IN)`
	`while (fok (fp)) print (fgets (fp))`
	`fclose (fp)`

`N fopen (S1, N1)`	Open the file `S1` with the access type `N1`. Valid values for `N1` are here: IN for reading OUT for writing EXT to expand (append to file end) UPD for reading and writing The return value (file index, short `handle`) must be in functions that evaluate this index used. The file must be closed at the end of the program at the latest. A maximum of 10 files can be used simultaneously.
`- fprint (X1, ..., N1)`	Writes arguments X1, ... in the file with the `handle` N1.
`- fprintf (S1, X1, N1)`	Writes argument X1 formatted in the file with the `handle`

	N1. The format description for S1 can be found in the "Format character string" section.
- fprintln (X1, ..., N1)	Like fprint (), but with line feed at the end.
- rename (S1, S2)	Rename file S1 to S2
S fgets (N1)	Reads a string from the file with the handle N1.
N fexist (S1)	Returns 1 (true) if the file exists named S1, the other if 0th
- fcopy (S1, S2)	Copy file S1 to S2
mkdir (S1)	Generating directory named S1
L dir (S1)	Lists contents of the S1

Input / Output (console)

With these library functions, you control the output on the screen and the input from the keyboard.

- print (X1, ...)	Output of the arguments X1, ... on the screen or the log file of the job.
- printf (S1, X1)	Formatted output of the argument X1 on the screen or the log file of the job. The format description to S1, refer to the section "Format String" after.
- printl (X1, ...) N	Like print (), but with line feed at the end.
N scan ()	Reads a number of stdin.
S scans ()	Reads a string of stdin.

Conversion functions

`N num (S1)`	ASCII code of the first character of `S1`.
`S str (N1)`	Returns ASCII `N1` as a string.
`N strtod (S1)`	Converts a string to a numeric value. Permissible content of `S1` comes from the set of numbers "0123456789".

Mathematical functions

`N abs (N1)`	Absolute value of `N1`.
`N acos (N1)`	Arc cosine of `N1` (with -1 <= N1 <= 1).
`N asin (N1)`	Arc sine of `N1` (with -1 <= N1 <= 1).
`N atan (N1)`	Arc tangent of `N1`.
`N atan2 (N1, N2)`	Arc tangent of `N1 / N2` (with N1 and N2 is not 0).
`N cos (N1)`	Cosine of `N1` (in radians)
`N cosh (N1)`	Cosine hyperbolic of `N1`.
`N exp (N1)`	Exponential function of `N1` variable. (Corresponds to "E ^ N1", E is defined globally.)
`N int (N1)`	Integer part of `N1` (eg int (3.14) 3).
`N log (N1)`	Natural logarithm of `N1` (with: N1> = 0).
`N log10 (N1)`	Logarithm to the base 10 of `N1` (with N1> = 0).
`N sin (N1)`	Sine of `N1` (in radians).

N sinh (N1)	Hyperbolic sine of N1.
N sqrt (N1)	Squareroot of N1.
N tan (N1)	Tangent of N1 (in radians).
N tanh (N1)	Hyperbolic tangent of N1.

System functions

- writeparm (S1, S2, S3)	Changes the parameters in S2 S1 block the configuration file etc / alephino.cfg to the value S3. If the parameter already exists, the existing value is replaced; Otherwise a new entry is created. Example: Writeparm ("Communication", "Port", 4711)
S readparm (S1, S2)	Returns the contents of the entry of the parameter S2 in block S1 of the configuration file etc / alephino.cfg back.
- setup (N1, N2, N3)	Setting the PSI environment: Size stack (N1), Maximum program size (N2), Max. Call interleaving (N3). If N1, N2, or N3 are set to 0, preset values are taken. These are: setup (256,2000,100)
A sort (X1)	Writes the index of the field X1 ranked in A.
N index (X1, X2)	Returns the index of the field with the key X2 back in box X1. If the key is not found, -1 is returned.
S getparm (S1)	Supplies contents of the batch parameter S1.

String functions

`S sprint (X1, ...)`	Writes one or more arguments `X1, ..` in `S.`
`S sprintf (S1, X1)`	Writes an argument `X1` formatted string `S1` is returned in `S.` The format description to `S1,` refer to the section "format string" after.
`S sprintl (X1, ...)`	Like `sprint (),` but with line feed at the end.
`S strcat (S1, S2)`	Insertion of S1 and S2.
`N strcmp (S1, S2)`	Compare two string strings. Returns a number greater than or equal to 0 depending on whether S1 is greater than or equal to S2.
`N strlen (S1)`	Length of the string S1. Example: `strlen ( "asdf")` returns. 4
`S tolower (S1)`	Converts the letters of the string S1 into lowercase letters and returns the result in the string S
`S toupper (S 1)`	Converts the characters of the string S1 to uppercase and returns the result in the string S.
`S strrpl (S1, S2, S3)`	Replaced in `S1` the first occurrence of `S2` is replaced by `S3.` Returns the modified string. Example: `strrpl ( "Pi: & 1", "& 1", sprint (PI))` returns string `"Pi: 3.1415927".`
`N strstr (S1, S2)`	The position appears on the `S2` as a substring of `S1.` Returns -1 if S2 is not contained in S1 and has a value greater than or equal to 0 as the position of the first character of S2 in S1. Example: `strstr ( "asdfas", "as")` returns 0 `strstr ( "asdfas", "df")` returns 2, and `strstr ( "asdfas", "qw")` returns -1.
`S strsub (S1, N1, N2)`	Returns a substring of `S1` starting at position `N1` having the

	length of N2. If N2 is longer than S1, automatically the length of S1 is assumed. Example: strsub ("asdfas", 1,3) returns "sdf" strsub ("asdfas", 2,100) delivers "dfas".
L split line (S1, S2)	Splits the string S1 on the basis of the separator string S2. Returns a field with the extracted parts.
L splitregex (S1, S2)	Splits the string S1 on the basis of the regular expression S2. Returns a field with the extracted parts.
S transl (S1, S2)	Converts the character set of the string S1 according to the Translate table defined by S2. This must correspond to an entry defined in label TRANSL = in the server generation. Example: transl ("exercise", "ISOTOEXT") returns "exercise" in UTF-8 encoding (eg for writing into the database).

Time and date functions

N clock ()	System time in seconds.
S date ([S1])	System date in the format "DD.MM.YYYY HH: MI: SS". If a parameter is specified S1, this is rated as a format string.
S datadd (S1, N1)	Add (or subtract) the number of days N1 to (from) Date S1. S1 must be in the format "YYYYMMDD".

Alephino functions (database access)

N find (S1)	Search in the Alephino database. Returns the number N of result sets. S1 comprises the following components: POOL= <Data pool shortcut> (optional) FILE= <abbreviation of master file> QUERY= <Query in CCL syntax> Examples:

	POOL=B FILE=BEN QUERY=NAM=Meier * Returns all users named Meier FILE=TIT QUERY=SWT=Economy and JHR=1995 Provides all titles with keyword economy and year of publication 1995
N get (S1)	Search in the Alephino database via link between tables. Returns the number N of result sets. S1 comprises the following components: POOL= <Data pool shortcut> (optional) FILE= <abbreviation of master file> IDN= <Ident number of the link's source record> **or:** SET= <number of a result set NUMBER= <index of the link's source record in the set (starting at 1)> and: LINK= <link aspect that points from the source to the desired target record type> Examples: POOL=B FILE=BEN IDN=123 LINK=VBU Returns all active loans of the user with ID number 123. SET=5 NUMBER=2 LINK=TIT Suppose the set no. 5 contains author sets, all titles of the 2nd author are returned.
A getset (N1)	Returns the list A of the ID numbers contained in the set N1.
N getrec (S1)	Read record. Returns the number N of record-handles or a (negative) error code. Up to 10 records can be used at the same time. S1 comprises the following components: POOL= <abbreviation of the data pool> (optional) FILE= <abbreviation of the master file> IDN= <Ident number of the record> **or:** SET= <number of a result set NUMBER= <index of the desired record in the set (starting at 1)> Examples: POOL=B FILE=ORD IDN=8719 Returns the order number with ID number 8719. SET=8 NUMBER=7

	Returns the 7th set from the result set #8 (The information about the data pool and the master file are included in the set.)
`N imprec ()`	Reads the current record from the global record buffer of the Alephino Export / Import. Returns the number N of a set of handles or a (negative) error code.
`S getfld (N1, S1)`	Returns the contents of the field named `S1` from the data set with the handle `N1`.
`- putfld (N1, S1, X1)`	Writes the number or string `X1` in the field `S1` of the set with the handle `N1`.
`- delfld (N1, S1)`	`S1` Deletes the field from the set with the handle `N1`.

`N addrec (N1, S1)`	New record with handle N1 will be added to the database. Returns the ident number of the record. `S1` contains the components:
	`POOL= <Data pool shortcut> (optional)`
	`FILE= <abbreviation of the master file>`

`- updrec (N1)`	Writes the record with the handle `N1` in the database.
`- delrec (N1)`	Deletes the record with the handle `N1` from the database.
`- exprec (N1)`	Writes the record with handle N1 to the global record buffer for the export / import.
`- freerec (N1)`	Releases the record (the record handle) `N1`.
`N initrec ()`	Allocates a record handle (for a new record).

The format string

A format string for sprintf (), fprintf (), and printf () is a string containing printable characters and a format description consisting of different fields as described below. (Fields in square brackets are optional):

```
%[Mark][Min][.[Max]]Type
```

Description of the fields in the format description:

Field:	Meaning:
Mark	One or more of these characters -, +, #, or a space (see below).
Min	A number representing the minimum number of expressions value.
Max	A number representing the maximum number of the expression value.
Type	A letter that describes the type of the printed variable. (see below)

The field **mark** of Format Description:

Mark:	**importance**	**Default setting**
-	Left-aligned edition	Right-aligned edition
+	If the output is a numeric value, the sign + or - is output before the number.	Only for a negative number will be output - before the number.
<Space>	Positive numbers are filled with spaces on the left-hand side.	No spaces are output.
#	If the type %e, %E or %f is output, a decimal point is output. The type %g or %G is output with decimal point and terminating zeros.	A decimal point is only issued if numbers follow. Concluding zeros are suppressed, a decimal point is only output with subsequent numbers.

The field **type** of format description:

Type	**argument**	**Resulting output format**
e	number	Signed value in scientific format Example: -123.4567 is output as -1.234567e + 002 when %e is used
f	number	Signed value, (sign) (numbers). (Numbers) Example: -123.4567 is output as -123.456700 when %f is used

g	number	Signed value in the format% e or% f. The output as short as possible is output after the maximum "Max" and the contents.
s	String	Output of a string

In all output functions / procedures where the output format has not been explicitly specified, the format "% .8g" is used for numbers and the format "%s" is used.

Program creation

Include mechanism

To provide frequently used functions for reuse in separate files, instead of copying their source text into your current PSI program, PSI contains a statement of the form:

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

PSIsource.psi is the name of the include file. This is understood, including, if necessary, the preceding path, is optional, but may not contain more than 100 characters.
The label **#include** must be at the beginning of the line, the filename must be enclosed in angle brackets.

Program start

The starting point is the specification of a program name (procedure or function) without function bodies.
Usually a function call **main ()** exist at the end of the source code, which refers to a previously defined function with the same. However, the name main () for the start procedure is not mandatory.

Programming Alephino database services

Example 1:

Assumed you have an Excel file (format .csv) that contains person data separated by semicolons. Each line contains the personnel number, the first name, last name and the date of birth of a person in the format DD.MM.YYYY. From this, Alephino user master data are to be generated. The data is encoded in the ISO Latin1 character set.

```
proc main() {

  print ("We are reading our csv-file...")

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

  if (!fok(fp)) {
    printl ("Unable to read file")
  }

  recno = 1
  while (fok(fp)) {
     line = fgets(fp)
     columns = splitline(line, ";")
     maxcols = dim(columns)
     if (maxcols > 2) {
       printf("============== Record No.=%.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("Adding user " +  name + sprintf(" with IDN %.f", idn))
      freerec(record)
    }
  }
  fclose(fp)
}

main()
```

Our program uses print () output, which you then find in the job log, for example:

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

We are reading our csv-file...
============== Record No.=1 =================
New user Graca, Maria with IDN 2
============== Record No.=2 =================
New user Wurst, Hans with IDN 3
```

```
============== Record No.=3 ==================
New user Blaubär, Käpt'n with IDN 4
============== Record No.=4 ==================
New user Sorglos, Susi with IDN 5
...

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

Example 2:

This serves to determine how many items are associated with subjects containing the word "econom":

```
proc swtlink() {

  query = "FILE=SWT QUERY=SWT=econom*"
  printl ("Looking for... " + query)
  setno = find(query)
  results = getset(setno)
  ix = dim(results)
  os = sprintf ("Set-Number = %.f", setno) + sprintf (" (%.f Records)", 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 Title)",
ax))
    }
    freerec(handle)
    x = x + 1
  }
}

swtlink()
```

The output of our program into the job protocol is expected to look like:

```
JOB 000039 RUN_PSI 2013/10/16 16:21:24 START
2013/10/16 16:21:24 SCRIPT=example2.psi SAVE=Y

Looking for... FILE=SWT QUERY=SWT=econom*

Set-Number = 99 (9 Records)
```

```
1) economic calculation (1 title)
2) economic situation (1 title)
3) economic studies or (1 title)
4) economic computer science (1 title)
5) economic behavior (1 title)
6) economics (4 titles)
7) international Economic Policy (1 title)
8) economic policy (2 titles)
9) economic Theory (4 titles)
JOB 000039 RUN_PSI 10/16/2013 16:21:24 END
```