Bartels User Language®
Programmer's Guide

Bartels User Language Programmer's Guide
Published by: Bartels System GmbH, Munich
Last printing: November 2013

The information contained in the **Bartels AutoEngineer** publications as well as the products and/or programs described therein are subject to change without notice and should not be construed as a commitment by Bartels System.

Although Bartels System has gone to great effort to verify the integrity of the information provided with the **Bartels AutoEngineer** publications, these publications could contain technical inaccuracies or typographical errors. Bartels System shall not be liable for errors contained therein or for incidental consequential damages in connection with the furnishing, performance or use of this material. Bartels System appreciates readers' and/or users' comments in order to improve these publications and/or the products described therein. Changes are periodically made to the information therein. These changes will be incorporated in new editions of the **Bartels AutoEngineer** publications.

Preface

The Bartels User Language - Programmer's Guide describes how to use the **Bartels User Language** in **Bartels AutoEngineer**, i.e., how it is integrated to the **Bartels AutoEngineer** EDA system and how it can be applied. The following main topics are covered by this manual:

- basic concepts and description of the **Bartels User Language**
- the **Bartels User Language** programming system: **User Language Compiler** and **User Language Interpreter**
- **User Language** example source code listings, short information on the **User Language** programs supplied with the **Bartels AutoEngineer**
- special data types defined for accessing **Bartels AutoEngineer** design data
- **User Language** system function reference

This documentation is intended for the users of the **Bartels AutoEngineer** CAD software. The reader should be familiar with the use of his operating system, a text editor for generating ASCII files on his system and the **Bartels AutoEngineer**. Moderate experience with programming languages in general and the C programming language in particularity is recommended as well.

Those BAE users not planning to do any **User Language** programming should at least have a look to chapter 4 of this manual, where all **User Language** programs provided with the BAE software are listed with short descriptions.

Kindly note the Copyright before making use of the information provided with this documentation or applying the herein described products. The reader should also be familiar with the Notations and Conventions used throughout this documentation.

Organization of this Documentation

Chapter 1 introduces the basic concepts of the **Bartels User Language**.

Chapter 2 describes in detail the definition of the **Bartels User Language** and explains how to write **User Language** application programs.

Chapter 3 describes the **Bartels User Language** programming system. It explains how to compile **User Language** programs using the **Bartels User Language Compiler**, and how to run **User Language** programs using the **Bartels User Language Interpreter**.

Chapter 4 lists all of the **Bartels AutoEngineer User Language** include files and **User Language** programs with short descriptions, and provides information on how to make the programs available to the BAE software.

Appendix A describes the conventions and the valid parameter value ranges for accessing the **User Language** index variable types and system functions.

Appendix B describes the index variable type definitions of the **Bartels User Language**.

Appendix C describes the system functions included with the **Bartels User Language**.

Related Documentation

The Bartels AutoEngineer® - Installation Guide describes the **Bartels AutoEngineer** configurations and system requirements and provides detailed **Bartels AutoEngineer** installation instructions for all supported hardware and software platforms.

The Bartels AutoEngineer® - User Manual describes in detail how to use the **Bartels AutoEngineer** CAE/CAD/CAM design system. The following main topics are covered by this manual:

- Introduction: System Architecture, general Operating Instructions, Design Database
- Circuit Design (CAE), **Schematic Editor**
- Netlist Processing, Forward and Backward Annotation
- PCB Design and manufacturing data processing (CAD/CAM), **Layout Editor**, **Autoplacement**, Autorouting, **CAM Processor**, **CAM View**
- IC/ASIC Design, **Chipeditor** for interactive IC mask layout, **Cell Placer** and **Cell Router** for place & route, GDS and CIF import and export
- Neural Rule System
- Utility Programs

The Bartels AutoEngineer® - Symbol and Part Libraries documentation contains detailed information about the symbol and part libraries provided with the **Bartels AutoEngineer** CAE/CAD/CAM design system.

Problems, Questions, Suggestions

We appreciate comments from the people who use our system. In particular we are thankful for suggestions on how to improve the **Bartels AutoEngineer** and/or the **Bartels User Language** by introducing new or improving existing functions. Please do not hesitate to contact Bartels Support if you have questions or problems related to the use of the **Bartels User Language**. Check the Bartels Website at http://www.bartels.de for our address.

Documentation Notations

The reader should be familiar with the following notations used throughout the **Bartels AutoEngineer** documentation:

Mouse	pointing device (mouse, trackball, etc.) to be used for moving the menu and graphic cursors as well as for selecting functions
Info Field	menu field on the right top of the screen for displaying system status messages
Main Menu	function menu permanently available in the upper right screen area used for selecting a subordinate function menu
Menu	function menu in the lower right screen area currently selected from main menu
Submenu	subordinate function menu in the lower right screen area activated intermediately whilst using another menu function
Graphic Workarea	workarea for graphic interaction in the upper left screen area
Status Line	lower left screen line used for displaying system status messages and for performing interactive user queries
Menu Cursor	rectangle-shaped cursor for selecting a menu function
Graphic Cursor	cross-shaped cursor in the graphic workarea (crosshairs)
Menu Prompt	user query in the status line
Popup Menu	menu optionally displayed on top of the graphic workarea for selecting function-specific objects or for activating menu-specific functions
Button	selectable popup menu entry for choosing a certain menu element or for activating a menu-specific function
Select Function	move menu cursor to a function of the currently active function menu
Activate	hitting the mouse button
Pick	select an object to be manipulated using the graphic cursor
Place	move an element to a certain position in the graphic workarea
Select	select an element to be manipulated or a function to be executed by pressing the mouse button
Confirm	confirm the execution of a function verified by user query

The following acronyms are used throughout the **Bartels AutoEngineer** documentation:

BAE	acronym for identifying the **Bartels AutoEngineer** EDA software
BAEICD	acronym for the **Bartels AutoEngineer** IC/ASIC design system optionally included with workstation-based BAE configurations
SCM	acronym for the **Schematic Editor** program module of the **Bartels AutoEngineer** circuit design system
GED	acronym for the graphical PCB **Layout Editor** program module of the **Bartels AutoEngineer** PCB design system
AP	acronym for the **Autoplacement** program module of the **Bartels AutoEngineer** PCB design system
AR	acronym for the **Autorouter** program module of the **Bartels AutoEngineer** PCB design system
NAR	acronym for the advanced **Neural Autorouter** program module of the **Bartels AutoEngineer** PCB design system
CAM	acronym for the **CAM Processor** program module of the **Bartels AutoEngineer** PCB design system
CV	acronym for the **CAM View** program module of the **Bartels AutoEngineer** PCB design system
CED	acronym for the **Chip Editor** program module of the **Bartels AutoEngineer** IC/ASIC design system
CP	acronym for the **Cell Placement** program module of the **Bartels AutoEngineer** IC/ASIC design system
CR	acronym for the **Cell Router** program module of the **Bartels AutoEngineer** IC/ASIC design system
UL	acronym for the **Bartels User Language** programming language
ULC	acronym for the **Bartels User Language Compiler**
ULI	acronym for the **Bartels User Language Interpreter**

Documentation Conventions

Unless otherwise mentioned, the following symbolic conventions are used throughout the **Bartels AutoEngineer** documentation:

`Lineprint`	Lineprint font represents text output generated by the system.
Boldface	Boldfaced words or characters in format or command descriptions represent topic definitions or syntactic terminals, i.e., commands or keywords to be inserted directly.
Emphasize	Emphasized text is used for optical accentuation.
" "	Double quotes denote names and/or path names or enclose characters and/or character sequences directly to be inserted.
[ ]	Square brackets enclose optional items in format or command descriptions.
{ }	Braces enclose a list of items in format or command description, from which one has to be chosen.
|	A vertical bar separates items in a list of choices.
< >	Angle brackets enclose the logical name of a key on the keyboard. In format or command descriptions, angle brackets enclose values to be supplied.
>	Boldfaced greater signs in lineprint font are used for denoting prompts on operating system level.
...	Horizontal ellipsis points indicate either optional repetition of the preceding element in format or command descriptions or absence of irrelevant parts of a figure or example.
:	Vertical ellipsis points indicate absence of irrelevant parts of a figure, an example or a format or command description.
	Any Mouse Button (MB)
	Left Mouse Button (LMB)
	Middle Mouse Button (MMB)
	Right Mouse Button (RMB)
	Keyboard (input) - Return/Enter key (CR)
a **b** ...	Keyboard (input) - standard key(s)
F1 **F2** ...	Keyboard (input) - function key(s)
`filename`	File or directory path name.
`keyword`	Topic definitions or syntactic terminals, i.e., commands or keywords to be inserted directly.
`message`	BAE/system status or error message display.
Menu	**Bartels AutoEngineer** menu.
Menu Function	**Bartels AutoEngineer** menu function.
Menu Option	**Bartels AutoEngineer** menu option.
Button	**Bartels AutoEngineer** (popup) menu button.
ul.ulh	(Hypertext link to) **Bartels User Language** include file.
ULPROG	(Hypertext link to) **Bartels User Language** program description.
ul.ulc	(Hypertext link to) **Bartels User Language** program source file.
ul_function	(Hypertext link to) **Bartels User Language** system function description.
UL_INDEX	(Hypertext link to) **Bartels User Language** index type description.

UTILPROG	(Hypertext link to) **Bartels AutoEngineer** utility program description.
new feature	New features which are made available with regular (weekly) software updates/builds are highlighted in the online documentation.

The character sequences mentioned above may regain original meaning when used in programming languages, interpreter languages, specification languages, syntax description languages, etc.

Contents

Tables

Chapter 1
Introduction

This chapter introduces the basic concepts of the **Bartels User Language**. It describes the purpose of the **Bartels User Language**, provides remarks on the characteristics of this programming language, and introduces the Compiler and the Interpreter of the **Bartels User Language** programming system.

Contents

1.1 What is Bartels User Language?

1.1.1 Purpose

Bartels User Language introduces almost unlimited features for accessing database contents and activating system functions of the **Bartels AutoEngineer (BAE)**. With **Bartels User Language** the BAE user e.g., is able to

- produce manufacturer-specific **CAM Processor** outputs
- implement and integrate user-specific menu functions ("macros")
- provide special report functions
- introduce special design rule checkers
- develop automatic library management routines
- implement special automatic part placement and routing features
- provide CAM batch programs
- apply the **Neural Rule System** throughout the **AutoEngineer**
- integrate customer-specific relational databases
- provide tools for third party design data input/output

Bartels AutoEngineer provides powerful features for transparently integrating **User Language** programs to the BAE menu system. Key bindings can be used to define hotkeys for calling frequently required **User Language** programs.

There might be BAE users out there not being very skilled at software development. Our customers are not necessarily expected to spend their time practicing extensive **User Language** programming (though they of course can do that). The **User Language** concept rather enables Bartels System to implement almost arbitrary advanced and/or additional BAE software features without the need to change the BAE software kernel which would require a time-consuming BAE software release process. Due to this concept, Bartels System is able to offer superior quality in software support, i.e., highest flexibility and shortest response time at the implementation of customer-specific BAE features. As a result, a large number of **User Language** programs developed due to user-specific demands are delivered with the BAE software. Since these programs are provided with source code, BAE users easily can adjust them to even more specific requirements. See chapter 4 of this manual for a list of the **User Language** programs provided with the BAE software (including short program descriptions), and for information on how to install these programs for proper use throughout the **Bartels AutoEngineer**.

1.1.2 Components

Bartels User Language consists of its language definition, the **Bartels User Language Compiler** and the **Bartels User Language Interpreter**, respectively.

Definition of the User Language Programming Language

Bartels User Language is a C-based programming language including powerful internal object-oriented programming (OOP) features such as automatic memory management for list processing, string (class) data type, etc. **Bartels User Language** provides special variable types for accessing the design database (DDB) of the **Bartels AutoEngineer**. A system function library containing standard functions (as known from C) and BAE system functions is included with the **Bartels User Language**. See chapter 2 for a detailed description of the **User Language** definition. See appendix B for a detailed description of the variable types defined for accessing DDB. See appendix C for a complete description of the functions included with the **User Language** system function library.

Bartels User Language Compiler

The **Bartels User Language Compiler (ULC)** is used for translating the **Bartels User Language** source code files into machine code to be executed by the **Bartels User Language Interpreter**. The translation process includes checks on data type compatibility as well as on program executability. The compiler is able to run optimizer passes optionally. Special compiler options can be used to generate linkable **Bartels User Language** libraries. The built-in Linker of the **User Language Compiler** features both static library linking (at compile time) and preparation of dynamic library linkage (at runtime). See chapter 3.2 for a complete description of the **User Language Compiler**.

Bartels User Language Interpreter

The **Bartels User Language Interpreter** is used to execute compiled **User Language** programs, i.e., to (dynamically link and) run **User Language** machine programs generated by the **User Language Compiler**. The **Bartels User Language Interpreter** is integrated to the **Schematic Editor**, the **Layout Editor**, the **Autorouter**, the **CAM Processor**, the **CAM View** module and the **Chip Editor** of the **Bartels AutoEngineer**. I.e., compiled **User Language** programs can be called from any of these **Bartels AutoEngineer** modules. The program call facilities include explicit program calls from a special BAE menu function as well as implicit program calls by function key-press or program module startup. See chapter 3.3 for a complete description of the **Bartels User Language Interpreter**.

1.2 Characteristics of the Bartels User Language

1.2.1 Bartels User Language Compared to C

The **Bartels User Language** source file format is based on the C programming language. As with C and/or C++, comments are enclosed with `/*` at the beginning and `*/` at the end or can start with `//` and are delimited by the end of the line.

Bartels User Language supports the basic data types `char`, `int` and `double`. The C basic data type `float` as well as pointers and the possibility of qualifying basic data types (`short`, `unsigned`, `long`) are not supported by **Bartels User Language**. The **User Language** basic data type `string` can be used for representing `char` arrays. Another extension to C is the **User Language** basic data type `index`, which provides access to the **Bartels AutoEngineer** Design Database (DDB) via predefined index types. Thereby `index` can be understood as index to a vector of DDB structures. A special operator is available for accessing the index-addressed structure elements. The `index` types and the corresponding structure elements are predefined (see appendix B).

User Language provides dynamic array memory management. This means, that there is no need to define array length limits. **Bartels User Language** even allows for direct assignments of type compatible complex data types (structures, arrays) with equal dimensions.

Bartels User Language does not support explicit declaration of storage classes `auto`, `extern` and `register`. Variables defined in functions (local variables) are supposed to be of storage class `auto` and those defined outside any function (global variables) are supposed to be global unless they are explicitly declared `static`. Functions defined in the program text are assumed to be global, unless they are explicitly defined `static`. The scope of global variables and functions covers the whole program, whilst `static` variables and functions are valid only in the currently compiled program text (but not in any other program or library module yet to be linked with the program).

Bartels User Language provides a macro preprocessor to support language expansion mechanisms. As with C, the preprocessor statements `#include`, `#define`, `#undef`, `#if`, `#ifdef`, `#ifndef`, `#else` and `#endif` are available.

1.2.2 Data Types, Constants, Variables

Bartels User Language provides the *basic data types* `char` (character), `int` (numeric integer value), `double` (double precision numeric real value), `string` (character array) and `index` (predefined index to DDB structure; see appendix B). Moreover complex composed data type definitions (vectors and/or array as well as structures) can be derived from basic data types.

Basic data types can be represented by *constants*. `char` constants must be delimited by single quotes. `string` constants must be delimited by double quotes. Special characters in `char` or `string` constants must be prefixed by the backslash escape character (`\`). Integer constants (`int`) can be specified in decimal (base 10), octal (base 8) or hexadecimal (base 16) representation. Numeric real constants (`double`) can be specified either in fixed floating point representation or in scientific floating point representation (with exponent).

Constant expressions are composed of constants and operators. Constant expressions are evaluated by the **Bartels User Language Compiler** during the translation process (CEE - Constant Expression Evaluation).

All *variables* must be declared before use. A variable declaration determines the name and the data type of the variable. Variable names (identifiers) must start either with a letter or an underscore (_) and can then have letters, digits or underscores in accordance with the C standard. The **Bartels User Language Compiler** distinguishes between lower and upper case letters. Variable declarations can contain variable value initializations. The **User Language Compiler** issues warning messages when accessing variables which have not been initialized, and the **User Language Interpreter** assigns type compatible null values to such variables.

1.2.3 Operators, Assignments

Bartels User Language supports all the C-known operators (`?:`, `+`, `-`, `*`, `/`, `%`, `>`, `>=`, `<`, `<=`, `==`, `!=`, `&&`, `||`, `!`, `++`, `--`, `&`, `|`, `^`, `<<`, `>>`, `~`). The operator evaluation sequence and priority correspond to the C programming language. Operands of different data types are automatically casted to a common and/or operator-compatible data type if possible (the **User Language Compiler** issues an error message if this is not possible). The add operator (`+`) and the comparison operators (`>`, `>=`, `<`, `<=`, `==`, `!=`) can operate directly on the `string` data type.

Assignments usually are performed with the general assignment operator (`=`). The expression value on the right side of the equal sign is assigned to the variable on the left side. **Bartels User Language** also provides the composed assignment operators as known from C (`+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `|=`, `^=`, `<<=`, `>>=`).

1.2.4 Control Structures

The sequence of the instructions to be executed by a program is determined by control structures. **Bartels User Language** provides all C-known control structures except for the `goto` statement and the definition of labels. I.e., the available **Bartels User Language** control structures are `if`, `if-else`, `switch`, `while`, `for`, `do-while`, `break` and `continue`. A special **User Language** control structure is introduced by the `forall` statement which provides a scan of the specified `index` data type and can be conditional.

1.2.5 Program Flow, Functions

Usually, a function is defined for solving a certain sub-problem derived from different tasks. Using functions can simplify the process of software maintenance considerably. **Bartels User Language** provides a function library containing a predefined set of system functions (see appendix C). Beyond that the programmer can write his own functions (user functions). The definition and declaration of the user functions and their parameters correspond to the C programming language. The first user function to be called by the **Bartels User Language Interpreter** when running a program is the one named `main`. A **User Language** program not containing a `main` function usually won't do anything at all. **Bartels User Language** does not distinguish between "call-by-value" and "call-by-reference" when passing function parameters. All function parameters are evaluated using the "call-by-reference" method. Therefore no pointers are required for passing changed parameter values back to the caller of a function.

1.2.6 Special In-Build Features

Bartels User Language provides some powerful in-build features worthwhile to be mentioned here especially.

A BNF precompiler is integrated to the **Bartels User Language**. This BNF precompiler with its corresponding scanner and parser functions can be utilized to implement programs for processing almost any foreign ASCII file data format. See section 2.6.4 for a detailed description of the BNF precompiler facilities.

Bartels User Language provides SQL (Structured Query Language) functions for maintaining relational databases, thus introducing powerful software tools for programming database management systems. These tools e.g., can be utilized for integrating a component database to the **Bartels AutoEngineer** for performing stock and cost expenditure analysis on different variants of a layout including facilities for choosing components with controlled case selection and part value assignment. This is just one example from the wide range of possible database applications. Utilizing database systems could also be worthwhile in the fields of project and version management, address list maintenance, production planning and inventory control, supplier and customer registers management, etc. See appendix C of this documentation for the descriptions of the SQL system functions.

Chapter 2
Language Description

This chapter describes in detail the definition of the **Bartels User Language** and explains how to write **User Language** application programs. The **Bartels User Language** elements are explained in detail, and their usage is illustrated by examples wherever necessary. Additionally, hints are given on how to use the **User Language** programming environment and how to interface to the **Bartels AutoEngineer**.

Contents

2.1 Introducing User Language Programming

This section provides small programming examples in order to introduce the most important elements of the **Bartels User Language**. The purpose thereby is to demonstrate - without entering into formal details or describing exceptions - the basic methods of developing **User Language** programs.

2.1.1 The first User Language Program

The only way to learn how to use the **User Language** is by doing it, i.e., write **User Language** programs, and compile and execute them. The first **User Language** program to be implemented should print a message, and then wait for an interactive keyboard input to abort the program (this is a frequently required programming feature). As already mentioned in the introduction, an **User Language** program must at least contain a **main** function. What we need in this **main** function are instructions for printing the desired message and for activating the requested user query. Both of these instructions can be realized by calling corresponding **User Language** system functions (**printf** and **askstr**). These system functions are known to the **User Language Compiler**, and they are bound to the **Bartels User Language Interpreter**. The programmer just has to know, how these functions are to be called, and what they do (this information can be taken from appendix C of this manual). You should now use your editor for preparing a file named **ulprog.ulc** with the following **User Language** source code (the **.ulc** file name extension is used by the **User Language Compiler** for **User Language** source code file recognition):

```
main()
{
      printf("User Language Program");
      askstr("Press ENTER to continue ",1);
}
```

The above listed **User Language** program does just contain the definition of the function **main**. Parentheses are required after the function name. Usually, formal function parameters are listed inside these parentheses. To distinguish function names from variable names, the parentheses are required even if (as in the example above) no function parameter exists at all. Within the braces the function block is defined. The function block is composed of the statements to be executed by the function. Each statement must be delimited by a semicolon (**;**). The first statement of the **main** function is the call to the **printf** function (to be recognized by the opening parenthesis after the function name). The parameter which is passed to the **printf** function is a constant string (enclosed by double quotes). This string is the message, which the program will print to the screen when executed after faultless compilation. The second statement is a call to the function **askstr**. This function issues the prompt string, which is specified with the first function parameter, and waits for an interactive string input. The user interaction takes place in the status line of the **Bartels AutoEngineer**, and the second **askstr** parameter indicates the maximum permissible input string length. The **askstr** call is the last statement to be processed by the program, i.e., the program finishes after executing the **askstr** call. Once the program source code is edited and stored to **ulprog.ulc**, it can be translated with the following **User Language Compiler** call:

```
ulc ulprog
```

The **User Language Compiler** issues the following message, if no compilation error occurred:

```
============================
BARTELS USER LANGUAGE COMPILER
============================

Compiling source code file "ulprog.ulc"...
Program 'ulprog' successfully created.
Source code file "ulprog.ulc" successfully compiled.

No errors, no warnings.
User Language Compilation successfully done.
```

Once the **User Language Compiler** program source code has been translated, and the corresponding machine program named **`ulprog`** has been stored to the **`ulcprog.vdb`** file of the **Bartels AutoEngineer** programs directory, the program can be executed by the **Bartels User Language Interpreter**. This can be applied e.g., by starting the **Bartels AutoEngineer Schematic Editor** and by activating the Run User Script function from the File menu. The program name (**`ulprog`**) must be specified to the corresponding query:

After starting the program, the BAE graphic workspace is switched to text output mode, and the **`User Language Program`** message is printed to the screen. Subsequently, the **`Press ENTER to continue`** prompt is displayed in the BAE input window. Return key input terminates the **User Language** program and restores the graphic workspace.

2.1.2 Variables, Arithmetic and Functions

The next example illustrates a series of further specific **User Language** characteristics. The following **User Language** program examines some circles (specified by center point and radius) to check whether they overlap (drill data test?!), and issues corresponding messages:

```
// Circle Test Program

double tol=0.254*5;           // Tolerance

struct pos {                  // Position descriptor
      double x;               // X coordinate
      double y;               // Y coordinate
      };

struct circle {               // Circle descriptor
      double rad;             // Circle radius
      struct pos c;           // Circle position
      };

// Main program
main()
{
      // Define three circles
      struct circle c1 = {  4.5, { 19.4, 28.3} };
      struct circle c2 = { 17.0, { 37.6, 9.71} };
      struct circle c3 = { 1.5E01, { 25, 0.2e2} };
      // Perform circle test
      printf("Circle 1 - 2 overlap : %d\n",circletest(c1,c2));
      printf("Circle 1 - 3 overlap : %d\n",circletest(c1,c3));
      printf("Circle 2 - 3 overlap : %d\n",circletest(c2,c3));
      // Prompt for continue
      askstr("Press ENTER to continue ",1);
}

int circletest(c1,c2)
// Circle test function
// Returns: nonzero if overlapping or zero else
struct circle c1,c2           /* Test circles 1 and 2 */;
{
      double d                /* Distance value */;
      // Get circle center point distances
      d=distance(c1.c,c2.c);
      // Error tolerant check distance against radius sum
      return(d<=(c1.rad+c2.rad+tol));
}

double distance(p1,p2)
// Get distance between two points
// Returns: distance length value
struct pos p1                 /* Point 1 */;
struct pos p2                 /* Point 2 */;
{
      double xd=p2.x-p1.x     /* X distance */;
      double yd=p2.y-p1.y     /* Y distance */;
      // Calculate and return distance
      return(sqrt(xd*xd+yd*yd));
}
```

The above listed program source code contains a series of comments enclosed by **/*** and ***/**; such comments can extend over several lines, but they must not nest. Another type of comment starts with **//** and extents to the end of line. Since comments can keep the program source code well understandable, it is recommended to use such inline documentation to simplify **User Language** software maintenance.

The program above also contains a series of variable definitions. All variables must be declared before use. A variable declaration determines the name and the data type of the variable. **Bartels User Language** distinguishes between global variables, local variables and function parameters. Global variables are valid throughout the entire program text. Local variables are valid in the function where they are defined. Function parameters are used for passing values to functions. In the example above, `tol` is the only global variable (with data type `double`). Local variables are, e.g., `xd` and `yd` (data type `double`) in the `distance` function. Function parameters are, e.g., `c1` and `c2` in the `circletest` function; these two parameters are of the specially defined combined `struct circle` data type. Variable declarations can contain variable value initializations (see the global variable `tol` or the local variables `xd` and `yd` in the `distance` function). Combined data type variables can be initialized (see the local `struct` variables `c1`, `c2` and `c3` in the `main` function). A list of variable names can be specified at the declaration of variables (see the declaration of the parameters `c1` and `c2` in the `circletest` function).

Values are calculated within expressions. The equals sign (`=`) can be used for assigning the resulting expression value to a variable.

A data type must be specified at the definition of functions. In the example above, the `distance` function is of type `double`, and the `circletest` function is of type `int`. The function data type is set to `int` if the function data type specification is omitted (as with the `main` function in the example above). A special function data type is `void`. Each function - except for the `void` functions - returns a value compatible to the corresponding function data type. The function return value is passed back to the caller of the function with the `return` statement, which is coincidentally the last instruction to be executed by the function.

2.1.3 Arrays and Control Structures

The following example shows how to use arrays and control structures. A list of integer values are transformed into strings, and a report of the transformations is printed:

```
// Integer list
int intary[]={ 0,17,-12013,629,0770,0xFF,-16*4+12 };

// Main program
main()
{
      int i                   /* Loop control variable */;
      // Set last integer value
      intary[10]=(-1);
      // Loop through integer list
      for (i=0;i<=10;i++)
              // Print integer and integer string
              printf("%8d : \"%s\"\n",intary[i],inttostr(intary[i]));
      // Prompt for continue
      askstr("Press ENTER to continue ",1);
}

string inttostr(int intval)
// Convert integer value to a string
// Returns: resulting string
{
      string resstr=""        /* Result string */;
      int n=intval,i=0        /* Integer value, loop counter */;
      char sign               /* Sign character */;
      // Test for negative integer value
      if (n==0)
              // Return zero integer string
              return("0");
      else if (n>0)
              // Set sign to plus character
              sign='+';
      else {
              // Make integer value positive
              n=-n;
              // Set sign to minus character
              sign='-';
              }
      // Build result string
      do {    // Get and append next character
              resstr[i++]=n%10+'0';
              } while ((n/=10)!=0);
      // Append zeros
      while (i++<15)
              resstr+='0';
      // Append sign character
      resstr+=sign;
      // Reverse string
      strreverse(resstr);
      // Return string result
      return(resstr);
}
```

In the example above, an integer array (global **int** variable **intary**) is declared and (partially) initialized. The bracket pair after the variable name **intary** defines an one-dimensional **int** vector. Multiple vector dimensions can be specified by appending further bracket pairs to the declaration (**intary[][]...[]**). Since the **User Language** provides powerful in-build features for dynamically managing arrays, it is not necessary, to define array length limits; i.e., both the **User Language Compiler** and the **Bartels User Language Interpreter** require just the information about the dimension of an array and/or vector. Nevertheless some checks are applied in order to prevent from accessing non-existent array elements (which would cause memory protection faults); the Compiler is able to check for constant negative (i.e., invalid) array indices, and the Interpreter is able to check whether an array index refers to an array element outside the currently engaged array field range. The array index value 0 always refers to the first array element.

The **string** data type corresponds to an one-dimensional array of type **char**. **User Language** provides in-build features for the direct assignment of arrays and/or vectors with corresponding data type and equal dimension. These features have been utilized at the initialization of **resstr** (local **string** variable of the **inttostr** function) as well as with the assignment of the **return** value of the **inttostr** function. The add operator can also be applied to **string** values, with the result of the add operation corresponding to a string catenation.

The example above contains some control structures. A **for** loop for processing the elements of the **intary** array variable is applied in the **main** function. The **inttostr** function uses a **while** loop and a **do-while** loop for manipulating the **resstr string** variable. The **inttostr** function utilizes an **if** control structure to process dependent program blocks according to the current value of local variable **n**.

2.2 Lexical Conventions

Bartels User Language defines spacing, identifier, constant, reserved word and operator token classes.

2.2.1 Spacing

The spacing token class includes blanks, tabulators, newlines and comments. Comments start with the token **/*** and end with ***/**; they do not nest. Another type of comment starts with the token **//** and extends to the end of line. Spacings are ignored by the Compiler except as they serve to separate adjacent identifiers, reserved words and constants.

2.2.2 Identifiers

An identifier is the name of a variable, a function or a symbolic constant. Each Identifier consists of a sequence of letters and digits. The first identifier character must be a letter. The underscore (_) is treated as a letter. The **User Language Compiler** distinguishes between upper case letters and lower case letters (case-sensitivity). Identifiers must differ from reserved words (see below).

Examples:

```
X_coord   value   P4   File_Name   _euklid
```

2.2.3 Constants and Constant Expressions

This section describes the **Bartels User Language** constant types.

Integer Constants

Numeric integer constants are associated with the data type **int**. They consist of a sequence of digits, and they are usually interpreted as decimal numbers. A constant integer specification is interpreted as octal number (in base 8), if it starts with **0** (digit zero), in which case the digits 8 and 9 are not allowed. A constant integer specification is interpreted as hexadecimal number (in base 16), if it starts with **0x** or **0X** (digit zero followed by letter **x**) in which case the letters **a** to **A** through **f** to **F** correspond to the hexadecimal digit values 10 through 15. Negative integer constants are not provided (the minus sign rather works as operator in order to form negative constant integer expressions; see below).

Examples:

```
1432   073   0xF4A5   9
```

Floating Point Constants

Floating point constants are associated with the data type **double**. They consist of an integer part, a decimal point (**.**), a fraction part, an **e** or **E** character (letter **e**), and an optionally signed integer exponent. Either the integer part or fraction part (not both) can be missing; or otherwise either the decimal point or the e letter and the exponent (not both) can be missing.

Examples:

```
2.54   .78   4.   4.1508E-3   0.81037e6   17228E5
```

Character Constants

Character constants are associated with the data type **char**. They consist of a single character enclosed by single quotes (apostrophes). The value of a character constant accords to the corresponding numeric value of the character in the machine's character set.

The escape character \ (backslash) can be used for specifying special characters. Table 2-1 contains a list of characters represented by escape sequences.

Table 2-1: Character Escape Sequences

Backspace	BS	**\b**
Horizontal Tabulator	HT	**\t**
Line Feed	LF	**\n**
Form Feed	FF	**\f**
Carriage Return	CR	**\r**
Escape Character	\	****
Apostrophe	'	**\'**
Null Character	NUL	**\0**

Arbitrary bit patterns consisting of an escape symbol followed by up to three octal digits can be specified to represent the value of the desired character; the null character (NUL, **\0**) is a special case of this construction.

String Constants

String constants are associated with the data type **string**. They consist of a sequence of characters enclosed by double quotes (quotation marks). The **Bartels User Language Compiler** automatically appends a null character (NUL, **\0**) to the end of string constants; this convention is utilized by the **Bartels User Language Interpreter** to match the end of constant strings. Quotation marks included with a constant string must be preceded by the escape character (****); in addition the same escape character sequences as for character constants (see above) are permitted.

Examples:

```
"IC1"   "4.8 kOhm"   "This is a string with Newline\n"
```

Constant Expressions

A constant expression is an expression, which is composed of constant values and operators. Constant expressions are evaluated at compile time already (CEE, Constant Expression Evaluation), i.e., they do not have to be calculated by the Interpreter at runtime. This means, that wherever constants are required, corresponding constant expressions can be used without disadvantages regarding to the program memory or runtime requirements.

Examples:

```
int i=19-(010+0x10);                CEE: int i=-5;
double d=-(4.7+2*16.3);             CEE: double d=-37.3;
string s="Part"+' '+"IC1";          CEE: string s="Part IC1";
```

2.2.4 Terminal Symbols

Reserved Words

Table 2-2 contains the list of **Bartels User Language** identifiers reserved for use as keywords. These identifiers can only be used in their predefined meaning.

Table 2-2: Reserved Words

`#bnf`	`#define`	`#else`	`#endif`	`#if`	`#ifdef`	`#ifndef`	`#include`
`#undef`	`break`	`case`	`char`	`continue`	`default`	`do`	`double`
`else`	`for`	`forall`	`if`	`index`	`int`	`of`	`return`
`static`	`string`	`struct`	`switch`	`typedef`	`void`	`where`	`while`

Operators

Table 2-3 lists the **Bartels User Language** operators. These operators activate special operations regarding to the current program context.

Table 2-3: Operators

`!`	`!=`	`%`	`%=`	`&`	`&&`	`&=`	`(`	`)`	`*`	`*=`
`+`	`++`	`+=`	`,`	`-`	`--`	`-=`	`.`	`/`	`/=`	`:`
`;`	`<`	`<<`	`<<=`	`<=`	`=`	`==`	`>`	`>=`	`>>`	`>>=`
`?`	`[`	`]`	`^`	`^=`	`{`	`|`	`|=`	`||`	`}`	`~`

2.3 Data Types and Definitions

2.3.1 Data Types

Bartels User Language provides the following basic data types:

`char`	Character taken from the machine's character set
`int`	Numeric integer value
`double`	Numeric double precision floating point value
`string`	Character array
`index`	Index to predefined BAE DDB structure

Bartels User Language provides the following combined data types:

array	Collection of elements with same data type
`struct`	Collection of elements with different data types

Data Type Conversion

Some operators can cause implicit data type conversions. A series of arithmetic operations require operand(s) with special data types. Likewise a corresponding data type compatibility is required with the assignment of values to variables and/or the passing of function parameters. The **User Language Compiler** checks the compatibility of the operands. Operands of different data types are automatically casted to a common or valid data type if possible. These type casts are applied according to the following rules: valid type conversions without loss of information are `char` to `int`, `char` to `string` and `int` to `double`; permissible type casts with a loss of information are `int` to `char` and `double` to `int`. The **User Language Compiler** issues error messages if the required type compatibility can not be achieved by applying the type cast rules.

2.3.2 Variables

All global and local variables must be declared before use. A variable declaration defines the name and the data type of the variable. Such declarations determine, how user-introduced names are to be interpreted by the **User Language**. Each declaration consists of a data type specification and a list of declarators. Each declarator is composed of the corresponding variable name and an optional initialization.

Basic Data Types

The declaration of **char** variables is applied as in

```
char c;
char TAB = '\t', NEWLINE = '\n';
```

where the **char** variables **c** (not initialized), **TAB** (initialized with the tabulator control character) and **NEWLINE** (initialized with the newline control character) are declared.

The declaration of **int** variables is applied as in

```
int i, MAXLINELEN = 80;
int pincount = 0;
```

where the **int** variables **i** (not initialized), **MAXLINELEN** (initialized with the value 80) and **pincount** (initialized with 0) are declared.

The declaration of **double** variables is applied as in

```
double x_coord, y_coord;
double MMTOINCH = 1.0/25.4;
double starttime = clock();
```

where the **double** variables **x_coord** and **y_coord** (not initialized), **MMTOINCH** (initialized with a numeric expression) and **starttime** are declared; the variable **starttime** is initialized with the return value of the system function **clock** (i.e., the elapsed CPU time).

The declaration of **string** variables is applied as in

```
string s1;
string ProgName = "TESTPROGRAM", ProgVer = "V1.0";
string ProgHeader = ProgName+"\t"+ProgVer;
```

where the **string** variables **s1** (not initialized), **ProgName** (initialized with **TESTPROGRAM**), **ProgVer** (initialized with **V1.0**) and **ProgHeader** are declared; **ProgHeader** is initialized with an expression which emerges from a catenation of the **string** variable **ProgName**, the tabulator control character and the **string** variable **ProgVer**.

The declaration of **index** type variables is applied as in

```
index L_MACRO macro;
index L_CNET net1, net2;
```

where the **index** variables **macro** (**index** variable type **L_MACRO**), **net1** and **net2** (**index** variable type **L_CNET**) are declared. The declaration of **index** variable types consists of the keyword **index** followed by the name of the **index** variable type (e.g., **L_MACRO** and/or **L_CNET**) and the desired variable name(s). The identifiers of the **index** variable types are predefined (see also appendix B of this manual). Only **index** variable types compatible to each other can be used in the same program. The reason for this restriction is, that with **index** data types the access to corresponding entries of the **Bartels AutoEngineer** design data base (DDB) is defined; the availability of these DDB entries differs according to the interpreter environment (i.e., the **Schematic Editor** provides data type definitions which are not available in the layout system). The **User Language Compiler** issues an error message if incompatible **index** variable types are used in the same program. The **User Language Interpreter** behaves similarly; an error message is issued and the program is canceled when trying to run a **User Language** program with references to **index** variable types not compatible with the current interpreter environment. Please refer to appendix A and/or appendix B of this manual for information about **index** data type compatibilities.

Arrays

An array (or vector) is a complex data type composed of elements of the same data type. With the declaration of array variables the specification of the array dimension is required in addition to the data type and variable name definition. The dimension is specificied by appending bracket pairs to the variable name, with each bracket pair corresponding to one dimension. At the initialization of array variables, the corresponding values are to be separated by commas, and each array dimension is to be enclosed with braces.

The declaration of array variables is applied as in

```
int intary[], intfield[][][];
double valtab[][] = {
      { 1.0, 2.54, 3.14 },
      { 1.0/valtab[0][1], clock() }
      };
string TECHNOLOGIES[] = {
      "TTL", "AC", "ACT", "ALS", "AS", "F",
      "H", "HC", "HCT", "HCU", "L", "LS", "S"
      };
```

where the **int** arrays **intary** (1-dimensional) and **intfield** (3-dimensional), the 2-dimensional **double** array **valtab** and the 1-dimensional **string** array **TECHNOLOGIES** are declared. The declarations of **valtab** and **TECHNOLOGIES** contain initializations according to the following assignments:

```
valtab[0][0] = 1.0;
valtab[0][1] = 2.54;
valtab[0][2] = 3.14;
valtab[1][0] = 1.0/valtab[0][1];
valtab[1][1] = clock();
TECHNOLOGIES[0] = "TTL";
TECHNOLOGIES[1] = "AC";
TECHNOLOGIES[2] = "ACT";
:
TECHNOLOGIES[11] = "LS";
TECHNOLOGIES[12] = "S";
```

The basic **User Language** data type **string** is equivalent to a 1-dimensional **char** array, i.e., the declarations

```
string s;
```

and

```
char s[];
```

are synonymous.

Structures

A structure is a complex data type composed of elements with different data types, i.e., the elements of a structure are to be defined with different names. The purpose of structure definitions is to unite different variable types, which share special relations regarding on how they are to be processed. It has to be distinguished between structure definitions and structure declarations. A structure definition is composed of the keyword **struct**, the name of the structure definition and the list of structure element definitions (enclosed with braces). A structure declaration consists of the keyword **struct**, the name of a valid structure definition and the name of the variable to be associated with the structure definition. Structure definitions and structure declarations can be combined. The name of the structure definition can be omitted. Initializations in structure declarations are allowed in which case the syntax corresponds to the array declaration conventions (see above).

The declaration of structures is applied as in

```
// Structure declarations

struct coordpair {
      double x
      double y;
      };

struct coordpair elementsize = {
      bae_planwsux()-bae_planwslx(),
      bae_planwsuy()-bae_planwsly()
      };

struct elementdes {
      string fname, ename;
      int class;
      struct coordpair origin, size;
      } element = {
              bae_planfname(),
              bae_planename(),
              bae_planddbclass(),
              {
                      bae_planwsnx(),
                      bae_planwsny()
                      },
              elementsize
              };

struct {
      string id, version;
      struct {
              int day;
              string month;
              int year;
              } reldate;
      } program = {
              "UL PROGRAM",
              "Version 1.1",
              { 4, "July", 1992 }
              };
```

where the definition of the structure **coordpair**, the declaration of the variable **elementsize** (structure of type **coordpair**), the definition of the structure **elementdes**, the declaration of the variable **element** (structure of type **elementdes**) and the declaration of the **struct** variable **program** is accomplished. The declarations of **elementsize**, **element** and **program** contain initializations according to the following assignments:

```
elementsize.x=bae_planwsux()-bae_planwslx();
elementsize.y=bae_planwsuy()-bae_planwsly();
element.fname=bae_planfname();
element.ename=bae_planename();
element.class=bae_planddbclass();
element.origin.x=bae_planwsnx();
element.origin.y=bae_planwsny();
element.size=plansize;
program.id="UL PROG";
```

```
program.version="Version 1.1";
program.reldate.day=4;
program.reldate.month="July";
program.reldate.year=1992;
```

The following example illustrates how structure and array definitions and/or declarations can be combined:

```
struct drilldef {
      index L_DRILL drilltool;
      struct { double x, y; } drillcoords[];
      } drilltable[];
```

Data Type Renaming

Bartels User Language provides a mechanism for renaming data types. This feature allocates an additional name for an already known data type (but it does not create a new data type). Data type renaming is accomplished by the specification of the keyword **typedef** followed by a valid data type specification and the new name to be introduced for this data type. A data type specification introduced with **typedef** can subsequently be used as data type specifier when declaring variables, functions or function parameters.

Data type renaming is utilized as in

```
typedef index L_CNET NETLIST[];
typedef int IARY[];
typedef IARY MAT_2[];
typedef struct {
      int pointcount;
      struct {
              int t;
              double x,y;
              } pointlist[];
      } POLYLIST[];
MAT_2 routmatrix;
NETLIST netlist;
POLYLIST polygonlist;
```

where the variables **routmatrix** (2-dimensional **int** array), **netlist** (1-dimensional **index** array of type **L_CNET**) and **polygonlist** (1-dimensional array of structures containing an **int** element and a **struct** array) are declared.

2.3.3 Functions

A function usually is defined for solving a certain sub-problem derived from larger problems. The use of functions can simplify the process of software maintenance considerably since complex operation sequences can be applied repeatedely without the having to code the corresponding instructions time and again. **Bartels User Language** distinguishes between the predefined system functions and the user-defined functions.

Function Definition

The **Bartels User Language** system functions are known to the **User Language Compiler**, and they are bound to the **User Language Interpreter**. See appendix C of this manual for the description of the **Bartels User Language** system functions. The programmer can make use of the system functions or write his own functions.

A function definition consists of the function header and the function block. The function header is composed of the function type specification, the function name, and the definition and declaration of the function parameters. The function type determines the data type of the value to be returned by the function. The **void** function type applies to functions without return value. The function data type defaults to **int** if the function type specification is omitted. The function type is followed by the function name, which must be unique throughout the program text. The function parameter definition consists of the list of the function parameter names and/or declarations (separated by commas); the function parameter list must be enclosed by parentheses. All function parameters - except for the **int** parameters and those already explicitly declared with the parameter list - must be declared at the end of the function header, with the declaration of the parameters corresponding to the declaration of normal variables (see above). The function header is followed by the function block. The function block must be enclosed by braces, and consists of the statements to be executed by the function.

Function definition examples:

```
double netroutwidth(index L_CNET net)
// Get the routing width of a given net
// Returns : width or 0.0 if two pins with different width
{
      index L_CPIN pin;       // Pin index
      int pincnt=0;           // Pin count
      double rw=0.0;          // Rout width
      // Loop through all pins
      forall (pin of net) {
              // Test if the pin introduces a new rout width
              if (pin.RWIDTH!=rw && pincnt++>0)
                      return(0.0);
              // Set the rout width
              rw=pin.RWIDTH;
              }
      // Return the rout width
      return(rw);
}

int allpartsplaced()
// Test if all net list parts are placed
// Returns : 1 if all parts are placed or zero otherwise
{
      index L_CPART cpart;    // Connection part index
      // Loop through the connection part list
      forall (cpart where !cpart.USED)
              // Unplaced part matched
              return(0);
      // All parts are placed
      return(1);
}
```

```
double getdistance(xs,ys,xe,ye)
// Get the distance between two points
// Returns : the distance length value
double xs, ys;                // Start point coordinate
double xe, ye;                // End point coordinate
{
      double xd=xe-xs;        // X distance
      double yd=ye-ys;        // Y distance
      // Calculate and return the distance (Pythagoras)
      return(sqrt(xd*xd+yd*yd));
}

double arclength(r,a1,a2)
// Get arc segment length by radius and start-/end-point angle
// Returns : the arc segment length value
double r;                     // Radius
double a1;                    // Start point angle (in radians)
double a2;                    // End point angle (in radians)
{
      // Arc; "absolute" angle between start and end point
      double arc = a1<a2 ? a2-a1 : 2*PI()+a2-a1;
      // Get and return the arc segment length
      return(arc*r);
}

double getangle(cx,cy,x,y)
// Get the angle of a circle arc point
// Returns : the angle (in radians; range [0,2*PI])
double cx, cy;                // Circle center coordinate
double x, y;                  // Circle arc point coordinate
{
      double res;             // Result value
      // Get arc tangent of angle defined by circle point
      res=atan2(y-cy,x-cx);
      // Test the result
      if (res<0.0)
              // Get the "absolute" angle value
              res=PI()-res;
      // Return the result value
      return(res);
}

double PI()
// Returns the value of PI in radians
{
      // Convert 180 degree and return the result
      return(cvtangle(180.0,1,2));
}

void cputimeuse(rn,st)
// Report CPU time usage (in seconds)
string rn;                    // Routine name
double st;                    // Start time
{
      // Print CPU time elapsed since start time
      printf("(%s) Elapsed CPU Time = %6.1f [Sec]\n",rn,clock()-st);
}
```

Function Call and Parameter Value Passing

Each function known in a **User Language** program and/or program module can be called in this program and/or program module. However, only system functions compatible to each other can be used in the same program. The reason for this restriction is, that a system function is implemented and/or available in a certain set of interpreter environments only (e.g., the system function for setting **CAM Processor** plot parameters obviously can not be called from the **Schematic Editor**). The **User Language Compiler** issues an error message if incompatible system functions are used in a program. The **User Language Interpreter** behaves similarly; an error message is issued and the program is canceled when trying to run a **User Language** program with references to system functions not compatible to the current interpreter environment. Please refer to appendix A and/or appendix C of this manual for information about system function compatibilities.

A function call consists of the function name and - enclosed with parentheses - the list of the parameters (arguments) to be passed to the function.

The contents of global program variables are available in each function of the same scope. I.e., global variables can be used at any time for passing values to functions. Besides that values can be passed with the function parameters. Since the usage of parameters provides easy maintenance, this method should be preferred. The list of parameters which is passed to the function must correspond with the formal parameter list introduced with the function definition (i.e., the parameter count as well as the data types must match). At the function call, the values of the current parameters are copied to the corresponding formal parameters. After successful execution of the function, each parameter value changed by the function is stored back to the current parameter (this applies only if the parameter refers to a variable). Finally, there is the possibility of passing values with the function return value, where the **`return`** statement is used for setting a function result value which is passed back to the caller of the function and can be evaluated in the expression containing the function call.

Examples for function calls and value passing:

```
// Date structure
struct date { int day, month, year; };
// Global program variables
string globalstr="Global string";
int fctcallcount=0;

// Main program
main()
{
      // Local variables of main
      string resultstr="function not yet called";
      struct date today = { 0, 0, 0 };
      double p=0.0, b=2.0, e=10.0;
      // Print the global variables
      printf("fctcallcount=%d, %s\n",fctcallcount,globalstr);
      // Print the local variables
      printf("resultstr=\"%s\"\n",resultstr);
      printf("today : %d,%d,%d",today.day,today.month,today.year);
      printf("\t\tb=%.1f, e=%.1f, p=%.1f\n",b,e,p);
      // Call function
      resultstr=function(today,b,e,p);
      // Print the global variables
      printf("fctcallcount=%d, %s\n",fctcallcount,globalstr);
      // Print the local variables
      printf("resultstr=\"%s\"\n",resultstr);
      printf("today : %d,%d,%d",today.day,today.month,today.year);
      printf("\t\tb=%.1f, e=%.1f, p=%.1f\n",b,e,p);
}
```

```
string function(curdate,base,exponent,power)
struct date curdate;          // Current date parameter
double base;                  // Base parameter
double exponent;              // Exponent parameter
double power;                 // Power parameter
{
      // Increment the function call count
      fctcallcount++;
      // Set the global string
      globalstr="Global string changed by function";
      // Get the current date
      get_date(curdate.day,curdate.month,curdate.year);
      // Calculate the power
      power=pow(base,exponent);
      // Return with a result string
      return("function result string");
}
```

The example above produces the following output:

```
fctcallcount=0, Global string
resultstr="function not yet called"
today : 0,0,0         b=2.0, e=10.0, p=0.0
fctcallcount=1, Global string changed by function
resultstr="function result string"
today : 4,6,92               b=2.0, e=10.0, p=1024.0
```

Control Flow and Program Structure

After calling a function, this function keeps control of the program flow until it meets with another function call or a **`return`** statement or else has reached its end after processing its last statement. At a function call the control is passed to the called function. When reaching a **`return`** statement or the end of the function, the control is passed back to the caller of the function. If a function with the name **`main`** is defined in a program, the **User Language Compiler** produces machine code for calling this function immediately after initializing the global program variables. I.e., the program control flow usually starts with the **`main`** function. Since each function passes the control back to the caller (unless the program contains endless recursions), the control will finally fall back to the **`main`** function. If the end of the **`main`** function is encountered or if a **`return`** statement is reached in the **`main`** function, then the end of the program is reached and the control is passed back to the **User Language Interpreter**, which then terminates the program flow.

Recursive Functions

Functions can be used recursively, i.e., a function can call itself directly or indirectly. This however is meaningful only if with each recursive function call a condition changes in order to reach a clearly defined final state, which causes a recursion interrupt (otherwise the function is endless-recursive and the program runs "forever").

Recursive programming of functions can save program code and increase the legibility of the source code. However, program runtime and memory requirements increase with recursive programming and endless recursions might be implemented inadvertently. Hence careful considerations should be taken on the use of recursive functions. The **Bartels User Language Interpreter** eventually encounters an out of memory and/or stack overflow error when processing endless recursive functions since at least a return address must be stored for each function call.

2.3.4 Scope Rules

The **User Language Compiler** checks the validity of each object reference (name and/or identifier) of the program to be compiled. For that purpose a valid program scope is assigned to each identifier used in the program. This (lexical) identifier scope is the region of the program text where the identifier is defined. he corresponding object is known and can be referenced throughout this scope. There is a distinction between global and local scopes. The global scope extends to the entire program (i.e., separately compiled program modules and/or libraries to be linked later), whilst local scopes correspond with the function definitions.

The function names of a program are global, i.e., they are valid throughout the entire program. Variable and type names defined inside a function are local to this function; variable and type names defined outside any function are global. Function parameter names are treated like local variable names, i.e., they are local to the corresponding function. Structure definition names on principle are global throughout the currently compiled program text. Function and global variable scopes can be restricted to the currently compiled program text by assigning the `static` storage class. The `static` storage class is used to avoid name conflicts when binding and/or linking different program modules and/or libraries.

To avoid name conflicts, the elements of each object class must have different names inside their corresponding valid scopes. Local object references have higher priority than global object references.

2.4 Expressions

This section describes the expression operators provided by the **Bartels User Language**. The operators are introduced in a sequence according to decreasing operator precedence. The associativity, i.e., the order of evaluation is mentioned as well; each operator is evaluated either from the left to the right or vice versa (depending on the implicit parentheses). The precedence indicates the priority at which the corresponding expression is evaluated. The sequence of evaluation is undefined, unless the precedence takes effect. It is left up to the implementation, to evaluate partial expressions in the most efficient way, even if these expressions cause side effects. The evaluation sequence of the side effects thereby is undefined. Expressions with associative and/or commutative operators can be rearranged arbitrarily. Certain sequences of operator evaluation can be forced by assignments to (temporary) variables.

2.4.1 Primary Expressions

Primary expressions include constants, object references, parenthesis-enclosed expressions, function calls, array element accesses, structure element accesses or index element accesses. These operations are left-associative, i.e., they are evaluated from left to right.

Constants and Object References

Each suitable arranged identifier referring to an object is a valid primary expression. Integer constants, floating point constants, character constants and string constants are primary expressions as well (see chapter 2.2.3 for the representation of constants).

Parenthesis-Enclosed Expressions

An expression enclosed with parentheses is a primary expression. Parentheses can be set explicitly to redefine the sequence of evaluation. Since the multiplicative operators have higher precedence than the additive operators, the resulting value of the expression in

```
a + b * c
```

emerges from the sum of variable a plus the product of b*c, while the resulting value of the expression in

```
(a + b) * c
```

emerges from the product of the sum a+b multiplied by the variable c.

Function Call

A function call consists of the name of the function to be called, followed by a parenthsis-enclosed list of comma-separated expressions. The values of the expression list represent the current parameters for the function call. A function call can be used as primary in any other expression, if the corresponding function returns a non-void value. Examples for typical function calls are

```
init_states(pass=1);
printf("This is a message!\n");
printf("Element %s\n",bae_planename());
xpos=nref_xcoord(ask_partname())-bae_planwsnx();
```

Array Element Access

A primary expression followed by a bracket-enclosed expression again is a primary expression. This operation applies for the access to an array (or **string**) element. The expression left to the brackets refers to the array; the expression enclosed with the brackets is interpreted as **int** array index value indicating the required array element. The index value must not be negative, and the index value 0 (zero) refers to the first element of the array. When storing to an array, the **User Language Interpreter** automatically adjusts the array length limit and redefines the currently valid array range if necessary. Read access to an array is only permitted in the defined array range. In the following example the **strisoctal** function checks, whether the given **string** value contains octal digits only (**0** to **7**), and returns 1 if so or zero otherwise:

```
int strisoctal(string str)
{
      for (i=0;i<strlen(str);i++)
            if (!isdigit(str[i]) || str[i]=='8' || str[i]=='9')
                  return(0);
      return(1);
}
```

In the example above, the array range is checked with the **strlen** system function. The following example uses a special **int** variable ("filecount") for performing the array range check:

```
string curfilename="", filelist[];
int i, filecount=0;
while (scandirfnames(".",".ddb",curfilename)==1)
      filelist[filecount++]=curfilename;
for (i=0;i<filecount;i++)
      printf("File %s \n",filelist[i]);
```

Within the example above first a list of file names (to be found in the current directory and ending on **.ddb**) is build, and then this file name list is printed.

Structure and Index Element Access

A primary expression followed by a point and an identifier again is a primary expression. The expression left to the point refers to a structure or **index** type, and the identifier following to the point operator designates a defined element of the corresponding structure or **index** type. The write access to structure elements is always possible, but no storage is permitted on **index** type elements (this would cause the **User Language Compiler** to issue a corresponding error message). The read access to the elements of a currently valid **index** variable always is permissible, while only previously initialized **struct** variable elements can be read (otherwise the **User Language Interpreter** would encounter a memory access violation and issue a corresponding error message). The following program part defines a list of structures and produces for each macro of the currently loaded layout a list element containing the macro name and the macro class:

```
int macrocnt=0;
struct { string name; int class; } macrolist[];
index L_MACRO macro;
forall (macro) {
      macrolist[macrocnt].name=macro.NAME;
      macrolist[macrocnt++].class=macro.CLASS;
      }
```

2.4.2 Unary Expressions

Unary expressions include all operators evaluating a single operand. These operations are right-associative, i.e., they are evaluated from right to left.

Increment and Decrement

The unary increment operator **++** changes its operand by adding the value 1 to the value of the operand. The unary decrement operator **--** changes its operand by subtracting the value 1 from the operand. The increment and decrement operators can be used either as prefix operator as in **++n** or as postfix operator as in **n++**. Both the prefix and the postfix notation cause an increment and/or decrement of the operand. The difference however is that the prefix expression changes the operand before using its value, while the postfix expression changes the operand after using its value. I.e., the result of these expression can have different meanings according to the program context. If, e.g., the value of **count** is 12, then

```
n = --count ;
```

sets the value of **n** to 11, but

```
n = count-- ;
```

sets the value of **n** to 12 (in both cases the value of **count** becomes 11).

Arithmetic Negation

The resulting value of the unary operator **-** is the arithmetic negation of its operand, i.e., the operand's value multiplied by (-1).

Logical Negation

The resulting value of the unary operator **!** is the logical negation of its operand. The value is set to either 1 for zero operand values (i.e., 0 or empty string for **string** operands), or 0 for nonzero operand values. The type of the result is **int**.

Bit Complement

The unary operator **~** yields the one's-complement of its operand. The operand must be of type **int**; each 1-bit of the operand is converted to a 0-bit and vice versa.

2.4.3 Binary Expressions

Binary expressions include all operators evaluating two operands. These operations are left-associative, i.e., they are evaluated from left to right.

Product

The operators for multiplication and division produce a product expression. The usual arithmetic type conversions are performed.

The binary **`*`** operator indicates multiplication. The resulting value of this operation is the product of its two operands. The multiplication operator is associative and commutative, and expressions with several multiplications at the same level can be rearranged arbitrarily.

The binary **`/`** operator indicates division. The resulting value emerges from dividing the first operand (dividend) by the second operand (divisor). Integer division truncates any fractional part. A zero value is not allowed for the divisor since division by zero is not permissible.

The binary **`%`** (modulus) operator yields the remainder of dividing the first operand (dividend) by the second operand (divisor); floating point operand values are not allowed, and the divisor must not be zero. The **`%`** operator can be utilized as in

```
febdays = (year%4==0 && year%100!=0 || year%400==0) ? 29 : 28 ;
```

where the value of **`year`** is tested on leap-year match, and the value of **`febdays`** is set accordingly.

Sum

The operators for addition and subtraction produce a sum expression. The usual type conversions are performed.

The binary **`+`** (plus) operator indicates addition. If this operator is applied on numerical operands, then the result is the sum of its two operands, and the operation is commutative; if the add operator is applied on **`string`** operands, then the result is a string value generated by appending the second **`string`** to the first **`string`**, and the operation is not commutative.

The binary **`-`** (minus) operator indicates subtraction. The resulting value of this operation is the difference of its operands. The second operand is subtracted from the first.

Shift Operation

The binary shift operators **`<<`** and **`>>`** can be applied to integer operands. They perform a left (**`<<`**) or right (**`>>`**) shift of their first operand by the number of bit positions specified with the second operand. Vacated significant bits of the first operand are filled with 0-bits. The result value of a bit-shift operation is undefined, if the second operand is negative. If the second operand is zero (i.e., 0 shift operations requested), then the first operand leaves unchanged. A right-shift by one bit corresponds with a (fast) integer division by 2; a left-shift by one bit corresponds to a (fast) multiplication by two; a left-shift by two bits corresponds with a multiplication by four, etc.

Comparison

The resulting value of the binary comparison operators **`<`** (less than), **`<=`** (less equal), **`>`** (greater than) and **`>=`** (greater equal) is the **`int`** value 1, if the specified comparison relation is true for the two operands; otherwise the result value is 0. The comparison operators can be applied on **`string`** operands directly.

Equivalence

The resulting value of the binary equivalence operators **`==`** (equal) and **`!=`** (not equal) is the **`int`** value 1, if the specified equality relation is true for the two operands; otherwise the result value is 0. The equivalence operators can be applied on **`string`** operands directly; they correspond to the comparison operators, but they have less precedence.

Bitwise AND

The binary bitwise AND operator **&** applies to integer operands only; the usual arithmetic type conversions are performed. The result is the bitwise AND function of its operands. This operator is associative and commutative and expressions involving **&** can be rearranged.

Bitwise Exclusive OR

The binary bitwise exclusive OR operator **^** applies to integer operands only; the usual arithmetic type conversions are performed. The result is the bitwise exclusive OR (XOR) function of its operands. This operator is associative and commutative, and expressions involving **^** can be rearranged.

Bitwise Inclusive OR

The binary bitwise inclusive OR operator **|** applies to integer operands only; the usual arithmetic type conversions are performed. The result is the bitwise inclusive OR function of its operands. This operator is associative and commutative, and expressions involving **|** can be rearranged.

Logical AND

The logical AND operator **&&** returns the **int** value 1 if both its operands are nonzero or 0 otherwise. This operator strictly guarantees left-to-right evaluation; i.e., the second operand is not evaluated if the value of the first operand is zero, such that in an expression like

```
x<100 && fct(x)
```

the **fct** function is only called if the value of **x** is less than 100.

Logical OR

The logical OR operator **||** returns the **int** value 1 if either of its its operands is nonzero or 0 otherwise. This operator strictly guarantees left-to-right evaluation; i.e. the second operand is not evaluated if the value of the first operand is nonzero, such that in an expression like

```
test1() || test2()
```

the **test2** function is only called if the **test1** function returns zero.

Conditional Evaluation

The ternary operator **?:** is the conditional evaluation; this operation is right-associative, i.e., it is evaluated from right to left. The first expression is evaluated and if it is nonzero, the result is the value of the second expression, otherwise that of third expression. Usual type conversions are performed to bring the second and third expressions to a common type. Only one of the second and third expression is evaluated. An expression for assigning a conditional expression value to a result as in

```
result = logexpr ? trueexpr : falsexpr ;
```

is equivalent to the following control structure:

```
if (logexpr)
      result = trueexpr ;
else
      result = falseexpr ;
```

The following example utilizes the conditional expression operator to calculate the maximum of two values:

```
maxval = (val1>=val2) ? val1 : val2 ;
```

Assignments

User Language provides a series of assignment operators, all of which are right-associative. All assignment operators require an unary expression as their left operand; the right operand can be an assignment expression again. The type of the assignment expression corresponds to its left operand. The value of an assignment operation is the value stored in the left operand after the assignment has taken place. The binary **=** operator indicates the simple assignment; the binary operators ***=**, **/=**, **%=**, **+=**, **-=**, **>>=**, **<<=**, **&=**, **^=** and **|=** indicate a compound assignment expression. A compound assignment of general form as in

```
expr1 <operator>= expr2
```

is equivalent with the expression

```
expr1 = expr1 <operator> (expr2)
```

where, however, expr1 is evaluated only once (consider the parentheses round expression expr2). An assignment expression sequence as in

```
a = 5 ;
b = 3-a ;
c = b+a ;
c -= a *= b += 4+2*a ;
```

stores the values 60, 12 and -57 to the variables **a**, **b** and **c**, respectively.

2.4.4 Expression List

Each expression can consist of a list of comma-separated binary expressions. A pair of expressions separated by comma is evaluated left-to-right and the value of the left expression is discarded. The type and value of the result are the type and value of the right operand. The comma operator can be utilized as in

```
c -= (a=5, b=3-a, c=b+a, a*=b+=4+2*a) ;
```

where the values 60, 12 and -57 are stored to the variables **a**, **b** and **c**, respectively. In contexts where the comma is given a special meaning, e.g., in a list of actual function parameters and lists of initializers, the comma operator can only appear in parentheses; e.g., the function call

```
fct ( x, (y=8, y*25.4), z )
```

has three arguments, the second of which has the value 203.2.

2.4.5 Precedence and Order of Evaluation

Table 2-4 summarizes the rules for precedence and associativity of all **User Language** operators. Operators on the same line have the same precedence; rows are in order of decreasing precedence.

Table 2-4: Operator Precedence and Order of Evaluation

Operation	Operator(s)	Associativity
Primary	() [] .	left to right
Unary	! ~ ++ -- -	right to left
Product	* / %	left to right
Sum	+ -	left to right
Shift	<< >>	left to right
Comparison	< <= > >=	left to right
Equality	== !=	left to right
Bit And	&	left to right
Bit Xor	^	left to right
Bit Or	|	left to right
Logical And	&&	left to right
Logical Or	||	left to right
Conditional	?:	right to left
Assignment	= += -= etc.	right to left
Expression List	,	left to right

2.5 Control Structures

This section describes the control flow constructions provided by **Bartels User Language**. Control flow statements specify the order in which computations are processed. According to the principles of structured programming **Bartels User Language** distinguishes between concatenation (sequential program element), alternation and repetition (CAR - Concatenation, Alternation, Repetition).

2.5.1 Concatenations

Statements

A statement consists of an expression (see chapter 2.4) followed by a semicolon (**;**), as in

```
tabulator = '\t' ;
distance = sqrt(a*a+b*b) ;
filename += extension = ".ddb" ;
++ary[i] ;
printf("Part %s ;\n",partname) ;
```

The semicolon is a statement terminator. An empty statement is encountered by

```
;
```

where the expression at the left of the semicolon is omitted. Empty statements can be used to define dependent (dummy) statements (e.g., inside loops). A statement is indicated as dependent statement, if it is context-dependent to an alternation or a repetition (see below).

Bartels User Language allows the specification of statements without side-effect as in

```
27+16.3;
++11;
```

Statements without side-effect are worthless since they neither change any variable value by assignment nor do they activate any function. I.e., **User Language Compiler** issues a warning message if a statement without side-effects is encountered.

Blocks

A block consists of a sequence of declarations (see chapter 2.3.2) and statements and is enclosed with braces (**{** and **}**). I.e., the braces apply for grouping declarations and statements together into a compound statement or block, which then is syntactically equivalent to a single statement. Compound statements are most commonly used at the definition of functions or for grouping multiple dependent statements of an alternation or repetition.

2.5.2 Alternations

Alternations make decisions according to a special expression value in order to branch to the execution of corresponding dependent (compound) statements.

if- and if-else Statement

The formal syntax of the **if** statement is

```
if (expression)
      statement
```

where the dependent statement of the **if** statement is only executed if the expression value is nonzero (i.e., a value different from 0 or the empty string on **string** expressions). The formal syntax of the **if-else** statement is

```
if (expression)
      statement1
else
      statement2
```

where the **if** expression is evaluated, and **statement1** is executed if the expression value is nonzero or else **statement2** is executed if the expression value is zero. Dependent statements of an **if** and/or **if-else** statement can be **if** or **if-else** statements again, i.e., **if** and **if-else** statements can nest as in

```
if (expression)
      statement
else if (expression) {
      if (expression)
              statement
      }
else if (expression)
      statement
:
else
      statement
```

Since the **else** part of an **if-else** statement is optional, there is an ambiguity when an **else** is omitted from a nested **if** sequence. This is resolved in a way that the **else** is associated with the closest previous **else**-less **if**. The **if** and/or **if-else** statement can be utilized as in

```
string classname="SCM ";
if (bae_planddbclass()==800)
      classname+="Sheet";
else if (bae_planddbclass()==801 || bae_planddbclass()==803)
      classname+="Symbol/Label";
else if (bae_planddbclass()==802)
      classname+="Marker";
else {
      classname="***INVALID***";
      printf("No valid element loaded!\n");
      }
```

where the class of the currently loaded SCM element is determined and the value of the variable **classname** is set accordingly.

switch Statement

The **switch** statement is a special multi-way decision maker that tests whether an expression matches one of a number of other expressions, and branches accordingly. The formal syntax of the **switch** statement is

```
switch (expression)
      statement
```

Each dependent statement of the **switch** statement can be preceded with an arbitrary number of **case** labels of the form

```
case expression :
```

or

```
default :
```

The statements between the **case** labels are strictly dependent to the closest previous **case** label. The dependent statements of a **case** label are only executed if the value of the **case** expression matches the value of the **switch** expression. The **default** label specifies an arbitrary value, i.e., the statements following to the **default** label is always executed. **case** labels do not have any effect on the sequence of computing (the execution continues as if there is no **case** label). The **break** statement (see also chapter 2.5.4) can be used in a **case** segment to leave the **switch** control structure. The **switch** statement can be utilized as in

```
string classname="SCM ";
switch (bae_planddbclass()) {
      case 800 :
      classname+="Sheet";
      break;
      case 801 :
      case 803 :
      classname+="Symbol/Label";
      break;
      case 802 :
      classname+="Marker";
      break;
      default :
      classname="***INVALID***";
      printf("No valid element loaded!\n");
      }
```

where the class of the currently loaded SCM element is determined and the value of the variable **classname** is set accordingly.

2.5.3 Repetitions

Repetitions are control structures forming a loop for the repetitive computing of certain parts of a program. Each repetitive statement provides a method for testing a certain condition in order to end the processing of the loop. If a program runs into a loop, where the loop-end condition is never reached, then the control flow cannot be passed back to the caller of the program (i.e., the program runs "forever"). This is a fatal programming error, which the **User Language Compiler** can recognize under certain conditions.

while Statement

The formal syntax of the **while** statement is

```
while (expression)
      statement
```

where the dependent statement is repeated until the value of the **while** expression is zero (0 or empty string for **string** expressions). The **while** statement can be utilized as in

```
// ASCII file view
main()
{
      string fname;           // File name
      int fh;                 // File handle
      string curstr="";       // Current input string
      // Set the file error handle mode
      fseterrmode(0);
      // Print the program banner
      printf("ASCII FILE VIEWER STARTED\n");
      // Repeatedly ask for the input file name
      while (fname=askstr("File Name (press RETURN to exit) : ",40)) {
              // Open the input file
              printf("\n");
              if ((fh=fopen(fname,0))==(-1)) {
                      printf("File open failure!\n");
                      continue;
                      }
              // Get the current input string
              while (fgets(curstr,128,fh)==0)
                      // Print the current input string
                      puts(curstr);
              // Test on read errors; close the file
              if (!feof(fh) || fclose(fh))
                      // Read or close error
                      break;
              }
}
```

where the contents of user-selectable files are listed to the terminal. The **continue** statement (see also chapter 2.5.4) causes the next iteration of the **while** loop to begin immediately. The **break** statement (see also chapter 2.5.4) provides an immediate exit from the **while** loop.

do-while Statement

The formal syntax of the **do-while** statement is

```
do
      statement
while (expression);
```

where the dependent statement is repeated until the value of the **do-while** expression is zero (0 or empty string for **string** expressions). The dependent statement always is executed at least once (contrary to **while**).

for Statement

The formal syntax of the **for** statement is

```
for (expression1; expression2; expression3)
      statement
```

which is equivalent to

```
experession1;
while (expression2) {
      statement;
      expression3;
      }
```

where **expression1** is evaluated, and then the dependent statement is executed and **expression3** is evaluated until **expression2** is zero. I.e., **expression1** typically is used for initialization, **expression2** applies the loop-end test and **expression3** performs something like an increment. Any of the three expressions can be omitted, although the semicolons must remain. The **for** statement can be utilized as in

```
void strwords(string s)
{
      string strlist[];
      int strcount,i,j;
      char c;
      for ( strcount=0,j=0,i=0 ; c=s[i++] ; ) {
              if (c==' ' || c=='\t') {
                      if (j>0) {
                              strcount++;
                              j=0;
                              }
                      continue;
                      }
              strlist[strcount][j++]=c;
              }
      for ( i=strcount ; i>=0 ; i-- )
              printf("%s\n",strlist[i]);
}
```

where the function **strwords** separates the given **string** parameter into words to be stored to a list and printed in reverse order afterwards.

forall Statement

The **forall** statement applies for automatic sequential processing of the currently available elements of an **index** data type. The formal syntax of the **forall** statement is

```
forall (identifier1 of identifier2 where expression)
      statement
```

where identifier1 must refer to an **index** variable type specifying the type of the **forall** index to be processed; the **forall** statement automatically initializes and "increments" this variable. The **forall** loop is terminated after the last element of the **index** list has been processed. The **of** statement of the **forall** statement restricts access to those elements of a currently valid **index** element of the next higher hierarchy level; i.e., the **of** index must refer to an **index** type which allows for the processing of the **forall** **index** type. The **where** expression determines whether the dependent statement should be processed for the current **forall** index (if **where** expression nonzero) or not (if **where** expression zero). Both the **of** statement and the **where** statement are optional; thus the shortest possible form of a **forall** statement is

```
forall (identifier1)
      statement
```

The **forall** statement can be utilized as in

```
index L_CPART part;
index L_CPIN pin;
forall (part where part.USED) {
      forall (pin of part where pin.NET.NAME=="vcc")
              printf("Part %s, Pin %s ;\n",part.NAME,pin.NAME);
```

where the list of part pins connected to the net **vcc** is printed to the terminal. See appendix B of this manual for a description of the **index** variable types.

2.5.4 Program Flow Control

Besides the previously described control flow statements **Bartels User Language** provides some additional structures for controlling the program flow.

break Statement

The formal syntax of the **break** statement is

```
break;
```

The **break** statement must be dependent to a repetitive statement (**while**, **do-while**, **for** or **forall**) or to a **switch** statement (otherwise the Compiler issues an error message). The **break** statement provides an early exit from a repetition, just as from **switch**; i.e., **break** statements cause the innermost enclosing loop (or **switch**) to be exited immediately.

continue Statement

The formal syntax of the **continue** statement is

```
continue;
```

The **continue** statement must be dependent to a repetitive statement (**while**, **do-while**, **for** or **forall**), or otherwise the Compiler issues an error message. The **continue** statement causes the innermost enclosing loop to restart immediately at the beginning of the loop. I.e., in **while** and **do-while** loops, **continue** causes the end-condition to be tested immediately; in **for** loops, **continue** passes control to the execution of the "increment" statement.

Function Call and return Statement

Both the function call facilities and the **return** statement have been introduced already. These features are so important for controlling the program flow that they are worth a brief separate discussion at this place.

A function call is an expression which performs a jump to the first statement of the corresponding function definition. The **return** statement can only be used in functions; it terminates the execution of the currently active function and jumps back to the instruction immediately following to the previous function call (i.e., it passes the control flow back to the caller of the function). The general formal syntax of the **return** statement is

```
return;
```

or

```
return expression;
```

If the **return** statement does not contain any expression, then the return value of the corresponding function is undefined. Otherwise the function return value is set to the **return** expression, which must be compatible to the function data type (otherwise, the Compiler issues an error message). If the end of a function block is encountered, and the last statement has not been a **return** statement, then the Compiler automatically produces code corresponding to a valid **return** statement with a function-compatible default (zero) return value. The generation of a default return value is only possible for basic function data types I.e., a function definition as in

```
struct structname functionname() { }
```

causes a Compiler error message since the **Bartels User Language Interpreter** would encounter a memory protection fault when trying to access any of the elements of this return value.

2.6 Preprocessor Statements

The **Bartels User Language Compiler** contains a preprocessor capable of processing special preprocessor statements. Preprocessor statements must start with a hash (#) and they are delimited by the end of the corresponding source code line. Preprocessor statements can appear anywhere and have effect which lasts (independent of scope) until the end of the source code program file.

2.6.1 File Inclusion

Bartels User Language provides the **#include** statement as known from C. The formal syntax of the **#include** statement is

```
#include "filename" EOLN
```

where **filename** must be the name of a **User Language** source code file. This statement is terminated by end-of-line. An **#include** statement causes the replacement of that statement by the entire contents of the specified **User Language** source code file. Such inclusion files can contain general definitions frequently used for different programs. It is a good idea to use include files in order to reduce the expenditure of software maintenance. **#include** statements can be nested, unless they do not refer identical file names (data recursion).

When including source code files one should consider, that not all of the therein contained definitions are really needed by a program; thus, it is recommended to run the **User Language Compiler** with the optimizer to eliminate redundant parts of the program.

The following example shows the include file **baecall.ulh**, which contains the function **call** for activating **AutoEngineer** menu functions:

```
// baecall.ulh -- BAE menu call facilities
void call(int menuitem)       // Call a BAE menu function
{
    // Perform the BAE menu call
    if (bae_callmenu(menuitem))
    {
        // Error; print error message and exit from program
        perror("BAE menu call fault!");
        exit(-1);
    }
}
```

The following example shows the source code of the **ZOOMALL** program for executing the **AutoEngineer** Zoom All command; the program utilizes the **call** function from the included **baecall.ulh** source file:

```
// ZOOMALL -- Call the BAE Zoom All command
#include "baecall.ulh"        // BAE menu call include
main()
{
    // Call Zoom All
    call(101);
}
```

2.6.2 Constant Definition

A preprocessor statement of the form

```
#define IDENT constexpr EOLN
```

causes the Compiler to replace subsequent instances of the IDENT identifier with the value of the given constant expression (constexpr). This statement is terminated by end-of-line. The features introduced by the **#define** statement are most valuable for definition of substantial constants.

A constant definition introduced by **#define** is valid to the end of the program text unless deleted by a preprocessor statement of the form

```
#undef IDENT EOLN
```

A special **#define** statement form is given by

```
#define IDENT EOLN
```

where just a name is defined. The existence or no-existence of such a name definition can be checked with the **#ifdef** and **#ifndef** preprocessor statements (see also chapter 2.6.3).

The **#define** statement can be applied as in the following examples:

```
#define DDBCLASSLAYOUT 100
#define mmtoinch 25.4
#define inchtomm 1.0/mmtoinch
#define REPABORT "Operation aborted."
#define ERRCLASS "Operation not allowed for this element!"
#define GERMAN  1
#define ENGLISH 0
#define LANGUAGE GERMAN
#define DEBUG
```

2.6.3 Conditional Compilation

A preprocessor statement of the form

```
#if constexpr EOLN
```

causes the Compiler to check, whether the specified constant expression is nonzero. A preprocessor statement of the form

```
#ifdef IDENT EOLN
```

causes the Compiler to check, whether the name specified with the identifier is defined (through a **#define** statement). A preprocessor statement such as

```
#ifndef IDENT EOLN
```

causes the Compiler to check, whether the name specified with the identifier is undefined.

The source code lines following to **#if**, **#ifdef** or **#ifndef** are only compiled if the checked condition is true. Otherwise, the the corresponding source code lines are merely checked for correct syntax.

An **#if** preprocessor statement can optionally be followed by a preprocessor statement of the form

```
#else EOLN
```

A preprocessor statement of the form

```
#endif EOLN
```

terminates the if construct. If the checked condition is true then the source code lines between **#else** and **#endif** are not compiled. If the checked condition is false then the source code lines between if-statement and **#else** (or **#endif** on lacking **#else**) are not be compiled. Such **#if** preprocessor constructs can nest.

The following example illustrates the features of conditional compilation:

```
#define ENGLISH 0
#define GERMAN 1
#define LANGUAGE ENGLISH

#if LANGUAGE == GERMAN
#define MSGSTART "Programm gestartet."
#endif
#if LANGUAGE == ENGLISH
#define MSGSTART "Program started."
#endif

#define DEBUG

main()
{
#ifdef DEBUG
    perror(MSGSTART);
#endif
    :
}
```

2.6.4 BNF Precompiler

A BNF precompiler is integrated to the **Bartels User Language**. This BNF precompiler with its corresponding scanner and parser functions can be utilized to implement programs for processing almost any foreign ASCII file data format.

Each **User Language** program text can contain up to one preprocessor statement of the form

```
#bnf { ... }
```

which can cover an arbitrary number of source code lines. The **#bnf** statement activates the BNF Precompiler of the **Bartels User Language**. BNF (Backus Naur Form) is a formalism for describing the syntax of a language. The BNF definition (enclosed with the braces of the **#bnf** statement) of a language consists of a sequence of rule defining the language grammar, i.e., the valid word and/or character sequences for building sentences in this language. A rule of the BNF notation consists of a grammar term (non-terminal symbol) and a sequence of one or more alternative formulations, which are assigned to the grammar term by the operator **:** (to be read as "consists of"); alternative formulations are separated by the **|** operator. A formulation consists of a sequence of grammar terms and input symbols (terminal symbols), whith empty formulations being also allowed. The grammar term of the first rule of a BNF definition is called start symbol. Grammar terms can be referenced recursively, thus allowing for the specification an infinite number of valid sentences of infinite length. Each rule definition must be terminated by the colon operator (**;**).

The language's vocabulary is determined by the terminal symbols specified with the BNF definition. The keywords **IDENT** (identifier), **NUMBER** (numeric constants), **SQSTR** (single quoted string), **DQSTR** (double quoted string), **EOLN** (end of line, **\n**), **EOF** (end of file and/or end of string scanning strings), **EOFINC** (end of include file) and **UNKNOWN** (special character sequence not explicitly defined) stand for generalized terminal symbols from in the corresponding word class. Special user-specific terminal symbols must be quoted. The BNF Precompiler applies automatic classification to assign these terminal symbols to either of the word classes keyword (reserved words consisting of three or more characters), single character operator (consisting of one special character) or double character operator (consisting of two special characters).

Keywords can be specified as in

```
"SECTION"   'ENDSEC'   'Inch'   'begin'   "end"   'include'
```

Double character operators can be specified as in

```
'<='   '++'   "&&"   "+="   '::'   ':='
```

Single character operators can be specified as in

```
'+'   '-'   "*"   "/"   "="   "["   "}"   '#'   '.'
```

Spacings (blanks, tabulator, newline) are not significant for defining a grammar; they serve just for separating adjacent symbols. Comments belong to the spacing token class as well. For comment type definitions, it is necessary to define comment delimiter operators. On default, the BNF Precompiler assigns the operators **/*** (comment start recognition) and ***/** (comment end recognition) for comments which can span multiple lines. This assignment can be changed at the beginning of the BNF definition. The command for the default setting is:

```
COMMENT ("/*", "*/") ;
```

Omitting the second parameter from the **COMMENT** statement as in

```
COMMENT ('//');
```

configures comments which extend to the end of the line. Please note that the **COMMENT** default setting is reset if a **COMMENT** statement is added to the BNF definition. I.e., the following statements must both be added at the beginning of the BNF definition to configure **/*** and ***/** for multi-line comments and **//** for comments to end-of-line:

```
COMMENT ('/*', '*/');
COMMENT ('//');
```

A special feature of the **Bartels User Language** BNF Precompiler is the possibility of defining an action for each grammar term and/or input symbol of a formulation. An action is specified by appending the (parentheses-enclosed) name of a user function to the symbol. The parser automatically calls the referenced user function upon recognition of the corresponding input symbol. These user functions must be properly defined with data type **int**; their return value must be (-1) on error or zero otherwise. Up to one action function parameter of type **int** is allowed; this parameter must be specified as parentheses-enclosed integer constant after the function name in the BNF definition.

See the **Bartels User Language** syntax description in chapter 2.7 for a detailed description of the BNF precompiler syntax definition.

The BNF Precompiler compiles the BNF definition to **User Language** machine code, which represents a state machine for processing text of the defined language. This state machine can be activated with either of the **Bartels User Language** system functions **synparsefile** and/or **synparsestring**. **synparsefile** activates a parser for processing a file name specified input file, whilst **synparsestring** can be used to process strings rather than file contents; the referenced parser action functions are automatically called as required. The **synscanline** and **synscanstring** system functions can be utilized in these parser action functions for querying the current input scan line number and the input scan string. The current scan string can be subjected to semantic tests. The **synparsefile** and/or **synparsestring** functions are terminated if the end of the input file and/or the input string terminator is reached or if a syntax error (or a semantic error encountered by a parser action function) has occurred.

For completeness reasons the system functions **synscnaeoln**, **synscanigncase** and **synparseincfile** are still to be mentioned. The **synscaneoln** scan function is used to enable and/or disable the BNF parser's end-of-line recognition, which is disabled on default. The EOLN terminal symbol of a BNF definition can be recognized only if the EOLN recognition is activated with the **synscaneoln** function. The **synscanigncase** scan function is used to enable and/or disable the BNF parser's case-sensitivity when scanning keyords. The **synparseincfile** function can be utilized for processing include files. The parser starts reading at the beginning of the name-specified include file when calling the **synparseincfile** function. An **EOFINC** terminal symbol is returned if the end of an include file is reached, and reading resumes where it was interrupted in the previously processed input file. The **EOFINC** terminal symbol is obsolete if the **synparseincfile** function is not used.

See appendix C for a detailed description of the **Bartels User Language** BNF scanner and parser system functions.

The usage of the BNF Precompiler is illustrated by the following **User Language** example program. The purpose of this program is to read part placement data from an ASCII file and to perform the corresponding placement on the layout currently loaded to the **Bartels AutoEngineer Layout Editor**. The input placement data is supposed to be organized according to the following example:

```
// This is a comment @

LAYOUT    # This is a comment extending to the end of line

UNITS {
    LENGTH = ( 1.0 INCH ) ;
    ANGLE  = ( 1.0 DEGREE ) ;
    }

PLACEMENT {
    'ic1' : 'dil16' {
        POSITION = (0.000,0.000) ;
        ROTATION = 0.000 ;
        MIRROR   = 0 ;
        }
    'ic2' : 'dil16' {
        POSITION = (2.250,0.100) ;
        }
    'ic3' : 'dil16' {
        POSITION = (1.000,0.394) ;
        ROTATION = 23.500 ;
        }
    'ic4' : 'so16' {
        POSITION = (0.000,0.700) ;
        ROTATION = 0.000 ;
        MIRROR   = 1 ;
        }
    }

END
```

The following listing shows a program which utilizes the BNF Precompiler and the corresponding scanner/parser functions in order to load placement data from external files according to the example above:

```
// READLPLC -- Read Layout Placement from ASCII File
//_____________________________________________________________
// BNF input syntax definition
#bnf {
      COMMENT ("//", "@") ;
      COMMENT ("#") ;
      placefile
              : "LAYOUT" placeunits placedata "END"
              ;
      placeunits
              : "UNITS" "{"
                "LENGTH" "=" "(" floatnum placelengthunit ")" ";"
                "ANGLE" "=" "(" floatnum placeangleunit ")" ";"
                "}"
              ;
      placelengthunit
              : "INCH" (p_unitl(1))
              | "MM" (p_unitl(2))
              | "MIL" (p_unitl(3))
              ;
      placeangleunit
              : "DEGREE" (p_unita(1))
              | "RAD" (p_unita(2))
              ;
      placedata
              : "PLACEMENT" "{" placecommands "}"
              ;
      placecommands
              : placecommands placecommand
              |
              ;
      placecommand
              : identifier (p_pname) ":" identifier (p_plname)
                "{" placepos placerot placemirror "}" (p_storepart)
              ;
      placepos
              : "POSITION" "="
                "(" floatnum (p_px) "," floatnum (p_py) ")" ";"
              ;
      placerot
              : "ROTATION" "=" floatnum (p_pa) ";"
              |
              ;
      placemirror
              : "MIRROR" "=" NUMBER (p_pm) ";"
              |
              ;
      identifier
              : SQSTR (p_ident)
              ;
      floatnum
              : NUMBER (p_fltnum(0))
              | "-" NUMBER (p_fltnum(1))
              ;
}
```

```
//_____________________________________________________________________
// Globals
double plannx=bae_planwsnx(); // Element origin X coordinate
double planny=bae_planwsny(); // Element origin Y coordinate
double lenconv;               // Length conversion factor
double angconv;               // Angle conversion factor
string curpn;                 // Current part name
string curpln;                // Current physical library name
double curx,cury;             // Current coordinates
double cura = 0.0;            // Current angle (default: 0.0)
int curm = 0;                 // Current mirror flag (default: 0)
string curid;                 // Current identifier
double curflt;                // Current float value
struct partdes {              // Part descriptor
      string pn;              //  Part name
      string pln;             //  Physical library name
      double x,y;             //  Coordinates
      double a;               //  Angle
      int m;                  //  Mirror flag
      } pl[];                 // Part list
int pn=0;                     // Part count

//_____________________________________________________________________
// Main program
main()
{
      string fname;           // Input file name
      // Test if layout loaded
      if (bae_planddbclass()!=100)
            errormsg("Command not allowed for this element!","");
      // Get and test the placement file name
      if ((fname=askstr("Placement File : ",40))=="")
            errormsg("Operation aborted.","");
      // Parse the placement file
      perror("Reading placement data...");
      parseerr(synparsefile(fname),fname);
      // Perform the placement
      placement();
      // Done
      perror("Operation completed without errors.");
}
```

```
//_____________________________________________________________________
// Part list management and placement

void gcpart()
// Get or create some part list entry
{
      index L_CPART cpart;    // Part index
      index L_NREF nref;      // Named reference index
      int slb=0;              // Search lower boundary
      int sub=pn-1;           // Search upper boundary
      int idx;                // Search index
      int compres;            // Compare result
      // Loop until search area empty
      while (slb<=sub) {
              // Get the search index
              idx=(slb+sub)>>1;
              // Get and test the compare result
              if ((compres=strcmp(curpn,pl[idx].pn))==0)
                      errormsg("Multiple defined part '%s'!",curpn);
              // Update the search area
              if (compres<0)
                      sub=idx-1;
              else
                      slb=idx+1;
              }
      // Check if part is placed already
      forall (nref where curpn==nref.NAME)
              // Part already placed; abort
              return;
      // Check net list consistence
      forall (cpart where curpn==cpart.NAME) {
              // Check the plname
              if (curpln!=cpart.PLNAME)
                      // Netlist definition mismatch
                      errormsg("Wrong part macro name '%s'!",curpln);
              // Done
              break;
              }
      // Insert the new entry to the part list
      pn++;
      for (idx=pn-2;idx>=slb;idx--)
              pl[idx+1]=pl[idx];
      pl[slb].pn=curpn;
      pl[slb].pln=curpln;
      pl[slb].x=curx;
      pl[slb].y=cury;
      pl[slb].a=cura;
      pl[slb].m=curm;
}

void placement()
// Perform the placement
{
      int i;                  // Loop control variable
      // Iterate part list
      for (i=0;i<pn;i++) {
              // Place the part
              if (ged_storepart(pl[i].pn,pl[i].pln,
               pl[i].x,pl[i].y,pl[i].a,pl[i].m))
                      errormsg("Error placing part '%s'!",pl[i].pn);
              }
}
```

```
//_____________________________________________________________________
// Error handling

void parseerr(status,fn)
// Handle a syntax/parser error
int status;                   // Scan status
string fn;                    // File name
{
      string msg;             // Error message
      // Evaluate the scan status
      switch (status) {
              case 0 : // No error
              return;
              case 1 :
              msg="No BNF definition available!";
              break;
              case 2 :
              msg="Parser already active!";
              break;
              case 3 :
              sprintf(msg," Error opening file '%s'!",fn);
              break;
              case 4 :
              msg="Too many open files!";
              break;
              case 5 :
              sprintf(msg,"[%s/%d] Fatal read/write error!",
               fn,synscanline());
              break;
              case 6 :
              sprintf(msg,"[%s/%d] Scan item '%s' too long!",
               fn,synscanline(),synscanstring());
              break;
              case 7 :
              sprintf(msg,"[%s/%d] Syntax error at '%s'!",
               fn,synscanline(),synscanstring());
              break;
              case 8 :
              sprintf(msg,"[%s/%d] Unexpected end of file!",
               fn,synscanline());
              break;
              case 9 :
              sprintf(msg,"[%s/%d] Stack overflow (BNF too complex)!",
               fn,synscanline());
              break;
              case 10 :
              sprintf(msg,"[%s/%d] Stack underflow (BNF erroneous)!",
               fn,synscanline());
              break;
              case 11 :
              sprintf(msg,"[%s/%d] Error from parse action function!",
               fn,synscanline());
              break;
              default :
              sprintf(msg,"Unknown parser error code %d!",status);
              break;
              }
      // Print the error message
      errormsg(msg,"");
}
```

```
void errormsg(string errfmt,string erritem)
// Print an error message with error item and exit from program
{
      string errmsg;          // Error message string
      // Build and print the error message string
      sprintf(errmsg,errfmt,erritem);
      perror(errmsg);
      // Exit from program
      exit(-1);
}

//_____________________________________________________________________
// Parser action routines

int p_unitl(code)
// Handle the length units definition request
// Returns : zero if done or (-1) on error
{
      // Set the length conversion factor
      switch (code) {
              case 1 : lenconv=cvtlength(curflt,1,0); break; // Inch
              case 2 : lenconv=cvtlength(curflt,2,0); break; // mm
              case 3 : lenconv=cvtlength(curflt,3,0); break; // mil
              default : return(-1); // Error
              }
      // Return without errors
      return(0);
}

int p_unita(code)
// Handle the angle units definition request
// Returns : zero if done or (-1) on error
{
      // Set the angle conversion factor
      switch (code) {
              case 1 : angconv=cvtangle(curflt,1,0); break; // Deg
              case 2 : angconv=cvtangle(curflt,2,0); break; // Rad
              default : return(-1); // Error
              }
      // Return without errors
      return(0);
}

int p_storepart()
// Handle the store part request
// Returns : zero if done or (-1) on error
{
      // Get or create the part list entry
      gcpart();
      // Re-init the current angle and mirror mode
      cura=0.0;
      curm=0;
      // Return without errors
      return(0);
}

int p_pname()
// Receive a part name
// Returns : zero if done or (-1) on error
{
      // Store the current part name
      strlower(curpn=curid);
      // Return without errors
      return(0);
}

int p_plname()
// Receive a physical library name
// Returns : zero if done or (-1) on error
{
      // Store the current physical library name
```

```
      strlower(curpln=curid);
      // Return without errors
      return(0);
}

int p_px()
// Receive a part X coordinate
// Returns : zero if done or (-1) on error
{
      // Store the current part X coordinate
      curx=curflt*lenconv+plannx;
      // Return without errors
      return(0);
}

int p_py()
// Receive a part Y coordinate
// Returns : zero if done or (-1) on error
{
      // Store the current part Y coordinate
      cury=curflt*lenconv+planny;
      // Return without errors
      return(0);
}

int p_pa()
// Receive a part angle
// Returns : zero if done or (-1) on error
{
      // Store the current part angle
      cura=curflt*angconv;
      // Return without errors
      return(0);
}

int p_pm()
// Receive a part mirror flag
// Returns : zero if done or (-1) on error
{
      // Get and store the current part mirror flag
      curm=atoi(synscanstring())==0?0:1;
      // Return without errors
      return(0);
}

int p_ident()
// Receive an identifier
// Returns : zero if done or (-1) on error
{
      // Store the current string
      curid=synscanstring();
      // Return without errors
      return(0);
}
```

```
int p_fltnum(negflag)
// Receive a float value
// Returns : zero if done or (-1) on error
int negflag;                  // Negative number flag
{
      // Get the current float value
      curflt=atof(synscanstring());
      // Set negative on request
      if (negflag)
              curflt*=(-1);
      // Return without errors
      return(0);
}

//_____________________________________________________________________
// User Language program end
```

2.6.5 Program Caller Type and Undo Mechanism

Program Caller Type Setting

The **#pragma** preprocessor statement can be used to set the caller type of the compiled **User Language** program. This feature can be used to relax or restrict the compatibility of the **User Language** program at compile time, no matter whether module-specific system functions and/or index variable types are refered or not. The following table lists the possible caller type specifications (see also Appendix A.1.2).

Caller Type	Valid Interpreter Environment(s)
ULCALLERSTD	all BAE program modules
ULCALLERCAP	all Schematic Capture program modules
ULCALLERSCM	**Schematic Editor**
ULCALLERLAY	all Layout program modules
ULCALLERGED	**Layout Editor**
ULCALLERAR	**Autorouter**
ULCALLERCAM	**CAM Processor**
ULCALLERCV	**CAM View**
ULCALLERICD	all IC Design program modules
ULCALLERCED	**Chip Editor**

The

```
#pragma ULCALLERSTD
```

preprocessor statement forces the compiled **User Language** program caller type setting to standard (STD). The **Incompatible index/function reference(s)!** **User Language Compiler** error is suppressed. **User Language** programs compiled with the above statement can be called in any **Bartels User Language Interpreter** environment, even if the program code contains system functions or index variable types which are not compatible to that Interpreter environment. This allows for the implementation of programs with conditionally executed environment-specific program code. It is up to the program design to prevent from calling incompatible system functions or accessing invalid index variable types; otherwise the **Bartels User Language Interpreter** quits the program with a **UL(Line): System function not available in this environment!** runtime error message.

Configuring Undo Mechanism

On default, the execution of a **User Language** program adds an undo step in the BAE system. The

```
#pragma ULCALLERNOUNDO
```

can be used to prevent the system from adding an undo step for the execution of the compiled program. I.e., by declaring **ULCALLERNOUNDO** for programs which are not performing any operations relevant to the system's undo mechanism, it is possible to avoid redundant undo steps.

2.7 Syntax Definition

The following listing contains a BNF (Backus Naur Form) description of the **User Language** source code syntax. Comments are delimited by **/*** and ***/**. The colon (**:**) corresponds with an assignment operator and is to be read as *"is composed of"*. The vertical line (**|**) designates alternatives and is to be read as *"or"*. Identifiers are denoted by the **IDENT** symbol, constants are marked by **NUMBER** (numeric), **SQSTR** (character) and **DQSTR** (string), respectively. The **EOLN** symbols defines the end-of-line control character. Boldfaced words or character sequences represent the terminal symbols (reserved words and/or operators).

```
/* User Language Source Code Syntax */

/* Program definition */

program
        : progdefs
        ;

progdefs
        : progdefs progdef
        |
        ;

progdef
        : preproccmd
        | typedef
        | storageclass vardef
        | storageclass fctdef
        ;

/* Preprocessor command */

preproccmd
        : #include DQSTR EOLN
        | #define IDENT optexpr EOLN
        | #undef IDENT EOLN
        | #if expression EOLN
        | #ifdef IDENT EOLN
        | #ifndef IDENT EOLN
        | #else EOLN
        | #endif EOLN
        | #bnf { syntaxdef }
        | #pragma pragmaval
        ;

pragmaval
        : ULCALLERSTD
        | ULCALLERCAP
        | ULCALLERSCM
        | ULCALLERLAY
        | ULCALLERGED
        | ULCALLERAR
        | ULCALLERCAM
        | ULCALLERCV
        | ULCALLERICD
        | ULCALLERCED
        | ULCALLERNOUNDO
        ;

/* Type definition */

typedef
        : typedef declaration
        ;
```

```
/* Variable definition */

vardef
        : typespec initdecs ;
        ;

/* Function definition */

fctdef
        : fcttype IDENT ( fctpars ) fctpardecs { cmditems }         ;

fcttype
        : typespec
        | void
        |
        ;

fctpars
        : fctpardefs
        |
        ;

fctpardefs
        : fctpardefs , IDENT
        | IDENT
        ;

fctpardecs
        : fctpardecs vardef
        |
        ;

/* Storage class */

storageclass
        | static
        | structdef
        ;

/* Type specification */

typespec
        : int
        | double
        | char
        | string
        | index IDENT
        | structdef
        | IDENT
        ;

/* Struct definition */

structdef
        : struct IDENT
        | struct IDENT { members }
        | struct { members }
        ;

members
        : members declaration
        | declaration
        ;
```

```
/* Declaration */

declaration
        : typespec decs ;
        ;

decs
        : decs , declarator
        | declarator
        ;

/* Initialized declarator list */

initdecs
        : initdecs , initdec
        | initdec
        |
        ;

initdec
        : declarator
        | declarator = initializer
        ;

/* Declarator */

declarator
        : IDENT
        | declarator [ ]
        ;

/* Initializer */

initializer
        : assignment
        | { initializers }
        ;

initializers
        : initializers , initializer
        | initializer
        ;

/* Command block */

cmdblock
        : { cmditems }
        | cmditem
        ;

/* Command list */

cmditems
        : cmditems cmditem
        |
        ;
```

```
/* Command item */

cmditem
        : preproccmd
        | typedef
        | vardef
        | ifcmd
        | switchcmd
        | forcmd
        | whilecmd
        | docmd
        | forallcmd
        | return optexpr ;
        | break ;
        | continue ;
        | optexpr ;
        ;

/* If control structure */

ifcmd
        : if ( expression ) cmdblock elsecmd
        ;

elsecmd
        : else cmdblock
        |
        ;

/* Switch control structure */

switchcmd
        : switch ( expression ) { caseblocks }
        ;

caseblocks
        : caseblocks caseblock
        |
        ;

caseblock
        : cases cmditems
        ;

cases
        : cases case
        | case
        ;

case
        : case expression :
        | default :
        ;

/* For control structure */

forcmd
        : for ( optexpr ; optexpr ; optexpr ) cmdblock
        ;

/* While control structure */

whilecmd
        : while ( expression ) cmdblock
        ;
```

```
/* Do control structure */

docmd
        : do cmdblock while ( expression ) ;
        ;

/* Forall control structure */

forallcmd
        : forall ( IDENT forallof forallwhere ) cmdblock
        ;

forallof
        : of IDENT
        |
        ;

forallwhere
        : where expression
        |
        ;

/* Expression */

optexpr
        : expression
        |
        ;

expression
        : expression , assignment
        | assignment
        ;

/* Assignment */

assignment
        : unary = assignment
        | unary |= assignment
        | unary ^= assignment
        | unary &= assignment
        | unary <<= assignment
        | unary >>= assignment
        | unary += assignment
        | unary -= assignment
        | unary *= assignment
        | unary /= assignment
        | unary %= assignment
        | conditional
        ;

/* Conditional evaluation */

conditional
        : log_or
        | log_or ? conditional : conditional
        ;

/* Logical OR */

log_or
        : log_and
        | log_and || log_or
        ;
```

```
/* Logical AND */

log_and
        : bit_or
        | bit_or && log_and
        ;

/* Bit OR */

bit_or
        : bit_xor
        | bit_or | bit_xor
        ;

/* Bit Exclusive OR */

bit_xor
        : bit_and
        | bit_xor ^ bit_and
        ;

/* Bit AND */

bit_and
        : equality
        | bit_and & equality
        ;

/* Equivalence comparison */

equality
        : comparison
        | equality == comparison
        | equality != comparison
        ;

/* Comparison */

comparison
        : shift
        | comparison < shift
        | comparison <= shift
        | comparison > shift
        | comparison >= shift
        ;

/* Shift operations */

shift
        : sum
        | shift << sum
        | shift >> sum
        ;

/* Addition and substraction */

sum
        : product
        | sum + product
        | sum - product
        ;
```

```
/* Multiplication and division */

product
        : unary
        | product * unary
        | product / unary
        | product % unary
        ;

/* Unary operators */

unary
        : primary
        | primary ++
        | primary --
        | - unary
        | ! unary
        | ~ unary
        | ++ unary
        | -- unary
        ;

/* Primary operators */

primary
        : IDENT
        | NUMBER
        | SQSTR
        | DQSTR
        | ( expression )
        | IDENT ( optexpr )
        | primary [ expression ]
        | primary . IDENT
        ;

/* BNF Precompiler syntax definition */

syntaxdef
        : commentdef grammar
        ;

commentdef
        : COMMENT ( commentdel commentend ) ;
        |
        ;

commentend
        : , commentdel
        |
        ;

commentdel
        : SQSTR
        | DQSTR
        ;

grammar
        : grammar rule
        | rule
        ;

rule
        : IDENT : forms ;
        | IDENT : forms | ;
        ;
```

```
forms
        : form | forms
        | form
        ;

form
        : form symbol action
        | symbol action
        ;

symbol
        : IDENT
        | SQSTR
        | DQSTR
        | IDENT
        | NUMBER
        | SQSTR
        | DQSTR
        | EOLN
        | EOF
        | EOFINC
        | UNKNOWN
        ;

action
        | ( IDENT ( NUMBER ) )
        | ( IDENT )
        |
        ;

/* BNF syntax description file end */
```

Chapter 3
Programming System

This chapter describes the **Bartels User Language** programming system. It explains how to compile **User Language** programs using the **Bartels User Language Compiler**, and how to run **User Language** programs using the **Bartels User Language Interpreter**.

Contents

Tables

3.1 Conventions

The **Bartels User Language** programming system consists of the **Bartels User Language Compiler** and the **Bartels User Language Interpreter**. The Compiler translates **User Language** source code into **User Language** machine code (programs or libraries), performs static library linking and generates information required for dynamic linking. The **User Language Interpreter** is used for (dynamically linking and) executing **User Language** machine programs. Since the **User Language Comiler** and the **User Language Interpreter** are applied time-independent (i.e., they are implemented in different program modules), the following conventions are required for correct program access.

3.1.1 Program Storage

The **Bartels User Language Compiler** stores successfully translated **User Language** programs (or libraries) to the `ulcprog.vdb` file of the **Bartels AutoEngineer** programs directory. The name of the machine program emerges from the program source code file name. The current **User Language Compiler** version and a caller type coding (determining the compatible **Bartels User Language** interpreter environments) are stored with the machine program code. The Compiler also stores any information required for dynamic link processes (i.e., library linking to be applied at program runtime). When calling a program, the **User Language Interpreter** loads the machine program with the specified name from the `ulcprog.vdb` file of the **Bartels AutoEngineer** programs directory. The program's **User Language Compiler** version is checked to ensure compatibility with the current **User Language Interpreter** version (otherwise the Interpreter might not understand the program structure). The caller type coding stored with the program is used to check whether the index variable types and system functions referenced by the program are implemented in the current **User Language Interpreter** environment. During program load, the Interpreter automatically performs all of the required dynamic link processes. The libraries to be linked with the program are checked for compatibility as well.

3.1.2 Machine Architecture

The machine architecture implemented in the **Bartels User Language** corresponds to a stack machine. The instruction set of this stack machine contains load commands (for loading variable values and/or constants), ALU commands (for activating the arithmetic-logic unit of the machine), store commands (for assignments), function call commands and stack management commands.

The instruction set of this machine is listed in table 3-1. The stack columns provide information on how many stack arguments are required by each instruction and how the stack size changes when executing the instruction.

Table 3-1: User Language Machine Instruction Set

Instruction	Stack Arguments	Stack Change	Instruction Designator
nop	0	0	No operation
add	2	-1	Add
addstr	2	-1	Add string
and	2	-1	And
bnot	1	0	Binary not
cmpeq	2	-1	Compare equal
cmpge	2	-1	Compare greater equal
cmpgt	2	-1	Compare greater
cmple	2	-1	Compare less equal
cmplt	2	-1	Compare less
cmpne	2	-1	Compare not equal
decr	1	0	Decrement
div	2	-1	Divide
divr	2	-1	Divide rest
incr	1	0	Increment
mul	2	-1	Multiply
neg	1	0	Negate
not	1	0	Not
or	2	-1	Or
shl	2	-1	Shift left
shr	2	-1	Shift right
sub	2	-1	Subtract
xor	2	-1	Exclusive or
cast t	1	0	Cast value
castoiv i	3	-2	Cast of index variable
getary	2	-1	Get array element
getidx i	1	1	Get index
getidxof i	3	-1	Get index of
loadas s	1	0	Load stack array element
loadav v	1	0	Load variable array element
loadchr c	0	1	Load character
loaddbl d	0	1	Load double

Instruction	Stack Arguments	Stack Change	Instruction Designator
loadint i	0	1	Load integer
loadiv v	2	-1	Load index variable
loadoiv v	4	-3	Load of index variable
loads s	0	1	Load stack
loadsd s	0	1	Load stack destructive
loadstr s	0	1	Load string
loaduref f	0	1	Load user function reference
loadv v	0	1	Load variable
loadvd v	0	1	Load variable destructive
storeas s	2	-2	Store stack array element
storeav v	2	-2	Store variable array element
stores s	1	-1	Store stack
storev v	1	-1	Store variable
pop s	0	0	Pop stack
popt	1	-1	Pop top of stack
push s	0	0	Push stack
swap s	0	0	Swap stack
xchg s	0	0	Exchange stack
xchgt	2	0	Exchange top of stack
jump p	0	0	Jump always
jumpeq p	2	-1	Jump if stack tops equal
jumpnz p	1	-1	Jump if stack nonzero
jumpz p	1	-1	Jump if stack zero
calls f	0	1	Call system function
callu f	0	1	Call user function
hlt	0	0	Halt program
ret	1	-1	Return (pop optional stack)
stop	0	0	Stop function

3.2 Compiler

The **Bartels User Language Compiler** translates **User Language** source code into **User Language** machine programs and/or **User Language** libraries. **User Language** machine programs can be executed by the **User Language Interpreter**. **User Language** libraries usually are generated from frequently used source code. **User Language** library machine code can be linked to other **User Language** machine code (programs or libraries). Library linking can be done either statically (at compile time by the **User Language Compiler**) or dynamically (at runtime by the **User Language Interpreter**). As an advantage of the **User Language** library concept, frequently used source code needs to be compiled *just once* and can subsequently be referenced through linking, which is much faster than compiling.

3.2.1 Mode of Operation

At the compilation of **User Language** source code the **User Language Compiler** performs comprehensive syntactic and semantic consistency checks, removes redundancies from the program and/or library, and finally produces - in very compact form - a source code equivalent **User Language** machine code (**User Language** program and/or **User Language** library). The built-in linker of the Compiler performs static library linking and generates information for dynamic linking on request. This is applied with the subsequently described sequence of operations.

Syntax Analysis and Semantic Check

The first phase of the compilation process performs syntactic analysis. A parser pass is applied to handle formal problems, i.e., to check whether the sequence of words and symbols from the source code represents a syntactically valid **User Language** program and/or library. With this parser pass, semantic checks are performed in order to ensure consistency and uniqueness of the source's variable, parameter, and function definitions. As a result, this first parser pass (pass 1) generates an internal symbol table, which is required for the semantic test performed in the second parser pass (pass 2). The semantic test includes a context-sensitive analysis of the source code text to suppress the misuse of the defined program objects, i.e., the semantic test checks on the validity of the usage of names as well as the admissibility of the operations on the defined objects.

Machine Code Generation

The source code equivalent machine code is constructed already whilst running the semantic test, i.e., with the second parser pass (pass 2). The machine code generated by the Compiler corresponds with a valid machine program and/or library only if the semantic test was completed without errors.

Linker

The built-in linker of the Compiler performs static library linking and generates information for dynamic linking on request.

The static link process (Compiler option **`-lib`**) binds machine code from required **User Language** libraries to the machine code currently to be translated. The requested libraries are checked for Compiler version compatibility. References to library machine code such as global function and/or variable addresses are resolved with consistency check.

The dynamic link process (Compiler option **`-dll`**) only simulates machine code binding. As a result, dynamic link library relocation tables for resolving references to dynamic link libraries at runtime is stored with the machine code. The information provided with these relocation tables is later used by the **User Language Interpreter** to check library compatibility when performing dynamic link processes at runtime. Note that **User Language** library modifications require the recompilation of all **User Language** programs and/or libraries containing dynamic link requests to the changed library.

Optimizer

The optimizer of the **User Language Compiler** can be activated using the Compiler option **`-O`**. The optimizer frees the machine code from redundancies and modifies the machine code in order make it more efficient. The optimizer recognizes and eliminates unused code segments as well as unreferenced function, variable, and parameter definitions. It changes variable references to constant accesses if possible (Constant Propagation), and it accomplishes algebraic optimizations. In most cases optimization considerably reduces machine code's memory and runtime requirements.

The optimization introduces a very useful side effect: optimizer-modified machine code can be checked for special programming errors, which the Compiler otherwise would not have been able to recognize.

Machine Code Check

After generating the machine code, the Compiler checks it again for fatal errors, which the Compiler eventually is able to recognize by analyzing the machine code. Such errors are division by zero, endless loop constructs and endless-recursive function calls.

Listing File Output

The **User Language Compiler** can be caused optionally to produce a listing file. The specifications in this file can be useful for locating errors occurred at runtime (i.e., with the program execution) only. The complete listing file content is composed of general program and/or library specifications (name, version, caller type), dynamic link request information, static link library reference listings, tables of the definitions (functions, variables, structures, etc.) used throughout the machine code, and the machine code listing (i.e., the list of the machine instructions including source text and machine code line number specifications).

The listing option **`-l`** of the **User Language Compiler** supports different modes for making the output more or less verbose. With this option, it is possible to, e.g., restrict output for **User Language** library documentation purposes (library function reference).

The **`-ld`** option allows for the specification of an alternative output directory for the listing files created with the **`-l`** option. This option is useful when applying **`make`** utilities for automatically compiling modified **User Language** programs as it allows to keep the source directories clean. With the BAE software, a **`makefile`** is provided in the **`baeulc`** directory. This **`makefile`** defines the dependencies between **User Language** programs and include files and works with listing files in a subdirectory (**`lst`**).

Machine Code Storage

With the final phase of the compilation process the machine code generated by the **User Language Compiler** is stored to the **`ulcprog.vdb`** file in the **Bartels AutoEngineer** programs directory. Each machine code is named according to the destination element name specified with the Compiler call (see Compiler options **`-Source`**, **`-cp`**, **`-cl`**). At the storage of machine code special database classes are assigned to **User Language** programs and/or **User Language** libraries, respectively.

Special **User Language Compiler** options allow for the deletion of **User Language** programs (option **`-dp`**) and/or libraries (option **`-dl`**) stored to **`ulcprog.vdb`**. With this option, it is possible to cleanup **`ulcprog.vdb`** from obsolete and/or redundant machine code.

3.2.2 Compiler Call

The translation of an **User Language** program and/or an **User Language** library is started with the **User Language Compiler** call.

Synopsis

The **User Language Compiler** must be called from the operating system shell. The synopsis for calling the Compiler is:

```
ulc [-wcon|-wcoff] [[-S[ource]] srcfile...]
    [-lib libname...] [-dll libname...]
    [{-cp|-cl} [dstname...]]
    [-I[nclude] includepath...] [-D[efine] macroid...]
    [-O[0|1]] [-e[0|1]] [-w[0|1|2|3|4]] [-t[0|1]]
    [-l[0|1|2|3|4|5]] [-ld listingdirectory]
    [-dp prgname...] [-dl libname...]
    [-ulp prgfilename] [-ull libfilename]
    [-log logfilename]
```

On syntactically wrong Compiler calls, the correct ULC command syntax is displayed, and the compilation process is aborted.

Options

Command line options of the **User Language Compiler** consist of the dash (**-**) or slash (**/**) start character followed by the option specification. Single-character option specifications optionally followed by a mode or toggle number are often known as switches or flags. Such special options can be grouped as in **/l2Ow3** or **-O1w3l2**, which both select listing mode 2, activate the optimizer, and set the warning severity level to 3.

Wildcard Option [-wcon|-wcoff]

The wildcard option is used to activate or deactivate wildcard processing at the specification of file and/or element names. On default wildcard processing is activated, i.e., omitting the wildcard option leaves wildcard processing activated. Option **-wcon** can be used for explicitly activating wildcard processing. With wildcard recognition activated, the character **?** can be used for matching any arbitrary character, and the character ***** can be used for matching an arbitrary number of arbitrary characters. Option **-wcoff** can be used to turn off wildcard processing. Wildcard recognition must be deactivated for explicitly processing names containing wildcard characters such as **SCM_?** or **GED_***.

Source File Option [[-S[ource]] srcfile...]

This option is used for specifying the file name(s) containing the source code to be compiled. File name specifications can contain a directory path, i.e., the source file names need not reside in the current directory. Wildcards are supported with the source file name specification if wildcard recognition is activated (see option **-wcon** above). Source file names can be specified with or without file name extension. On source file name specifications without extension the Compiler automatically assumes and/or appends file name extension **.ulc**. I.e., source file names with non-default extension must be specified with their extension, respectively. It is possible to, e.g., generate **User Language** libraries from include files usually named with extension **.ulh**. The type of **User Language** machine code to be generated is designated with either option **-cp** (**User Language** programs; see below) or option **-cl** (**User Language** libraries; see below). The name of the machine code element to be generated is derived from the source file name by stripping the directory path and the file name extension from the source file name. Non-default destination program and/or library element names can be specified with options **-cp** and **-cl** (see below). The **-Source** and/or **-S** option keywords are not required with source file specifications where file names cannot be intermixed with other name specifications, e.g., if source file names are the first names specified on the ULC command line. However the **-Source** (and/or **-S**) option can be used for explicit source file specification to avoid ambiguities in case where source file names are not the first name specifications on the ULC command line. At least one source file specification is required if neither option **-dp** nor option **-dl** (see below) is specified.

Static Link Option [-lib libname...]

The static link **-lib** option requires one or more library name specifications and at least one valid source file specification (see option **-Source** above). The machine code of the libraries specified with the **-lib** option must be available in the **ulcprog.vdb** file of the BAE programs directory, i.e., the required libraries must be compiled before they can be linked. The built-in linker of the **User Language Compiler** binds the machine code of these libraries to the machine code currently to be translated.

Dynamic Link Option [-dll libname...]

The dynamic link option **-dll** requires one or more library name specifications and at least one valid source file specification (see option **-Source** above). The machine code of the libraries specified with the **-dll** option must be available in the **ulcprog.vdb** file of the BAE programs directory, i.e., the required libraries must be compiled before they can be linked. The built-in linker of the **User Language Compiler** stores dynamic link request information with the machine code to enable the **User Language Interpreter** to perform dynamic linking of the requested libraries at runtime.

Create Program/Library Option [{-cp|-cl} [dstname...]]

The create option can be used to designate the type of machine code to be generated. The **User Language Compiler** can create either **User Language** programs or **User Language** libraries. On default, program generation request is assumed, i.e., omitting both the **-cp** and the **-cl** option defaults to **User Language** program creation. Option **-cp** explicitly selects program generation whilst option **-cl** selects library generation; both options must not be used together. On default, the destination element name is derived from the corresponding source file name; both the directory path and the source file name extension are stripped from the source file name to generate the destination element name. Options **-cp** and **-cl** allow for the specification of non-default destination program and/or library names. Only *one*source file specification (see option **-Source**) is allowed when explicitly specifying destination element name(s) with options **-cp** and **-cl**. The machine code generated by the Compiler is stored with the specified destination element name to the **ulcprog.vdb** file in the BAE programs directory. Wildcards are not supported with destination element name specifications. Multiple destination element name specifications can be used to store the machine code of a single source under different names, e.g., to generate programs **SCM_ST**, **GED_ST**, etc. from a single source file named **bae_st.ulh**.

Include Path Option [-I[nclude] includepath...]

The **-Include** (and/or **-I**) option is used for specifying multiple alternate include paths for include file name search. At least one include path argument is required. When encountering an **#include** preprocessor statement, the Compiler first checks the current directory for include file access and then searches the include paths in the sequence specified with the **-Include** option until the requested include file is found.

Define Option [-D[efine] macroid...]

The **-Define** (and/or **-D**) option is used for defining macros at the **User Language Compiler** call. At least one macro identifier is required. This option corresponds with the **#define** preprocessor statement, i.e., macros defined with the **-Define** option can be checked with the **#ifdef** or **#ifndef** preprocessor statements, thus giving more control on conditional compilation to the Compiler.

Optimizer Option [-O[0|1]]

The **-O** option is used to activate or deactivate the optimizer of the **User Language Compiler**. On default the optimizer is deactivated, i.e., omitting this option leaves the optimizer deactivated. Option **-O** or **-O1** activates the optimizer. Option **-O0** explicitly deactivates the optimizer. The optimizer frees the machine code from redundancies, and modifies it to make it more efficient. Optimizing machine code significantly reduces disk space and main memory requirements, and the resulting machine code can be loaded and executed much faster. It is strongly recommended to activate the optimizer.

Error Severity Option [-e[0|1]]

The **-e** option is used for setting the error severity level. On default the error severity level is set to 1, i.e. omitting this option selects error severity level 1. Option **-e0** sets error severity level 0. Option **-e** or **-e1** explicitly sets error severity level 1. Error severity level 1 compiles all specified sources; error severity level 0 causes the Compiler to stop the compilation process on any errors occurred during a single source compilation.

Warning Severity Option [-w[0|1|2|3|4]]

The **-w** option is used for setting the warning severity level in the range 0 to 4. On default, the warning severity level is set to 0, i.e., omitting this option selects warning severity level 0. Omitting explicit level specification with this option as with **-w** defaults to warning severity level 3. Each type of warning defined with the Compiler is assigned to a certain warning severity level. The Compiler only prints warnings with a warning severity level less than or equal the level selected with the **-w** option since higher warning severity levels denote less importance. With this option, it is possible to suppress less important warning messages.

Top Level Warnings Only Option [-t[0|1]]

The **-t** option controls whether warning messages related to the compilation of include files are omitted or not. On default (i.e., if this option is not specified or if its value is set to 0), warning messages related to the compilation of both top level source code files and include files are issued. Setting this option value to 1 prevents the **User Language Compiler** from issuing warning messages related to the compilation of include files, thus simplifying the analysis of warning messages when working with standard include files containing functions and variables which are not used by every program.

Listing Option [-l[0|1|2|3|4|5]]

The **-l** option is used to control the listing file output. Listing modes 0 to 5 can be specified. Mode 0 won't produce any listing output and mode 5 produces the most detailed listing. On default, listing mode 0 is selected, i.e., no listing file is generated if this options is omitted. Omitting explicit listing mode specification with this option as with **-l** defaults to listing mode 5. The listing output file name is derived from the corresponding source code file name, where the original file name extension is replaced with extension **.lst**. The listing file is for user information purposes only, i.e., it is not required by system.

Listing Directory Option [-ld listingdirectory]

The **-ld** option allows for the specification of an alternative output directory for the listing files created with the **-l** option. This option is useful when applying **make** utilities for automatically compiling modified **User Language** programs as it allows to keep the source directories clean. With the BAE software, a **makefile** is provided in the **baeulc** directory. This **makefile** defines the dependencies between **User Language** programs and include files and works with listing files in a subdirectory (**lst**).

Delete Program Option [-dp prgname...]

The **-dp** option is used for deleting previously compiled programs from the **ulcprog.vdb** file in the BAE programs directory. At least one program element name is required. Wildcards are supported with the program element name specification if wildcard recognition is activated (see option **-wcon** above). Warnings are issued when trying to delete non-existent programs. Program deletion is always processed before any source code compilation to avoid compilation process conflicts such as deleting a program immediately after it has been compiled with the same ULC call.

Delete Library Option [-dl libname...]

The **-dl** option is used for deleting previously compiled libraries from the **ulcprog.vdb** file in the BAE programs directory. At least one library element name is required. Wildcards are supported with the library element name specification if wildcard recognition is activated (see option **-wcon** above). Warnings are issued when trying to delete non-existent libraries. Library deletion is always processed before any source code compilation to avoid compilation process conflicts such as deleting a library immediately after it has been compiled with the same ULC call.

Program Database File Name Option [-ulp prgfilename]

On default, the **User Language Compiler** stores **User Language** programs to a file named **`ulcprog.vdb`** in the **Bartels AutoEngineer** programs directory. The **`-ulp`** option can be used to select a different **User Language** program database file.

Library Database File Name Option [-ull libfilename]

On default, the **User Language Compiler** stores **User Language** libraries to a file named **`ulcprog.vdb`** in the **Bartels AutoEngineer** programs directory. The **`-ull`** option can be used to select a different **User Language** library database file.

Log File Option [-log logfilename]

The **User Language Compiler** prints all messages to standard output and to a log file. Log file output is generated to save long message lists which could be generated at the compilation of different sources. On default the log file name is set to **`ulc.log`** (current directory). The **`-log`** option can be used to specify a non-default log file name.

Examples

Compilation of the **User Language** program contained in **ulcprog.ulc** with optimization and warning message output; the produced machine program is stored with the name **ulcprog** to the **ulcprog.vdb** file of the BAE programs directory:

```
ulc ulprog -Ow
```

Compilation of the **User Language** program contained in **ulcprog.ulc** with listing file output (to **ulcprog.lst**); the produced machine program is stored with the name **newprog** to the **ulcprog.vdb** file of the BAE programs directory:

```
ulc ulprog -l -cp newprog
```

Deleting the **User Language** programs named **ulcprog** and **newprog** and all **User Language** libraries with names starting with **test** and ending on **lib** from the **ulcprog.vdb** file of the BAE programs directory:

```
ulc -dp ulprog newprog -dl test*lib
```

Generate **User Language** library **libsll** from source file **libbae.ulh** (optimizer is activated; listing output is directed to file **libbae.lst**):

```
ulc libbae.ulh -cl libsll -l2O
```

Compile all current directory files with extension **.ulc** and statically link the generated program machine codes with library **libsll** (macro **USELIB** is defined for controlling conditional compilation; optimizer is activated):

```
ulc *.ulc -Define USELIB -lib libsll -O
```

Generate libraries **libstd** and **stdlib** from source file **std.ulh** (optimizer is activated, warning severity level is to 2):

```
ulc -w2 -O -cl libstd stdlib -Source std.ulh
```

Generate library **liblay** from source file **\baeulc\lay.ulh** with library **libstd** dynamically linked (optimizer is activated, warning severity level is set to 3, Compiler messages are directed to log file **genlib.rep** instead of **ulc.log**):

```
ulc /wO -cl liblay -S \baeulc\lay.ulh -dll libstd -log genlib.rep
```

Generate programs **laypcr** and **tracerep** from source files **laypcr.old** and **tracerep.ulc** with library **liblay** dynamically linked (optimizer is activated):

```
ulc laypcr.old /dll liblay /cp -O /S tracerep
```

3.2.3 Error Handling

One of the most important Compiler features is the error handling. This is due to the fact, that source codes, which contain errors and/or redundancies are most frequently processed (a correct, redundancy-free program is usually only compiled once). The error and warning messages issued by the Compiler are intended to support the programmer in developing error-free **User Language** programs and/or libraries without redundancies as quickly as possible.

The **User Language Compiler** prints all messages to the screen and to a log file. Log file output is generated to save long message lists which could be generated at the compilation of different sources. On default, the log file name is set to `ulc.log` (current directory); a non-default log file name can be specified using the `-log` Compiler option (see above).

This section lists all error and warning messages defined with the **User Language Compiler**. At the appearance of errors no valid machine code can be produced. Warnings indicate the generation of valid machine code, which, however, might show up with unpredictable side effects at runtime. With each message a line number is included wherever possible to localize the corresponding error. This line number specification refers either to the source code file (denoted by "Ll") or to the machine code (denoted by "Lp").

The warnings messages listed below are preceded with a number enclosed in square brackets. These are not part of the actually printed warnings, but denote the minimum warning severity level to be set with the `-w` option to prompt the **User Language Compiler** to issue the corresponding warnings. Warnings assigned to severity level 0 are always printed, regardless of the selected warning severity level.

General Messages

On syntactically wrong Compiler calls, the correct ULC command syntax is displayed, and the compilation process is aborted.

The following general Compiler messages denote current Compiler actions and/or issue resumes on the compilation process:

```
Deleting programs from "n"...
Program 'n' deleted.
Deleting libraries from "n"...
Library 'n' deleted.
Loading/linking libraries...
Compiling source code file "n"...
Program 'n' successfully created.
Library 'n' successfully created.
Source code file "n" successfully compiled.
e errors, w warnings.
User Language Compiler aborted!
User Language Compilation successfully done.
```

The following Compiler messages indicate general errors regarding the Compiler call, such as missing Compiler runtime authorization, invalid file and/or element name specifications, file access problems or link library access problems:

```
ERROR : Please check your User Authorization!
ERROR : File name "n" is too long!
ERROR : File name "n" contains invalid chars!
ERROR : Element name 'n' is too long!
ERROR : Element name 'n' contains invalid chars!
ERROR : Error writing listing file "n"!
ERROR : Error creating ULC log file "n"!
ERROR : Too many source code files specified!
ERROR : Source code file "n" not found!
ERROR : Library 'n' not found!
ERROR : User Language Library 'n' Version not compatible!
```

The following messages indicate general problems accessing specified directories and/or program or library elements or notify of link library inconveniences:

```
[0] WARNING : Directory 'n' not found/not available!
[0] WARNING : Program 'n' not found!
[0] WARNING : Library 'n' not found!
[0] WARNING : Library 'n' is not optimized!
```

Fatal Errors

The following internal compiler messages either indicate memory management errors or refer to Compiler implementation gaps:

```
(Ll) ERROR : List overflow!
(Ll) ERROR : Out of memory!
(Ll) ERROR : INTERNAL ERROR IN function -- PLEASE REPORT!
```

Parser Errors

The following messages indicate source code file access and/or syntax errors:

```
(Ll) ERROR : Cannot open source file "n"!
(Ll) ERROR : Cannot read source file "n"!
(Ll) ERROR : Source file expression too complex ('s')!
(Ll) ERROR : Source file element too long ('s')!
(Ll) ERROR : Syntax error at 'string' (unexpected symbol)!
(Ll) ERROR : Unspecified syntax analyzer error!
```

Semantic Errors and Warnings

The following errors can be issued whilst performing semantic source code analysis:

```
(Ll) ERROR : Identifier 'n' is too long!
(Ll) ERROR : Character 's' is too long / no character!
(Ll) ERROR : String "s" is too long!
(Ll) ERROR : Numeric value 's' is too long!
(Ll) ERROR : Invalid numeric value 's'!
(Ll) ERROR : Type 'n' not defined!
(Ll) ERROR : Multiple definition of type 'n'!
(Ll) ERROR : Function 'n' not defined!
(Ll) ERROR : Multiple definition of function 'n'!
(Ll) ERROR : Function 'n' is a system function!
(Ll) ERROR : Function parameter 'n' not defined!
(Ll) ERROR : Multiple definition of function parameter 'n'!
(Ll) ERROR : Multiple declaration of function parameter 'n'!
(Ll) ERROR : Variable 'n' not defined!
(Ll) ERROR : Multiple definition of variable 'n'!
(Ll) ERROR : Assignment to constant or result value is attempted!
(Ll) ERROR : Not an array; cannot perform index access!
(Ll) ERROR : Invalid array index specified!
(Ll) ERROR : Array subscript out of range!
(Ll) ERROR : Access to member ('n') of unknown struct!
(Ll) ERROR : Structure 'n' unknown/invalid!
(Ll) ERROR : Multiple definition of structure 'n'!
(Ll) ERROR : Structure member 'n' unknown/invalid!
(Ll) ERROR : Multiple definition of structure member 'n'!
(Ll) ERROR : Index 'n' unknown/invalid!
(Ll) ERROR : Index variable 'n' unknown/invalid!
(Ll) ERROR : 'n' is not an index variable!
(Ll) ERROR : 'forall'-index not defined for 'of'-index 'n'!
(Ll) ERROR : 'continue' not within a loop!
(Ll) ERROR : 'break' not within a loop or 'switch'!
(Ll) ERROR : 'void' function 'n' cannot return a value!
(Ll) ERROR : Function 'n' must 'return' a valid value!
(Ll) ERROR : 'return' expr. not type-compat. to function 'n'!
(Ll) ERROR : Expression not type-compatible to parameter 'n'!
(Ll) ERROR : Expression not type-compatible to variable 'n'!
(Ll) ERROR : Operand not type-compatible to the 'n'-operator!
(Ll) ERROR : Operands not type-compatible to the 'n'-operator!
(Ll) ERROR : Invalid assignment to active loop index variable!
(Ll) ERROR : Invalid 'n'-expression!
(Ll) ERROR : Unknown/undefined function 'n'!
(Ll) ERROR : Function 'n' - not enough parameters specified!
(Ll) ERROR : Function 'n' - parameter not compatible!
(Ll) ERROR : Function 'n' - parameter out of range!
(Ll) ERROR : Invalid '#if-#else-#endif' construct!
(Ll) ERROR : Identifier 'n' is defined as macro!
(Ll) ERROR : Access to void macro 'n'!
(Ll) ERROR : Cannot store BNF to UL library!
(Ll) ERROR : BNF redefined!
(Ll) ERROR : BNF symbol 'n' unknown/undefined!
(Ll) ERROR : BNF production 'n' double defined!
(Ll) ERROR : BNF reduce/reduce conflict at production 'n'!
(Ll) ERROR : BNF terminal symbol 'n' is invalid!
(Ll) ERROR : BNF comment delimiter 's' is invalid!
```

```
(Ll) ERROR : BNF function 'n' not of type 'int'!
(Ll) ERROR : Division by zero is attempted!
(Ll) ERROR : Endless loop!
(Ll) ERROR : Function 'n' - recursive call!
(Lp) ERROR : Stack overflow!
ERROR : End of file reached where '}' has been expected!
ERROR : Incompatible index/function reference(s)!
```

The following warnings can be issued whilst performing semantic source code analysis:

```
[1] (Ll) WARNING : BNF contains no valid productions!
[2] (Ll) WARNING : Function 'n' - default return value used!
[1] (Ll) WARNING : Function 'n' - too many parameters specified!
[2] (Ll) WARNING : Function 'n' - change of parameter n will be ignored!
[2] (Ll) WARNING : Function 'n' - change of parameter 'n' will be ignored!
[2] (Ll) WARNING : Constant 'n'-expression!
[2] (Ll) WARNING : Expression has no side-effects!
[2] (Ll) WARNING : Function 'n', local variable 'n' hides global variable!
[2] (Ll) WARNING : Function 'n', parameter 'n' hides global variable!
[4] (Ll) WARNING : Variable 'n' has not been initialized!
[4] (Ll) WARNING : Macro 'n' redefined!
```

Optimizer Warnings

The following warnings are issued by the optimizer and indicate source code redundancies:

```
[1] (Ll) WARNING : BNF is not referenced!
[2] (Ll) WARNING : Global variable 'n' not referenced!
[2] (Ll) WARNING : Function 'n' not referenced!
[2] (Ll) WARNING : Statement is not reached!
[3] (Ll) WARNING : Function 'n', Local variable 'n' not referenced!
[3] (Ll) WARNING : Function 'n', Parameter 'n' not referenced!
[4] WARNING : Library function 'n' not referenced!
[4] WARNING : Library variable 'n' not referenced!
[4] WARNING : Dynamic Link Library 'n' is not referenced!
```

Database Access Errors

The following messages indicate errors on the storage of the machine program:

```
ERROR : Cannot create database file "n"!
ERROR : Read/write error whilst accessing file "n"!
ERROR : Too many open files!
ERROR : File "n" is not a database/DDB file!
ERROR : File structure is damaged in file "n"!
ERROR : File structure is erroneous in file "n"!
ERROR : Function not available for old format!
ERROR : Database limit exceeded!
ERROR : File "n" is not compatible with program version!
ERROR : Element 'n' not found!
ERROR : Element 'n' exists already!
ERROR : File "n" not found!
ERROR : Record end reached!
ERROR : Unspecified database error!
```

3.3 Interpreter

The **Bartels User Language Interpreter** is integrated to the **Schematic Editor**, the **Layout Editor**, the **Autorouter**, the **CAM Processor**, the **CAM View** module and the **Chip Editor** of the **Bartels AutoEngineer**. I.e., the **Bartels User Language Interpreter** can be used for calling **Bartels User Language** programs from each of these BAE modules.

3.3.1 Mode of Operation

The **User Language Interpreter** is activated by calling a **User Language** program from one of the valid interpreter environments. Any **User Language** program call is processed by applying the subsequently described sequence of operations.

Program Load, Dynamic Link

When calling a **User Language** program, the **User Language Interpreter** first of all must load the required **User Language** machine program with the specified program name from the `ulcprog.vdb` file of the BAE programs directory. The **User Language Interpreter** applies a compatibility check; a consistency check is not necessary since this time-consuming work has been carried out by the **User Language Compiler** already. A **User Language** program is compatible to and executable in the current interpreter environment, if the program's **User Language Compiler** version is equal to the **User Language Interpreter** version, and if the program only references index variable types and system functions which are implemented in the current interpreter environment.

Each **User Language** program can contain dynamic link requests, i.e. requests on binding **User Language** library machine code to program machine code at runtime. Dynamic linking is automatically applied during program load. Each dynamic link library (DLL) must be available, and the definitions (variables, functions, function parameters) provided with each DLL machine code must match its definitions at compile time (otherwise the Interpreter might try to access non-existent or wrong library objects, which would result in undefined behavior or even system crash with design data loss). The built-in linker of the **User Language Interpreter** checks on dynamic link library compatibility and refuses to run the program when encountering any inconveniences (error message `Incompatible index/function reference(s)!`). In case of incompatible DLLs, the **User Language** program must be recompiled.

Program Execution

After loading (and dynamically linking) the **User Language** machine program, the **User Language Interpreter** starts executing the program. Program execution can be understood to be the simulation of the **User Language** machine architecture by processing the instructions of the machine program. Program execution starts with the first machine program instruction and is completed when the program counter refers to a non-existent machine program instruction.

Program Termination

After executing the machine program a program termination must be applied to release the main memory allocated during program execution (Program Cleanup), and to remove the machine program from main memory (Program Unload).

3.3.2 Program Call

When calling a **User Language** program, the name of the program to be executed must be specified. This program name specification can be accomplished either explicitly or implicitly according to one of the subsequently described methods.

Menu Call

The Run User Script function from the File menu is used for explicit **User Language** program calls. The Run User Script function activates a dialog with a program name prompt and a List button for interactive **User Language** program selection:

Keyboard Call

One method of implicit **User Language** program call is provided by pressing special keys of the keyboard whilst working in the BAE function menu, i.e., this type of implicit **User Language** program call is possible at any time unless another interactive keyboard input currently is awaited by the system. The specification of the program name is accomplished implicitly by pressing a special key. Table 3-2 contains the list of key bindings provided with the **User Language** interpreter environments. Programs named **bae_*** have higher priority than those with module-specific names. When pressing one of the keys listed in table 3-2, the **User Language** program with the corresponding name is automatically started (if it is available).

Table 3-2: Key-driven Program Call

Key Designator	Interpreter Environment / Program Name						
	BAE	**SCM**	**GED**	**AR**	**CAM**	**CV**	**CED**
Function Key F1	bae_f1	scm_f1	ged_f1	ar_f1	cam_f1	cv_f1	ced_f1
Function Key F2	bae_f2	scm_f2	ged_f2	ar_f2	cam_f2	cv_f2	ced_f2
Function Key F:	bae_f:	scm_f:	ged_f:	ar_f:	cam_f:	cv_f:	ced_f:
Function Key F12	bae_f12	scm_f12	ged_f12	ar_f12	cam_f12	cv_f12	ced_f12
Digit Key 0	bae_0	scm_0	ged_0	ar_0	cam_0	cv_0	ced_0
Digit Key 1	bae_1	scm_1	ged_1	ar_1	cam_1	cv_1	ced_1
Digit Key :	bae_:	scm_:	ged_:	ar_:	cam_:	cv_:	ced_:
Digit Key 9	bae_9	scm_9	ged_9	ar_9	cam_9	cv_9	ced_9
Standard Key a	bae_a	scm_a	ged_a	ar_a	cam_a	cv_a	ced_a
Standard Key b	bae_b	scm_b	ged_b	ar_b	cam_b	cv_b	ced_b
Standard Key c	bae_c	scm_c	ged_c	ar_c	cam_c	cv_c	ced_c
Standard Key :	bae_:	scm_:	ged_:	ar_:	cam_:	cv_:	ced_:

Event-driven Program Call

The **User Language Interpreter** environments are featuring event-driven **User Language** program calls, where **User Language** programs with predefined names are automatically started at certain events and/or operations such as after BAE module start, after loading and/or before saving a design element, when changing the zoom factor or when selecting a toolbar item. Table 3-3 lists the assignment of predefined **User Language** program names to corresponding interpreter environment events and/or operations. Programs named **bae_*** have higher priority than those with module-specific names. The BAE module start **User Language** program call method is most useful for automatic system parameter setup as well as for key programming and menu assignments (see also below). The element save and load program call methods can be used to save and restore element-specific parameters such as the zoom area, color setup, etc.

Table 3-3: Event-driven Program Call

Event	Interpreter Environment / Program Name						
	BAE	**SCM**	**GED**	**AR**	**CAM**	**CV**	**CED**
On BAE Module Start	**bae_st.ulc**	**SCM_ST**	**GED_ST**	**AR_ST**	**CAM_ST**	**CV_ST**	**CED_ST**
Before BAE Module Exit	**bae_exit.ulc**	**SCM_EXIT**	**GED_EXIT**	**AR_EXIT**	**CAM_EXIT**	**CV_EXIT**	**CED_EXIT**
After Element Load/Close	**bae_load.ulc**	**SCM_LOAD**	**GED_LOAD**	**AR_LOAD**	**CAM_LOAD**	**CV_LOAD**	**CED_LOAD**
After Element Creation	**bae_new.ulc**	**SCM_NEW**	**GED_NEW**	**AR_NEW**	**CAM_NEW**	**CV_NEW**	**CED_NEW**
Before Element Save	**bae_save.ulc**	**SCM_SAVE**	**GED_SAVE**	**AR_SAVE**	**CAM_SAVE**	**CV_SAVE**	**CED_SAVE**
After Element Save	**bae_savd.ulc**	**SCM_SAVD**	**GED_SAVD**	**AR_SAVD**	**CAM_SAVD**	**CV_SAVD**	**CED_SAVD**
On Dialog Activation	**bae_dial.ulc**	**SCM_DIAL**	**GED_DIAL**	**AR_DIAL**	**CAM_DIAL**	**CV_DIAL**	**CED_DIAL**
On Toolbar Selection	**bae_tool.ulc**	**SCM_TOOL**	**GED_TOOL**	**AR_TOOL**	**CAM_TOOL**	**CV_TOOL**	**CED_TOOL**
On Zoom Factor Change	**bae_zoom.ulc**	**SCM_ZOOM**	**GED_ZOOM**	**AR_ZOOM**	**CAM_ZOOM**	**CV_ZOOM**	**CED_ZOOM**
On Mouse Interaction (left mouse button click)	**BAE_MS**	**scm_ms.ulc**	**ged_ms.ulc**	**ar_ms.ulc**	**cam_ms.ulc**	**cv_ms.ulc**	**ced_ms.ulc**
On Frame selection with mouse	**bae_rect.ulc**	**SCM_RECT**	**GED_RECT**	**AR_RECT**	**CAM_RECT**	**CV_RECT**	**CED_RECT**
After Symbol/Part Placement	**BAE_PLC**	**scm_plc.ulc**	**ged_plc.ulc**	**AR_PLC**			**CED_PLC**
After Group Load Operations	**BAE_GRPL**	**scm_grpl.ulc**	**GED_GRPL**				**CED_GRPL**
On Incoming Message ***(BAE HighEnd)***	**BAE_MSG**	**scm_msg.ulc**	**ged_msg.ulc**	**AR_MSG**	**CAM_MSG**	**CV_MSG**	**CED_MSG**

Key Programming and Menu Assignments

Bartels User Language provides system functions for performing key programming and defining menu assignments. It is possible to define key bindings such as key m for activating the **MIRRON User Language** program or key U for activating the Undo BAE menu function. New or existing menus and/or menu entries can be (re-)configured to support special **User Language** program and/or BAE menu function calls. These features provide a powerful tool for configuring the menus of the **AutoEngineer** modules with integrated **User Language Interpreter**. It is a good idea to utilize the **User Language** startup programs for performing automatic key binding and menu setup. For an example on how to provide key bindings and menu assignments see the **UIFSETUP User Language** program distributed with the BAE software; this program is indirectly called from the startup programs listed in table 3-3. Even dynamic changes to the **AutoEngineer** user interface can be supported with special **User Language** programs for performing *online* key and menu programming. See the **KEYPROG User Language** program for implementations of online key programming facilities.

Menu Function Key Bindings

The **KEYPROG** key programming utility available through key 5 can be used in BAE pulldown menu interfaces for assigning BAE menu functions to keys by chosing Key Programming and Set HotKeys, double-clicking a key, clicking the Menu Selection button in the program menu and selecting the desired BAE menu function.

Macro Command Interpreter

A macro command interpreter is built into the Run User Script, function, the **ulsystem User Language** system function, the **bae.ini** key and menu function assignment facilities and the **KEYPROG** online key programming utility to allow for the specification of interaction/command sequences (macros) instead of **User Language** program names. Intercation codes must be separated with the **:** character. The **p** prefix is used to identify **User Language** program names (this prefix should be omitted if the **User Language** program name is the first item in the macro). The # prefix emulates a **bae_callmenu User Language** function call for activating a BAE menu function. Text input is specified through single-quoted strings. **t** awaits user text input. **s** activates a menu selection, which, if followed by **l**, **m** or **r** and a (zero-based) menu index triggers a menu function similar to a **bae_storemenuiact** function call. **m** awaits a mouse click, which, if followed by **l**, **m** or **r**, triggers a mouse click with input coordinates retrieved from the mouse position at the beginning of the interaction sequence if the mouse key in the macro isn't followed by coordinate specifications.

With macro specifications, it is possible to assign submenu functions such as Symbol/Label Query (macro **scmpart:s5:m:t:mr**) to keys. It is also possible to define macros for frequently required interaction sequences such as the **Schematic Editor** macro **#500:m:mr:sl3:'4':'0':mr:sl0** for creating a 4mm horizontal graphic line from the current mouse position to the right (SCM symbol graphic pin connection).

Customized Parameter Settings and Menu Configurations

Customized parameter settings, key definitions and menu assignments can be stored to the **bae.ini** file in the BAE programs directory. The definitions from this file are loaded once upon BAE startup and can subsequently be accessed with the **varget User Language** system function.

The BAE **User Language** programs are designed to evaluate relevant definitions from **bae.ini**. Changes to **bae.ini** are activated by simply restarting the affected BAE program module, thus eliminating the need to recompile any of the affected **User Language** programs. I.e., with **bae.ini**, user-specific parameter settings can easily be transferred between different BAE versions and/or installations.

The **bae.ini** file provides specific sections for different BAE program modules. Each BAE program module only loads relevant sections from **bae.ini**. The definitions from the **std** section are relevant for all modules.

bae.ini allows for generic parameter value assignments. The **key** and **fkey** commands are used for standard and function key assignments. The **addmenu**, **addmenuitem**, **addsmenuitem** and **addioitem** kewords allow for menu extensions. Please note that menu items can only be appended to main menus and import/export menus. Menu item insertion is prohibited to preserve online help topic assignment.

A **bae.ini** file with inline documentation for command syntax explanation is supplied with the BAE software. The original version of this file activates the parameter settings from the supplied BAE **User Language** programs. I.e., these (default) settings are also activated if the **bae.ini** is not available in the BAE programs directory.

3.3.3 Error Handling

User Language Interpreter errors might occur whilst executing a **User Language** program call. These errors are reported to the interpreter environment, and a corresponding error message is displayed in the status line of the interpreter environment. The following listings contain all error message definitions of the **User Language Interpreter**.

Fatal Internal Errors

The following internal **User Language Interpreter** errors either refer to memory management problems or indicate **User Language Interpreter** implementation gaps; these errors are fatal, i.e., they force an abortion of the current interpreter environment:

```
ERROR : Out of memory!
ERROR : Unspecified User Language interpreter error!
ERROR : INTERNAL ERROR -- PLEASE REPORT!
```

Program Loader Errors

The following **User Language Interpreter** errors refer to program load failures or compatibility problems:

```
ERROR : Program 'n' already loaded (recursive call)!
ERROR : Program 'n' not found!
ERROR : Incompatible User Language program version!
ERROR : Incompatible index/function reference(s)!
```

Program Runtime Errors

The following **User Language Interpreter** errors refer to problems whilst executing a **User Language** program, i.e., they might occur at program runtime; these error messages include a program counter (**PCl**) indicating the corresponding machine program line which caused the errors (this information can be used for finding the location of the corresponding source code line if a listing file has been produced by the **User Language Compiler**):

```
(PCl) ERROR : Stack underflow (program structure damaged)!
(PCl) ERROR : Stack overflow!
(PCl) ERROR : Division by zero is attempted!
(PCl) ERROR : Function call error!
(PCl) ERROR : System function not available here!
(PCl) ERROR : System function not implemented!
(PCl) ERROR : User function not found!
(PCl) ERROR : Referenced function is of wrong type!
(PCl) ERROR : Invalid parameter for referenced function!
(PCl) ERROR : Memory protection fault on array access!
(PCl) ERROR : Array index out of range!
(PCl) ERROR : File access error!
(PCl) ERROR : File read error!
(PCl) ERROR : File write error!
```

Database Access Errors

The following **User Language Interpreter** errors refer to problems accessing the **Bartels AutoEngineer** design database (DDB); such errors might indicate an erroneous **Bartels AutoEngineer** software installation, if they occur whilst loading a **User Language** program; a database access error occurring whilst program runtime usually is caused by an implementation error in the executed **User Language** program:

```
ERROR : Cannot create database file 'n'!
ERROR : Read/write error whilst accessing file 'n'!
ERROR : Too many open files!
ERROR : File 'n' is not a database/DDB file!
ERROR : File structure is damaged in file 'n'!
ERROR : File structure is erroneous in file 'n'!
ERROR : Function not available for old format!
ERROR : Database limit exceeded!
ERROR : File 'n' is not compatible with program version!
ERROR : Element 'n' not found!
ERROR : Element 'n' exists already!
ERROR : File 'n' not found!
ERROR : Record end reached!
ERROR : Unspecified database error!
```

Chapter 4
BAE User Language Programs

This chapter lists all **Bartels AutoEngineer User Language** include files and **User Language** programs with short descriptions, and provides information on how to make the programs available to the BAE software. The program listings are grouped by fields of application and corresponding interpreter environments.

Contents

4.1 User Language Include Files

This section provides a list of the **User Language** include files distributed with the **Bartels AutoEngineer** software. The include provide frequently used functions and definitions and are extensively referenced from the **User Language** programs shipped with the BAE software.

4.1.1 Standard Include Files

std.ulh (STD) -- Standard Include

The definitions and declarations from the **std.ulh** include file are compatible with all **User Language Interpreter** environments of the **Bartels AutoEngineer** (i.e., **Schematic Editor**, **Layout Editor**, **Autorouter**, **CAM Processor**, **CAM View** and **Chip Editor**, respectively). **std.ulh** provides general utilities and definitions for BAE software configuration query, data conversion, display management, error handling, menu and/or user interaction, internationalization, workarea manipulation, etc.

baeparam.ulh (STD) -- BAE Parameter Access

The definitions and declarations from include file **baeparam.ulh** are compatible with all **User Language Interpreter** environments of the **Bartels AutoEngineer**. **baeparam.ulh** provides definitions and functions for accessing BAE parameter settings. **baeparam.ulh** includes the **std.ulh** source code file (see above).

pop.ulh (STD) -- Popup Utilities

The definitions and declarations from the **pop.ulh** include file are compatible with all **User Language Interpreter** environments of the **Bartels AutoEngineer**. **pop.ulh** provides advanced popup utilities and definitions for BAE software configuration query, menu-driven file and element selection, directory list generation, popup message handling, menu and/or user interaction, etc. **pop.ulh** includes the **std.ulh** source code file (see above).

popdraw.ulh (STD) -- Popup Drawing Functions

The definitions and declarations from the **popdraw.ulh** include file are compatible with all **User Language Interpreter** environments of the **Bartels AutoEngineer**. **popdraw.ulh** provides general utilities and definitions for performing icon and button display and graphical output to popup menus and for maintaining toolbars. **popdraw.ulh** includes the **std.ulh** source code file (see above).

mnu.ulh (STD) -- Menu Functions

The definitions and declarations from the **mnu.ulh** include file are compatible with all **User Language Interpreter** environments of the **Bartels AutoEngineer**. **mnu.ulh** provides advanced popup and menu utilities such as text display menu, string query edit mask, color setup menu, BAE product info popup, menu and/or user interaction, etc. **mnu.ulh** includes the **pop.ulh** source code file (see above).

sql.ulh (STD) -- SQL Utilities

The definitions and declarations from the **sql.ulh** include file are compatible with all **User Language Interpreter** environments of the **Bartels AutoEngineer**. **sql.ulh** provides a series of useful SQL database management utilities. **sql.ulh** includes the **pop.ulh** source code file (see above).

xml.ulh (STD) -- XML Utilities

The definitions and declarations from the **xml.ulh** include file are compatible with all **User Language Interpreter** environments of the **Bartels AutoEngineer**. **xml.ulh** provides a series of useful XML file import and export functions. **xml.ulh** includes the **std.ulh** source code file (see above).

4.1.2 Schematic Include Files

scm.ulh (SCM) -- SCM/Schematic Editor Utilities

The definitions and declarations from the **scm.ulh** include file are compatible with the **Schematic Editor User Language Interpreter** environment of the **Bartels AutoEngineer**. **scm.ulh** provides SCM utilities for data conversion, SCM element copy, SCM element rule assignments, etc. **scm.ulh** includes the **std.ulh** source code file (see above).

4.1.3 Layout Include File

lay.ulh (LAY) -- Layout Utilities

The definitions and declarations from the **lay.ulh** include file are compatible with the **Layout Editor**, the **Autorouter** and the **CAM Processor User Language Interpreter** environments of the **Bartels AutoEngineer**. **lay.ulh** provides utilities for data conversion, geometric calculation, net list data evaluation, layout element copy, layer name query, etc. **lay.ulh** also provides utilities for accessing and applying **Bartels Rule System** features in the BAE layout system. **lay.ulh** includes the **std.ulh** source code file (see above).

4.1.4 IC Design Include Files

icd.ulh (ICD) -- IC Design Utilities

The definitions and declarations from the **icd.ulh** include file are compatible with the **IC Design Chip Editor User Language Interpreter** environment of the **Bartels AutoEngineer**. **icd.ulh** provides utilities for analytical geometry, **IC Design** element copy, layer name query, etc. **icd.ulh** also provides utilities for accessing and applying the **Bartels Rule System** features in the BAE **IC Design** system. **icd.ulh** includes the **std.ulh** source code file (see above).

4.2 User Language Programs

This section lists all **User Language** programs distributed with the **Bartels AutoEngineer**. The programs are grouped according to the fields of application and/or the corresponding interpreter environments. The source files for all these programs are supplied with the **Bartels AutoEngineer** software.

4.2.1 Standard Programs

The following **User Language** programs are compatible to all **Bartels AutoEngineer** interpreter environments (i.e., they can be called from the **Schematic Editor**, as well as from the **Layout Editor**, the **Autorouter**, the **CAM Processor**, the **CAM View** module, and the **Chip Editor**).

ARC (STD) -- Draw Arc/Circle

The **arc.ulc** **User Language** program designates the currently active BAE menu function and performs a submenu interaction for the quick drawing of an arc or circle. If called with no BAE menu function active, a dialog box for selecting the mode of operation is displayed. This program must be configured for implicit hotkey program call (e.g., C or C).

BAE_DIAL (STD) -- BAE Dialog Box Action

The **bae_dial.ulc** **User Language** program is automatically activated when selecting a dialog box item with registered action code. The dialog box activation data is transfered from the interaction queue to global variables.

BAE_EXIT (STD) -- BAE Program Exit Action

The **bae_exit.ulc** **User Language** program is automatically activated before exiting the current BAE module. It frees any active lock for the currently loaded element.

BAE_LOAD (STD) -- BAE Load/Close Action

The **bae_load.ulc** **User Language** program is automatically activated after loading, creating or closing an element. In pulldown menu interfaces, **bae_load.ulc** displays the file and element name of the loaded element and the mouse operation mode currently selected through **MSMODE**. The display appears in the info/status field of the pulldown menu interface. On element close, **bae_load.ulc** clears the file and element name display. **bae_load.ulc** also restores any element-specific toolbar display/attachment preference previously stored with **BAE_SAVE**.

BAE_NEW (STD) -- BAE New Element Action

The **bae_new.ulc** **User Language** program is automatically activated after loading, creating or closing an element. **bae_new.ulc** sets default parameters for the new element

BAE_RECT (STD) -- BAE Mouse Rectangle Frame Action

The **bae_rect.ulc** **User Language** program is automatically activated during program idle times when pressing the left mouse button and drawing a rectangular frame of more than 10 pixels width in the work area whithout releasing the mouse button. According to the mouse operation mode currently selected through **MSMODE**, **bae_rect.ulc** applies group function to the elements inside the frame rectangle.

BAE_SAVD (STD) -- BAE Save Done Action

The **bae_savd.ulc** **User Language** program is automatically activated after saving an element. **bae_savd.ulc** supports the execution of a user-specific **User Language** program.

BAE_SAVE (STD) -- BAE Save Action

The **bae_save.ulc** **User Language** program is automatically activated before saving an element. **bae_save.ulc** saves the toolbar design view windows and the toolbar display/attachment mode with the current element.

BAE_ST (STD) -- BAE Startup

The **bae_st.ulc** **User Language** program is automatically activated when starting a BAE program module (**Schematic Editor**, **Layout Editor**, **Neural Router**, **CAM Processor**, **CAM View** and/or **Chip Editor**). **bae_st.ulc** starts the **UIFSETUP** **User Language** program for activating predefined menu and key assignments in the current interpreter environment. According to the currently active interpreter environment **bae_st.ulc** also calls one of the programs **SCMSETUP**, **GEDSETUP**, **ARSETUP**, **CAMSETUP**, **CVSETUP** and **CEDSETUP** for setting module-specific standard parameters such as input/display grids, grid/angle lock, coordinate display mode, pick preference layer, mincon function, etc. Each of these programs can be easily customized for optimizing user-specific and/or project-specific BAE environments.

BAE_TOOL (STD) -- BAE Toolbar Action

The **bae_tool.ulc** **User Language** program is automatically called when activating a toolbar item. The action to be activated depends on the input string encountered by the **bae_tool.ulc** program. On integer input a menu call request is assumed, otherwise a **User Language** program call request is assumed. Both input string types also allow for the specification of a blank-separated parameter string to be passed to the interaction queue of the subsequently called menu function or **User Language** program.

BAE_ZOOM (STD) -- BAE Zoom Action

The **bae_zoom.ulc** **User Language** program is automatically activated when changing the current zoom window. **bae_zoom.ulc** updates the display overview window in the toolbar if the zoom factor has been changed.

BAEMAN (STD) -- BAE Windows Online Manual (Windows)

The **baeman.ulc** **User Language** program launches the default **Windows** web browser to display the **Bartels AutoEngineer** User Manual.

BITMAPIN (STD) -- Import Bitmapdaten

The **bitmapin.ulc** **User Language** program can be used to import different bitmap data formats. Bitmap data imported to the **Schematic Editor** is converted into graphic areas. Bitmap data imported into the layout is converted to documentary areas on a selectable layer. Imported bitmap data is automatically group-selected to allow for easy re-positioning, scaling, area type conversion, etc.

CLOGDEFS (STD) -- Check Logical Library Definitions

The **clogdefs.ulc** **User Language** program examines the SCM symbol names of selectable library files of the current directory and checks whether the corresponding logical library entries exist in a selectable layout library. The resulting report is displayed in a popup menu with file output option.

CMDCALL (STD) -- Execute Command Call Sequence

The **cmdcall.ulc** **User Language** program queries the input of a command sequence and executes this sequence. BAE menu function command sequences are documented in the reference manuals (see brgar.htm, brgcam.htm, brgcv.htm, brgged.htm and brgscm.htm). This is basically the same behaviour as Run User Script with the difference that the BAE window rather than a separate dialog window has the input focus. This enables remotely controlled BAE operations with tools such as **StrokeIT** for mouse gesture program control. The **cmdcall.ulc** program is pre-configured for hotkey `Shift-Ctrl-R`, thus a remotely controlled function call can be triggered through `Shift-Ctrl-R` followed by the command sequence and the `ENTER` key.

COPYELEM (STD) -- Copy DDB File Elements

The **copyelem.ulc** **User Language** program copies menu-selectable elements from one DDB file to another, thus providing facilities similar to the **COPYDDB** Utility program. The popup menu for selecting multiple source file elements of the chosen DDB class allows for wildcard element name specifications such as, e.g., `74ls*` for SCM symbols or `dil*` for layout part symbols. The copy/merge mode (Copy All for overwriting existing destination file elements or No Replacements for keeping existing destination file elements) must be selected after specifying an existing destination file containing elements of the chosen DDB class. With SCM symbols selected, there is also an option for copying symbol-specific logical library definitions.

DBCOPY (STD) -- SQL Database Copy

The **dbcopy.ulc** **User Language** program copies SQL table structures and database entries of selectable SQL database files.

DBREPORT (STD) -- SQL Database Report

The **dbreport.ulc** **User Language** program provides utilities for displaying SQL table structures and database entries of selectable SQL database files.

DELCOLOR (STD) -- Delete Selectable Color Table

The **delcolor.ulc** **User Language** program executes the Delete Element BAE file command with class color table.

DELDVINF (STD) -- Delete Design View Information

The **deldvinf.ulc** **User Language** program deletes all design view information from the currently loaded elements database file.

DESKCALC (STD) -- Desk Calculator

The **deskcalc.ulc** **User Language** program activates a popup menu with a desk calculator providing basic arithmetic operations and trigonometric functions.

DIR (STD) -- List Current Directory Files

The **dir.ulc** **User Language** program lists the file and subdirectory names of the current directory to the screen.

DISPUTIL (STD) -- Display Utilities

The **disputil.ulc** **User Language** program provides a common interface to frequently used module-specific display utility functions.

DISTANCE (STD) -- Distance Query

The **distance.ulc** **User Language** program displays absolute, horizontal (X) and vertical (Y) distances and the angle between two interactively selectable points. The length units for the distance display are initially retrieved from the current coordinate display mode and can be changed on request.

DONE (STD) -- Finish Input Interaction

The **done.ulc** **User Language** program terminates the input loop of any polygon/connection/path input function. This program must be configured for implicit hotkey program call (e.g., Enter).

FAVORITE (STD) -- Favourites Menu Management

The **favorite.ulc** **User Language** program provides functions for configuring and activating a user-defined favorites menu. This program is intended for toolbar button call.

FILEUTIL (STD) -- File Utilities

The **fileutil.ulc** **User Language** program provides a series of file utility functions such as list directory, copy/delete/list file, search and display DDB elements, display SQL database contents, etc.

FILEVIEW (STD) -- List File contents

The **fileview.ulc** **User Language** program lists the contents of a freely selectable ASCII file to the screen.

FINDELEM (STD) -- Find and Browse DDB Elements

The **findelem.ulc** **User Language** program scans the hard disk (i.e., selectable directories and subdirectories) for DDB elements of a selectable database class. The names of the elements to be searched can be specified with wildcards. The elements found by the element scanner can be sorted by file names or element names. A browser for displaying and/or loading the selected elements is activated for database classes compatible with the current interpreter environment.

GRTOGGLE (STD) -- Toggle Input Grid

The **grtoggle.ulc** **User Language** program toggles the currently selected input grid mode between gridless and grid locked. The angle lock flag is restored when switching to grid locked mode. This program is intended for implicit hotkey program call (e.g., g or G).

HELP (STD) -- Online Help System

The **help.ulc** **User Language** program provides BAE online help.

HISTORY (STD) -- Element History Call

The **history.ulc****User Language** program provides a file element history with menu selection option for load operations. This program is intended for toolbar button call.

HLPKEYS (STD) -- Online Help - Key Bindings Display

The **hlpkeys.ulc** **User Language** program displays the currently active key bindings. The listing is displayed in a popup menu with file output option.

HLPPROD (STD) -- Online Help - BAE Product Information

The **hlpprod.ulc** **User Language** program provides a popup with **Bartels AutoEngineer** product information such as BAE software configuration, activated user interface, currently active program module, version number, etc.

INFO (STD) -- Info

The **info.ulc** **User Language** program provides a common interface to frequently used environment-specific report functions. **INFO** checks the current environment and starts a specific **User Language** report program or help utility (i.e. **SCMPCR** in the **Schematic Editor**, **LAYPCR** in the Layout system, **HLPPROD** if no element loaded, etc.).

INIEDIT (STD) -- bae.ini Editor

The **iniedit.ulc** **User Language** is used for interactively editing common BAE system parameters in the `bae.ini` file from the BAE programs directory. The old `bae.ini` contents is backed up to `bae.bak`.

KEYPROG (STD) -- User Language Program Call and Key Programming Utility

The **keyprog.ulc** **User Language** program provides features for menu-controlled **User Language** program call, online key programming, menu assignment management and SQL-based **User Language** help text database management.

LARGER (STD) -- Increase Pick Element Size/Width

The **larger.ulc** **User Language** program designates the currently active BAE menu function and performs a submenu interaction for increasing the size or width of the currently picked element (if an object is picked and size/width increase is permitted). This program must be configured for implicit hotkey program call (e.g., b).

LFTOCRLF (STD) -- LF to CRLF Converter

The **lftocrlf.ulc** **User Language** program converts all line feed characters in a selectable ASCII file to carriage return and line feed control character sequences.

LIBCONTS (STD) -- List Library Contents

The **libconts.ulc** **User Language** program lists the names of all BAE library parts contained in the current directory's DDB and DEF files. Output is directed to popup menus with file output options, and can be used for library documentation purposes.

LIBCRREF (STD) -- Library Element Cross Reference

libcrref.ulc is a library management utility program which generates a cross reference for the library elements contained in the DDB files of the current directory. The library cross reference is displayed in a popup menu with file output option; it lists the selected library elements and denotes the DDB file(s) to contain these elements.

LISTDDB (STD) -- List DDB File Elements

The **listddb.ulc** **User Language** program produces a list of the database elements of a selectable DDB and displays this list in a popup menu with file output option.

LOADELEM (STD) -- Load Element with Check

The **loadelem.ulc** **User Language** program loads a menu-selectable BAE (project) element. Extended consistency check is carried out during load involving options for listing missing library definitions, unplaced layout parts, wrong package types, etc.

LOADFONT (STD) -- Load Character Font

The **loadfont.ulc** **User Language** program loads a menu-selectable character font and assigns it to the currently loaded SCM element.

LOADNEXT (STD) -- Load Next Element with Check

The **loadnext.ulc** **User Language** program determines both name and class of the currently loaded element and tries to load that element of the same class, which is stored after the currently loaded element in the corresponding DDB file (forward DDB file browse). A menu-driven load operation is carried out if no next element can be determined. Extended consistency checks are carried out during load operations with options for listing missing library definitions, unplaced layout parts, wrong package types, etc.

LOADPREV (STD) -- Load Previous Element with Check

The **loadprev.ulc** **User Language** program determines both name and class of the currently loaded element and tries to load that element of the same class, which is stored before the currently loaded element in the corresponding DDB file (backward DDB file browse). A menu-driven load operation is carried out if no previous element can be determined. Extended consistency checks are carried out during load operations with options for listing missing library definitions, unplaced layout parts, wrong package types, etc.

LROTATE (STD) -- Left Rotate Pick Element, Change Angle Direction

The **lrotate.ulc** **User Language** program designates the currently active BAE menu function and performs a submenu interaction for 90 degree left-rotate on the currently processed object (if an object is picked and if rotation is permitted). This program must be configured for implicit hotkey program call (e.g., l or L).

The L/R Rotation Angle function from the **GEDPART** **User Language** program can be used in the **Layout Editor** to specify an arbitrary part, pin, polygon, text and/or group rotation angle to be applied by **LROTATE**.

MACRO (STD) -- Macro Command Management

The **macro.ulc****User Language** program provides functions for maintaining and executing command macro sequences.

MIRROFF (STD) -- Mirror Off Pick Element

The **mirroff.ulc** **User Language** program designates the currently active BAE menu function and performs a submenu interaction for resetting the mirror mode of the currently processed object (if one is picked and if mirroring is permitted). This program must be configured for implicit hotkey program call (e.g., n or N).

MIRRON (STD) -- Mirror On Pick Element, Change Edit Direction

The **mirron.ulc** **User Language** program designates the currently active BAE menu function and performs a submenu interaction for mirroring the currently processed object (if an object is picked and if mirroring is permitted). This program must be configured for implicit hotkey program call (e.g., m or M).

MMB (STD) -- Middle Mouse Button Interaction

The **mmb.ulc** **User Language** programs performs a middle mouse button interaction for activating the online Display menu. **MMB** is intended to be assigned to a common key such as ␣ (space) to allow for the activation of the online BAE display menu by simply pressing a certain key.

MSMODE (STD) -- Set Mouse Context Operation Mode

The **msmode.ulc** **User Language** program can be used to set a mouse context operation mode (No Operation, Context Functions, Delete, Move, Select) for designating object-specific functions to be automatically activated when clicking objects with the left mouse button.

OSSHELL (STD) -- Run Operating System Shell

The **osshell.ulc** **User Language** program provides an interface to the operating system shell. The system **User Language** function is utilized for executing operating system commands. Kindly note the restrictions and limitations regarding the use of the **system** **User Language** function before using this program!

RENAMEEL (STD) -- Rename DDB File Elements

The **renameel.ulc** **User Language** program provides DDB file element renaming functions. Available element classes depend on the calling environment.

RROTATE (STD) -- Right Rotate Pick Element, Draw Rectangle

The **rrotate.ulc** **User Language** program designates the currently active BAE menu function and performs a submenu interaction for 90 degree right-rotate on the currently processed object (if an object is picked and if rotation is permitted). This program must be configured for implicit hotkey program call (e.g., r or R).

The L/R Rotation Angle function from the **GEDPART** **User Language** program can be used in the **Layout Editor** to specify an arbitrary part, pin, polygon, text and/or group rotation angle to be applied by **RROTATE**.

SAVEELAS (STD) -- Save Element As

The **saveelas.ulc** **User Language** program stores the currently loaded element with a freely selectable file and element name. Overwrite verification is automatically activated on existing destination elements.

SIZE (STD) -- Resize Pick Element

The **size.ulc** **User Language** program designates the currently active BAE menu function and performs a submenu interaction for resizing the currently processed object (if an object is picked and if resizing is permitted). This program must be configured for implicit hotkey program call (e.g., s or S). The new object size is specified through an option menu with a list of predefined size values and a function for specific size value input.

SMALLER (STD) -- Reduce Pick Element Size/Width

The **smaller.ulc** **User Language** program designates the currently active BAE menu function and performs a submenu interaction for reducing the size or width of the currently picked element (if an object is picked and size/width reduction is permitted). This program must be configured for implicit hotkey program call (e.g., B).

STEPDOWN (STD) -- Step One Layer Down

The **stepdown.ulc** **User Language** program designates the currently active BAE menu function and performs a **Layout Editor** submenu interaction for changing to the next lower layer if an object is picked and if a layer change is permitted. This program must be configured for implicit hotkey program call (e.g.,], Ctrl/mousewheel down).

STEPUP (STD) -- Step One Layer Up

The **stepup.ulc** **User Language** program designates the currently active BAE menu function and performs a **Layout Editor** submenu interaction for changing to the next upper layer if an object is picked and if a layer change is permitted. This program must be configured for implicit hotkey program call (e.g., [Ctrl]/mousewheel up).

TBATTACH (STD) -- Attach Toolbar

The **tbattach.ulc** **User Language** program attaches the toolbar build by the **TOOLBAR** **User Language** program to either of the four edges of the workspace (left, right, top or bottom). **TBATTACH** also provides an option for deactivating the toolbar display.

TOOLBAR (STD) -- Toolbar

The **toolbar.ulc** **User Language** program displays a toolbar providing advanced and frequently used functions and features such as display function short-cuts, file and element processing, design view management, info and report function access, mouse context operation mode setting, etc.

UIFDUMP (STD) -- Menu Assignments and Key Bindings Dump

The **uifdump.ulc** **User Language** program generates lists the menu assignments and key bindings of the currently active BAE program module. The listing is displayed in a popup menu with file output option.

UIFRESET (STD) -- Menu Assignments and Key Bindings Reset

The **uifreset.ulc** **User Language** program resets the complete menu assignments and key bindings of the currently active BAE program module.

UIFSETUP (STD) -- Menu Assignments and Key Bindings Setup

The **uifsetup.ulc** **User Language** program performs automatic setup of pre-defined menu assignments and key bindings in the currently active **User Language Interpreter** environment. This program is intended for implicit BAE program module startup call (i.e., program call via **User Language** program **BAE_ST**).

ZOOMIN (STD) -- Zoom In

The **zoomin.ulc** **User Language** program performs a zoom in/zoom larger display command.

ZOOMOUT (STD) -- Zoom Out

The **zoomout.ulc** **User Language** program executes the Zoom Out BAE display command.

4.2.2 Schematic Editor Programs

The following **User Language** programs are compatible to the **Schematic Editor** interpreter environment (i.e., they can be called from the **Schematic Editor**).

ATTRSET (SCM) -- SCM Symbol Attribute Assignment

The **attrset.ulc** **User Language** programs activates a dialog with advanced functions for assigning attribute values to mouse-selectable SCM symbols.

CHKSMAC (SCM) -- List undefined SCM Macro References

The **chksmac.ulc** **User Language** program lists the names of all undefined macros (library elements) referenced from the currently loaded SCM element to a popup menu with file output option.

DEF2CSV (SCM) -- Create Symbol Selection Database from .def Files

The **def2csv.ulc** **User Language** scans a selectable folder for logical definition files and creates `.csv` and `.map` files suitable for the creation of a symbol selection database with the **symattdb.ulc** and **symmapdb.ulc** **User Language** programs. The library name for each symbol is derived from the `.def` file name. I.e., **def2csv.ulc** should only be used for definition file names matching the corresponding SCM symbol library file name.

FINDSPRT (SCM) -- Search SCM Part

The **findsprt.ulc** **User Language** program searches for a name-selectable part on the currently loaded SCM sheet. The name selection menu provides all parts of all SCM sheets of the current DDB file including SCM sheet name, SCM part name and physical part name assigned by the **Packager**. Additional options are provided for direct SCM or layout part name search and for locating parts with selectable attribute values. Part search on alternate SCM sheets is done with verification only. **FINDSPRT** automatically zooms to the searched part.

LOGLEDIT (SCM) -- Loglib Editor Functions

The **logledit.ulc** **User Language** program can be used to load, edit and compile logical library part definitions in the **Schematic Editor**. On SCM symbol level, either the logical library definition of the currently loaded SCM symbol is loaded or a logical library definition template with all symbol pins is created. On SCM sheet level, the logical library definition of a mouse-selectable symbol can be created/loaded. For other element classes, the logical library definition can be specified and/or selected through an SCM symbol name query. The corresponding logical library definition can be selected from the currently processed design or library file, the default SCM library or any other DDB file. Once the logical library definition is loaded, it can be edited. Finally, the OK button can be used to compile the logical library definition. See the description of the **LOGLIB** utility program for information on the logical library definition format.

NETCONV (SCM) -- Logical Netlist Conversion

The **netconv.ulc** **User Language** program is used for transferring logical (i.e. unpacked) net list data from different ASCII formats (BAE, ALGOREX, Applicon, CADNETIX, CALAY, EEDESIGNER, Marconi ED, Mentor, MULTIWIRE, OrCAD, PCAD, RINF, SCICARDS, TANGO, VECTRON, VUTRAX, WIRELIST) to internal **Bartels AutoEngineer** format ready for processing with the BAE **Packager**. **NETCONV** reads the net list data from the selected net list file `<project>.net` and stores the net list to DDB file `<project>.ddb` using the element name specified in the net list file (default `netlist`). The logical net list generated by **NETCONV** can be translated to a physical net list using the **Packager**. In the subsequent layout process, it is possible to perform pin/gate swaps according to the logical library swap commands transferred by the **Packager**. For consistency checks, **NETCONV** requires access to the pin lists of the SCM symbols referenced from the net list. SCM symbol info is loaded either from the destination project file (`<project>.ddb`) or from a predefined list of SCM symbol library files. Net attributes for **Autorouter** control (`ROUTWIDTH`, `POWWIDTH`, `MINDIST`, `PRIORITY`) are transferred by connecting synthetically generated net attribute parts; required net attribute name references such as net attribute symbol names, pin names and net attribute names are predefined in the **NETCONV** source code. Net list transfer process messages, warnings and error messages are listed both to the screen and to a logfile named `bae.log`. After successfully transferring the net list, **NETCONV** recommends to run the **Packager**.

PERRLIST (SCM) -- Packager Error List Display

The **perrlist.ulc** **User Language** program displays a **Packager** error list in a modeless dialog with options for zooming to selectable error symbols.

PLANSORT (SCM) -- SCM Plan Sort

The **plansort.ulc** **User Language** program performs an automatic SCM sheet name numbering for selectable DDB files of the current directory.

SADDNAME (SCM) -- SCM Symbol Add Name

The **saddname.ulc** **User Language** program performs semi-automatic placement of text corresponding to the macro name of the currently loaded SCM symbol.

SAUTONAM (SCM) -- Automatic Schematic Symbol Rename Utilities

The **sautonam.ulc** **User Language** program activates a menu with a series of functions for automatically renaming symbols of the currently loaded SCM sheet. Symbol numbering is accomplished from the left top to the right bottom of the SCM sheet area. Symbol/part renumbering can be applied either on the currently loaded SCM sheet (option Current Sheet) only or on all sheets of the currently loaded SCM sheet (All Sheets).

SBROWSE (SCM) -- Schematic Symbol Browser

The **sbrowse.ulc** **User Language** program allows for the selection of SCM symbols to be loaded and/or placed. A graphical display of the currently selected library symbol is displayed in a popup menu.

SCM_GRPL (SCM) -- SCM Group Load Action

The **scm_grpl.ulc** **User Language** program is automatically activated after group load operations. **SCM_GRPL** updates symbol variant attributes.

SCM_MS (SCM) -- SCM Mouse Action

The **scm_ms.ulc** **User Language** program is automatically activated when pressing the left mouse button in the workarea whilst the **Schematic Editor** is idle. **SCM_MS** provides a context-sensitive function menu for the object at the current mouse position. The Load Element, New Element and Mouse Mode functions are provided if no element is loaded.

SCM_MSG (SCM/HighEnd) -- SCM Message System Action

The **scm_msg.ulc** **User Language** program provides functions to be performed when receiving messages from other **BAE HighEnd** modules. The actions to be triggered (global variable settings, **User Language** program calls) and the objects to be processed are designated by the incoming message string.

SCM_PLC (SCM) -- SCM Symbol Placement Action

The **scm_plc.ulc** **User Language** program is automatically activated after a symbol is placed. **SCM_PLC** updates the symbol placement pool.

SCMBOUND (SCM) -- Set SCM Workspace/Element Boundary

The **scmbound.ulc** **User Language** program provides functions for automatically enlarging and/or shrinking the element boundaries of the currently loaded SCM element.

SCMCON (SCM) -- SCM Connection Functions

The **scmcon.ulc** **User Language** program provides a menu with a series of advanced connection processing functions such as bus display mode setting, bus query and highlight nets.

SCMCRREF (SCM) -- SCM Plan Part/Label Cross Reference

The **scmcrref.ulc** **User Language** program produces a part and label cross reference listing for a selectable SCM sheet. The cross reference listing is displayed in a popup menu with file output option.

SCMDISP (SCM) -- SCM Display Functions

The **scmdisp.ulc** **User Language** program provides a series of advanced **Schematic Editor** display management functions.

SCMDRAW (SCM) -- SCM 2D Drawing Functions

The **scmdraw.ulc** **User Language** program provides a series of advanced SCM drawing functions such as producing circles, rectangles and/or arrows, performing distance measuring and generating rulers.

SCMDUMP (SCM) -- SCM ASCII Dump

The **scmdump.ulc** **User Language** program generates a dump of the currently loaded SCM element. The output is directed to an ASCII file.

SCMDXFDI (SCM) -- SCM AutoCAD/DXF Input

The **scmdxfdi.ulc** **User Language** program loads AUTOCAD/DXF drawing data onto the currently loaded SCM element. Input length units, input offsets and layer transformations can be adjusted as required.

SCMDXFDO (SCM) -- SCM AutoCAD DXF Output

The **scmdxfdo.ulc** **User Language** program produces an AUTOCAD/DXF output of the currently loaded SCM element.

SCMEDFDI (SCM) -- SCM EDIF Data Import

The **scmedfdi.ulc** **User Language** program imports EDIF SCM data consisting of net lists, SCM symbols and schematic drawings to the BAE **Schematic Editor**. Logical part library definitions required for the subsequent **Packager** process are automatically generated.

SCMEPS (SCM) -- SCM EPS/PDF Output

The **scmeps.ulc** **User Language** program produces either Adobe Portable Document Format (PDF) or Encapsulated PostScript (EPS) output of the currently loaded SCM element, featuring different scaling modes for output and optional clipping window selection. **SCMEPS** performs plot of different graphic objects (standard text, comment text, documentary areas, documentary lines, dotted lines, contact areas, connections, busses) from different DDB hierarchy levels (sheet, symbol, label, marker). Output options such as fill mode, color assignment and/or gray scale, hatching and dashed lines can be predefined for each object type to be plotted. Object-specific plot options are controlled with a special **SCMEPS** source code variable which is intended for customization. **SCMEPS** also considers any plot visibility rules assigned to plot elements with **SCMRULE**.

SCMGROUP (SCM) -- SCM Group Functions

The **scmgroup.ulc** **User Language** program provides a menu with advanced SCM group functions such as automatic selection and/or de-selection of all objects of a selectable type or with specific attributes, loading groups from different database hierarchy levels, changing the size of group-selected texts and global attribute value assignments to group-selected parts. A special function for selecting layout part sets from SCM symbol groups is also provided.

SCMIO (SCM) -- SCM Data I/O Functions

The **scmio.ulc** **User Language** program provides a menu with a series of advanced **Schematic Editor** data input/output functions. User-specific import/export functions can easily be added through `addioitem` commands in the `bae.ini` file of the BAE programs directory.

SCMMACL (SCM) -- Schematic Macro Load

The **scmmacl.ulc** **User Language** program loads the library macro of a mouse-selectable symbol, marker or pin from the currently loaded SCM element to the **Schematic Editor**.

SCMPART (SCM) -- SCM Symbol/Label Functions

The **scmpart.ulc** **User Language** program activates a menu with a series of advanced symbol and label functions such as mirror and/or rotate selectable symbols and/or labels, part renaming, symbol/label query, part search, part list generation, menu-driven attribute assignment, etc.

SCMPCR (SCM) -- SCM Report

The **scmpcr.ulc** **User Language** program provides detailed information about the currently loaded SCM element such as DDB file name, element name, block name of hierarchical sheet, element origin coordinates, element boundary coordinates, element size, reference listing (symbols/labels on SCM sheet, markers on symbol/label). The report output is displayed in a popup menu with file output option.

SCMPEDIT (SCM) -- SCM Position Pick/Element Edit

The **scmpedit.ulc** **User Language** provides functions to snap to the coordinates of elements placed at the current mouse position. This program must be configured for implicit hotkey program call (e.g., p or P).

SCMPLOT (SCM) -- SCM Plan Plot

The **scmplot.ulc** **User Language** program plots all SCM plans of a selectable DDB file using the current plot parameter settings (output device, plot scale, etc.). Available output formats are PostScript and HP-Laser (PCL).

SCMPOLY (SCM) -- SCM Polygon Functions

The **scmpoly.ulc** **User Language** program provides a menu with a series of advanced polygon processing functions such as defining and/or changing polygon types, polygon mirroring, copying group-selected polygons with scaling, activating 2D drawing routines, etc.

SCMRULE (SCM) -- SCM Rule Assignment Utility

The **scmrule.ulc** **User Language** program is used to attach/detach rules to/from SCM figure list elements. The rules should be defined using the **RULECOMPNeural Rule System Compiler** to avoid undefined rule system errors.

SCMSETUP (SCM) -- Schematic Editor Setup

The **scmsetup.ulc** **User Language** program sets a series of default SCM parameters and display modes.

SCMTEXT (SCM) -- SCM Text Functions

The **scmtext.ulc** **User Language** program provides a menu with a series of advanced text processing functions such as change and replace texts, set text sizes, change standard to comment text (and vice versa), text class assignment for text visibility control, convert texts to areas or lines, write texts on arcs, delete texts with repetitive selection, etc.

SCMVAR (SCM) -- SCM Sheet Variant Selection

The **scmvar.ulc** **User Language** program activates a selectable SCM sheet variant with optional variant-specific data transfer.

SLABCHK (SCM) -- SCM Label Name Check

The **slabchk.ulc** **User Language** program performs a SCM label name check across all schematic plans of a project. Unique label names are reported as possible errors.

SLIBCOMP (SCM) -- Compare SCM Library Elements

slibcomp.ulc is an SCM library management utility program for comparing selectable SCM library elements to designate equivalent and/or different symbol definitions (relating to pin naming, pin placement, pin macro usage, logical library element assignment, etc.). The library comparison result report is displayed in a popup menu with file output option.

SLIBDOC (SCM) -- SCM Library Documentation

The **slibdoc.ulc** **User Language** program is an SCM library management program intended for documentation purposes. **SLIBDOC** automatically places SCM symbols or SCM labels of a freely selectable DDB file onto superior SCM database hierarchy level elements. Different standard sheet formats are supported including automatic legend box generation with library file name and sheet number denotation. The symbols to be included with the library documentation can be selected by symbol name pattern (wildcard). Symbols are aligned to horizontal base lines and text is placed with each symbol to denote the symbol name. Optionally, batch-driven PostScript plot generation can be performed to produce library documentation output.

SLIBNEWS (SCM) -- Generate SCM Library News File

slibnews.ulc is an SCM library management utility program for preparing and/or supporting controlled SCM library element releases. **SLIBNEWS** scans a selectable DDB file for SCM symbols which are not contained in any of a selectable directory's DDB file. These "new" symbols are then copied to a different DDB file, thus providing a collection of not yet released SCM library symbols.

SLIBUTIL (SCM) -- SCM Library Management

The **slibutil.ulc** **User Language** program provides a menu with a series of advanced SCM library management utilities such as producing library cross references, generating library documentation, performing library consistency checks, copying or deleting menu-selectable DDB file elements, activating library edit batches, logical library definition file export, etc.

SMOVPINN (SCM) -- Shift Vertical SCM Symbol Pin Names

The **smovpinn.ulc** **User Language** program moves the names of the upper and lower pins of the currently loaded SCM symbol to the corresponding pin origins (to free areas for connecting to corresponding pins at suitable pin macro definitions).

SNEXTSYM (SCM) -- SCM Next Symbol Placmenent

The **snextsym.ulc** **User Language** program places the next symbol from the project symbol pool or repeats the last symbol placement if the pool is empty.

SPICESIM (SCM) -- Spice Circuit Data Output

The **spicesim.ulc** **User Language** program generates Spice net lists. **SPICESIM** provides options for generating a Spice net list from the currently loaded SCM sheet (Whole Sheet) or from the symbols of the currently selected SCM group (Group Symbols). The output is written to a file with extension `.cir`. Additional commands such as `.LIB` and `.INCLUDE` for addressing the Spice target system libraries, for defining Simulation Control Parameters, etc. should be added to prepare the **SPICESIM** output file for the Spice Simulator. The Spice Model and Spice Pin Order functions of the **SCMRULE** **User Language** program can be used for assigning SCM symbol Spice model types and Spice output pin sequences.

SPOPCOL (SCM) -- SCM Color Setup

The **spopcol.ulc** **User Language** program activates a popup menu for displaying and changing the current **Schematic Editor** color setup.

SSPINMAC (SCM) -- Set SCM Symbol Pin Macros

The **sspinmac.ulc** **User Language** program automatically resets all marker macros of the currently loaded SCM symbol to their default.

SSVGOUT (SCM) -- Schematic SVG (Scalable Vector Graphics) Output

The **ssvgout.ulc** **User Language** produces an SVG (Scalable Vector Graphics) export file from the currently loaded schematic element.

SSYMATTR (SCM) -- SCM Symbol Attributes Database Management

The **ssymattr.ulc User Language** program contains SQL functions for defining new and managing predefined part attribute data for capacitor and resistor part types in a relational database. Additional functions are provided on SCM plan level for semi-automatic part attribute settings and default attribute assignments.

SSYMEDIT (SCM) -- SCM Symbol Edit Functions

The **ssymedit.ulc User Language** program activates a menu with advanced SCM symbol edit functions such as setting the symbol origin, placing pin lists, replacing and/or shifting pin groups, moving pin names, changing the element boundaries, etc.

SSYMORIG (SCM) -- Set SCM Symbol Origin

The **ssymorig.ulc User Language** program automatically sets the origin of the currently loaded SCM symbol to a selectable pin.

SSYMPATT (SCM) -- SCM Symbol Name Pattern Settings

The **ssympatt.ulc User Language** program provides functions for setting base name pattern data used for automatic symbol naming. A name pattern can be specified on symbol level. A naming mode (Standard or Number scan) and an optional name pattern start number can be specified on schematic sheet level.

STXFIN (SCM) -- TXF Schematic Data Input

The **stxfin.ulc User Language** program imports TXF schematic data from a selectable ASCII file into the **Schematic Editor**.

SYMATTDB (SCM) -- Create SQL Attribute Database for SCM Symbol Selection

The **symattdb.ulc User Language** program provides functions for creating a SCM symbol selection SQL database. The database can be created by importing the contents of `.csv` files from a selectable input directory.

SYMEDBAT (SCM) -- SCM Symbol Edit Batch

The **symedbat.ulc User Language** program performs the setup of a batch with a series of advanced symbol edit actions. The edit batch can be accomplished to all SCM symbols of selectable SCM library files of the current directory. **SYMEDBAT** provides functions for defining, changing or deleting special texts, for changing the element boundaries, for resetting pin macros, for moving pin names, for setting the symbol origin, for default attribute value assignments, for automatic text placement, etc.

SYMMAPDB (SCM) -- Create Symbol Mapping SQL Database

The **symmapdb.ulc User Language** program provides functions for creating a symbol gate selection SQL database and filling it with the `.map` file contents of a selectable input file.

SYMSEL (SCM) -- SCM Symbol Placement with Attribute Selection

The **symsel.ulc User Language** program provides symbol placement capability with attribute/symbol selection from a database generated with the **SYMATTDB User Language** program.

TBDVSCM (SCM) -- SCM Toolbar Design View Maintenance

The **tbdvscm.ulc User Language** program handles SCM toolbar design view update requests and provides features for symbol cloning and automatic template part attribute settings.

4.2.3 Layout Programs

The following **User Language** programs are compatible to the **Bartels AutoEngineer** layout system interpreter environment (i.e., they can be called from the **Layout Editor**, the **Autorouter** and the **CAM Processor**).

AIRLDENS (LAY) -- Airlines Density Diagram

The **airldens.ulc** **User Language** program generate a colored graphical airline (unroutes) density diagram for the currently loaded layout. Display area scaling depends on the resolution of the currently used graphic device.

CHECKLNL (LAY) -- Check Layout against Netlist

The **checklnl.ulc** **User Language** program checks the currently loaded layout against the corresponding net list data and reports unplaced parts, wrong package types and missing pin definitions in a popup menu with file output option.

CHKLMAC (LAY) -- List undefined Layout Macro References

The **chklmac.ulc** **User Language** program lists the names of all undefined macros (library elements) referenced from the currently loaded layout element to a popup menu with file output option.

CONBAE (LAY) -- Connection List Output

The **conbae.ulc** **User Language** program generates net list data output for the currently loaded layout in BAE ASCII net list format. The output is directed to a file. Sorting of part and net names is accomplished on request. Part placement data, as well as automatically generated test points and single-pin nets can optionally be included with the output.

DRILLOUT (LAY) -- Drilling Data Output

The **drillout.ulc** **User Language** program generates drill data output for the currently loaded layout. **DRILLOUT** supports both Sieb&Meier and Excellon drill data format and also provides an option for displaying drill data statistics. Output drill class selection is accomplished via popup menu. A predefined drill tool table is used if the user refrains from automatic drill tool table generation. Optionally, a heuristic sort algorithm for optimizing (i.e., minimizing) the drill route can be activated. The **DRILLOUT** source code provides a Compiler directive for optionally compiling **DRILLOUT** for generating graphical drill route display (drill route simulation).

DUMPPLC (LAY) -- Placement Data Output

The **dumpplc.ulc** **User Language** program generates placement data output for the currently loaded layout. Either generic or BAE placement format can be selected for output. BAE placement data coordinates can optionally be rounded to the current X input grid. The output is directed to a file.

EDF20CON (LAY) -- EDIF 2.0 Netlist Data Import

The **edf20con.ulc** **User Language** program can be used for importing EDIF 2.0 net list data. **EDF20CON** reads the selected `.edn` EDIF file and stores the net list to a DDB file using the element name from the net list file (on default, the `LAYDEFELEMENT` name defined with the **BSETUP** utility program is used). For consistency checks, **EDF20CON** requires access to the pin lists of the part symbols referenced from the net list. Layout part symbol info is loaded either from the destination project file or from the currently selected layout library. Required part symbols not yet defined in the project file are automatically copied from the library file. **EDF20CON** performs automatic generation of synthetic logical library entries subsequently required by the **Packager** (part/package assignments with 1:1 pin mapping). Netlist transfer process messages, warnings and error messages are listed both to the screen and to a logfile named `bae.log`. After successfully tranferring the net list, **EDF20CON** recommends to run the **Packager** to prepare the logical net list into a physical net list ready for layout.

Warning

EDF20CON is only tested with ORCAD-EDIF output. Output from other schematic capture systems may require **EDF20CON** adaptions for proper operation.

GENCAD (LAY) -- GENCAD 1.4 Layout Data Export

The **gencad.ulc** **User Language** generates a GENCAD 1.4 DAT layout data file from the currently loaded layout.

HYPLYNX (LAY) -- HyperLynx Layout Simulation Data Output

The **hyplynx.ulc** **User Language** program generates a HyperLynx (`.HYP`) layout simulation data file from the currently loaded layout. The layer stackup is inserted from a selectable external data file with file name extension `.stk`. Irrgular pad shapes are converted to bounding rectangles. Signal layer 1 is output as `BOTTOM`. The layer specified as top layer is output as `TOP`. Inner layers (layer 2 to toplayer - 1) are output as `Inner_Layer_n` (with `n` in the range from 2 to toplayer - 1).

ICAPNET (LAY) -- ICAP Logical Netlist Import

The **icapnet.ulc** **User Language** program imports packed (physical) net list data from Intusoft's ICAP/4 SCM/simulation tool (version 8.2.10 / 1843 or higher; Tango net list export) to internal **Bartels AutoEngineer** format ready for processing with the BAE **Packager**. **ICAPNET** reads the net list data from the selected net list file `project.net` and stores the net list to DDB file `project.ddb` using the element name specified in the net list file (default `netlist`). **ICAPNET** generates 1:1 pin assignment definitions for parts without logical library definitions in the default layout library. Net list transfer process messages, warnings and error messages are listed to the screen and to a log file named `bae.log`. Upon successful import, **ICAPNET** automatically starts the **Packager** to translate the logical net list to a physical net list ready for layout with pin/gate swap support according to logical library definitions.

IPCOUT (LAY) -- IPC-D-356 Test Data Output

The **ipcout.ulc** **User Language** program produces an IPC-D-356 format test data output for the currently loaded layout. The output is directed to a file. Vias are output as mid points. Solder mask information is derived from documentary layer 2. The output is sorted by nets and drills and contains drilling class specific plating information. A pin/via/drill count statistic is added at the end of the output file.

LAYDUMP (LAY) -- Layout ASCII Dump Import/Export

The **laydump.ulc** **User Language** program provides functions for exporting and importing layout element data in a generic ASCII format suitable for processing with BNF parsers. The export function processes the currently loaded layout element with selectable output length units and writes its output to an ASCII file.

Note

The export function is available in all layout modules. However, the import function is only available in the **Layout Editor**.

LAYDXFDO (LAY) -- Layout AutoCAD/DXF Output

The **laydxfdo.ulc** **User Language** program produces an AUTOCAD/DXF output of the currently loaded layout element. Output layers can be selected through a layer selection menu. Alternatively, all currently visible layers or programatically defined layers can be selected for output.

LAYEPS (LAY) -- Layout EPS/PDF Output

The **layeps.ulc** **User Language** program produces either Adobe Portable Document Format (PDF) or Encapsulated PostScript (EPS) output of the currently loaded layout element, featuring different output formats and scaling modes and optional clipping window selection. Multiple layers and/or display items are plotted simultaneously with layer-specific options for fill mode, color and/or gray value, hatching, dashed lines and standard line widths, respectively. The output layers as well as the layer-specific plot options are defined in a special **LAYEPS** source code variable which is intended for customization. **LAYEPS** also supports plot mirror options, plot rotate options (for 1:1 scale output) and an option for plotting menu-selectable layers or the currently visible layers with color selection, color assignment or automatic gray scaling instead of the predefined layer set.

LAYERUSE (LAY) -- Layout Library Layer Usage Report

layeruse.ulc is a layout library management utility program for analyzing layer assignments of selectable DDB file layout elements. The layer assignment statistics are listed to an ASCII report file.

LAYPCR (LAY) -- Layout Report

The **laypcr.ulc** **User Language** program provides detailed information about the currently loaded layout element such as DDB file name, element name, element origin coordinates, element boundary coordinates, element, DRC checking parameter settings, power layer definitions, top layer setting, reference listing (parts on layout, padstacks on part, pads/drilling on padstack). The report output is displayed in a popup menu with file output option.

LAYZMBRD (LAY) -- Zoom to Layout Board Outline

The **layzmbrd.ulc** **User Language** program accomplishes a zoom to the board outline of the currently loaded layout.

LBROWSE (LAY) -- Layout Symbol Browser

The **lbrowse.ulc** **User Language** program allows for the selection of layout symbols to be loaded and/or placed. A graphical display of the currently selected library symbol is displayed in a popup menu.

LCIFOUT (LAY) -- Layout CIF Data Export

The **lcifout.ulc** **User Language** program generates Caltech CIF output from the currently loaded layout element. A dialog box allows for specifying the CIF output file name, selecting either all or only group-selected elements for output, and chosing the CIF output mode (flat or hierarchical).

LDEFMANG (LAY) -- Layout Library Part Placement Preferences Definition

The **ldefmang.ulc** **User Language** program is used to assign placement preferences derived from the Neural Rules System such as default angle and mirroring mode to the currently loaded layout library element. These preferences are automatically considered by the appropriate layout placement functions. Rules named **`rot0`**, **`rot90`**, **`rot180`**, **`rot270`**, **`mirroroff`** and **`mirroron`** should be defined using the **Neural Rule System Compiler** **RULECOMP** to avoid undefined Rule System errors (see the **partplc.rul** rule source code file for the required rule definitions).

LLIBCOMP (LAY) -- Compare Layout Library Elements

llibcomp.ulc is a layout library management utility program for comparing selectable layout library elements in order to designate equivalent and/or different symbol definitions (relating to pin naming, pin placement, padstack macro usage, logical library element assignment, etc.). The library comparison result report is displayed in a popup menu with file output option.

LLIBNEWS (LAY) -- Generate Layout Library News File

llibnews.ulc is a layout library management utility program for preparing and/or supporting controlled layout library element releases. **llibnews.ulc** scans a selectable DDB file for layout symbols which are not contained in any of a selectable directory's DDB file. These "new" symbols are then copied to a different DDB file, thus providing a collection of not yet released layout library symbols.

LLIBUTIL (LAY) -- Layout Library Management

The **llibutil.ulc** **User Language** program provides a menu with a series of advanced layout library management utilities such as producing library cross references, generating library documentation, performing library consistency checks, copying or deleting menu-selectable DDB file elements, activating library edit batches, etc.

LMACCREF (LAY) -- Layout Macro Cross Reference

lmaccref.ulc is a layout library management utility program which produces a cross reference listing for the layout macros of a selectable DDB file. A report is displayed in a popup menu with file output option, showing each layout macro with the list of superior layout elements referencing that macro.

LSVGOUT (LAY) -- Layout SVG (Scalable Vector Graphics) Output

The **lsvgout.ulc** **User Language** produces an SVG (Scalable Vector Graphics) export file from the currently loaded layout element.

NETSTAT (LAY) -- Net Higlight/Visibility Status Database Management

The **netstat.ulc** **User Language** provides SQL database functions for storing and loading net highlight and airline visibility information and.or settings.

PARTLIST (LAY) -- Part List Output

The **partlist.ulc** **User Language** program produces a part list for the currently loaded layout with different output format options such as BAE format and CSV/DBF format. The part list output is sorted by part types according to a source code defined list of part name patterns (e.g., `r*` for resistors, `c*` for capacitors, etc.). Another predefined list of attribute names designates the attributes values to be used as lower priority part type sort criteria (e.g., `$val` for collecting all resistors or capacitors with the same value, `$llname` for collecting all ICs with the same function, etc.). The BAE part list output data is written to a file with extension `.pl`. The CSV/DBF ASCII output is written to a file with extension `.csv` and lists each part in a single line with part type designator, part name, package type and attributes. The semicolon is used to delimit the part data entries.

PSTKDRL (LAY) -- Padstack/Drill Definition Report

The **pstkdrl.ulc** **User Language** program produces the list of the padstack and/or drill holes defined in a selectable DDB file. The output is directed to a popup menu with file output option.

ROUTINFO (LAY) -- Routing Data Analysis

The **routinfo.ulc** **User Language** program creates comprehensive layout trace length analysis reports to be stored with the currently processed layout and/or checked against previous results in order to estimate multiple router pass performance or consider re-entrant routing results after redesign.

ROUTING (LAY) -- Routing Data Output

The **routing.ulc** **User Language** program produces an output of the trace data defined on the currently loaded layout. The output is directed to a file.

TBDVLAY (LAY) -- Layout Toolbar Design View Maintenance

The **tbdvlay.ulc** **User Language** program handles layout toolbar design view update requests.

TESTDATA (LAY) -- Test Data Output

The **testdata.ulc** **User Language** program produces a generic format test data output for the currently loaded layout. The output is directed to a file.

TRACEREP (LAY) -- Trace Report

The **tracerep.ulc** **User Language** program reports statistical trace data of the currently loaded layout (trace lengths, via counts, etc.). The report output is directed to a popup menu with file output option.

UNCONPIN (LAY) -- Report Unconnected Netlist Pins on Layout

The **unconpin.ulc** **User Language** program reports unrouted net list pins of the current layout in a popup menu with file output option.

WRLOUT (LAY) -- Layout WRL/VRML 3D Data Output

The **wrlout.ulc** **User Language** generates 3D data output in WRL (VRML) format from the currently loaded layout.

4.2.4 Layout Editor Programs

The following **User Language** programs are compatible to the **Layout Editor** interpreter environment (i.e., they can be called from the **Layout Editor**).

AUTONAME (GED) -- Automatic Part Rename Utilities

The **autoname.ulc** **User Language** program activates a menu with different functions for automatically renaming parts of the currently loaded layout. Part numbering is accomplished from the left top to the right bottom of the layout area. A two-pass renaming algorithm is applied to prevent equal source and destination name prefix specifications from causing numbering gaps.

CONCONV (GED) -- Physical Netlist Conversion

The **conconv.ulc** **User Language** program is used for transferring physical (i.e. packed) net list data from different ASCII formats (BAE, ALGOREX, Applicon, CADNETIX, CALAY, EEDESIGNER, Marconi ED, Mentor, MULTIWIRE, OrCAD, PCAD, RINF, SCICARDS, TANGO, VECTRON, VUTRAX, WIRELIST) to internal **Bartels AutoEngineer** format ready for processing with the BAE **Packager**. **CONCONV** reads the net list data from the selected net list file **`<project>.con`** and stores the net list to DDB file **`<project>.ddb`** using the element name specified in the net list file (on default the **`LAYDEFELEMENT`** name defined with the **BSETUP** utility program is used). Subsequently the **Packager** must be used to translate the (pseudo-physical) logical net list generated by **CONCONV** to a true physical net list ready for layout. For consistency checks **CONCONV** requires access to the pin lists of the part symbols referenced from the net list. Layout part symbol info is loaded either from the destination project file (**`<project>.ddb`**) or from the currently selected layout library. Required part symbols not yet defined in the project file are automatically copied from the library file. Part placement data (see net list formats BAE, Mentor, RINF, VUTRAX) can optionally be transferred to the destination layout (automatic part placement is carried out if the destination layout does not yet exist). **CONCONV** performs automatic generation of synthetic logical library entries subsequently required by the **Packager** (part/package assignments with 1:1 pin mapping). Net list transfer process messages, warnings, and error messages are listed both to the screen and to a logfile named **`bae.log`**. After successfully transferring the net list, **CONCONV** recommends to run the **Packager**.

CONUTIL (GED) -- Net List Functions

The **conutil.ulc** **User Language** program activates a menu with a series of advanced net list functions such as net data report, net highlight, open pins report, net list check, etc.

DRCBLOCK (GED/HighEnd) -- Advanced DRC Utilities

The **drcblock.ulc** **User Language** program provides functions for defining and managing blocks (sets) of layer specific design rule parameters. Please note that the application of DRC blocks is restricted to **BAE HighEnd**.

EDIFOUT (GED) -- EDIF 2.0 Netlist Data Output

The **edifout.ulc** **User Language** program generates an EDIF 2.0 net list for external PLD layout/fitter programs. **EDIFOUT** reads a (logical) net list and automatically creates a layout worksheet to extract the required data. The EDIF net list consists of a library description, an interface description and the actual net list. The output is directed to a file with extension **`.edf`**.

FONTEDIT (GED) -- Font Editor

The **fontedit.ulc** **User Language** program activates a menu with functions for processing font data, i.e., for editing character fonts. **FONTEDIT** provides functions for loading and writing fonts. When loading a font the characters of a selectable font are loaded from the **`ged.fnt`** file from the BAE programs directory; the corresponding data is transformed into polygons on side 1 of documentary layer 2. These polygons can be manipulated using **Layout Editor** area processing function. The font write function of **FONTEDIT** scans the font polygon data and produces a font file in BAE ASCII font data format which then can be re-transferred to **`ged.fnt`** using the **FONTCONV** BAE utility program.

GED_MS (GED) -- GED Mouse Action

The **ged_ms.ulc** **User Language** program is automatically activated when pressing the left mouse button in the workarea whilst the **Layout Editor** is idle. **GED_MS** provides a context-sensitive function menu for the object at the current mouse position. The Load Element, New Element and Mouse Mode functions are provided if no element is loaded.

GED_MSG (GED/HighEnd) -- GED Message System Action

The **ged_msg.ulc** **User Language** program provides functions to be performed when receiving messages from other **BAE HighEnd** modules. The actions to be triggered (part placement, group selection(s), net highlight, part set selection from SCM symbol group, etc.) and the objects to be processed are designated by the incoming message string.

GED_PLC (GED) -- GED Part Placement Action

The **ged_plc.ulc** **User Language** program is automatically activated after a part is placed to update the netlist assistent or other part relevant permanent dialog boxes.

GEDBOUND (GED) -- Set Layout Workspace/Element Boundary/Origin

The **gedbound.ulc** **User Language** program provides functions for automatically enlarging and/or shrinking the element boundaries of the currently loaded layout element and for adjusting the layout origin to the system input grid origin.

GEDDISP (GED) -- GED Display Functions

The **geddisp.ulc** **User Language** program provides a series of advanced **Layout Editor** display management functions.

GEDDRAW (GED) -- GED 2D Drawing Functions

The **geddraw.ulc** **User Language** program provides a series of advanced **Layout Editor** drawing functions such as producing circles, rectangles and/or arrows, performing distance and/or area measuring, and generating rulers.

GEDGROUP (GED) -- GED Group Functions

The **gedgroup.ulc** **User Language** program provides a menu with advanced **Layout Editor** group functions such as automatic selection and/or de-selection of specially defined groups of elements of the currently loaded layout element (e.g., all objects of a selectable type or with specific attributes, all fixed/unfixed, all mirrored/unmirrored, all on selectable layer, all visible/invisible, etc.). Advanced features such as automatic group copy, loading groups from different database hierarchy levels, layer-selective group delete, changing text size and/or trace width of all group-selected texts and/or traces or resetting non-default name and attribute text placements on group-selected parts are also provided.

GEDIO (GED) -- GED Data I/O Functions

The **gedio.ulc** **User Language** program provides a menu with a series of advanced **Layout Editor** data input/output functions. User-specific import/export functions can easily be added through **`addioitem`** commands in the **`bae.ini`** file of the BAE programs directory.

GEDMACL (GED) -- Layout Macro Load

The **gedmacl.ulc** **User Language** program loads the layout macro of a mouse-selectable part, padstack or pad from the currently loaded layout element to the **Layout Editor**.

GEDPART (GED) -- GED Part and Placement Functions

The **gedpart.ulc** **User Language** program activates a menu with a series of advanced part and placement functions such as part search, place parts by part name pattern, place part set, part placement with automatic part selection and position suggestion, automatic placement of selectable part hierarchy groups according to previous party group placement, swap parts, mirror and/or rotate selectable parts, placement data input and/or output, delete constructive parts, generate placement histogram, (part) height DRC, hotkey rotation angle settings for **LROTATE** and **RROTATE**, etc.

GEDPICK (GED) -- Layout Polygon Cross/Center Pick Functions

The **gedpick.ulc** **User Language** program provides functions to snap to the cross points of polygon lines and centers of interpolated arc polygons. This program is intended for implicit hotkey program call (e.g., ⌧ or ⌧).

GEDPOLY (GED) -- GED Polygon Functions

The **gedpoly.ulc** **User Language** program provides a menu with a series of advanced polygon processing functions such as changing polygon layers, defining polygon types, setting polygon line widths, converting polygon corners into arcs or

diagonal segments, splitting and/or joining documentary lines, copying group-selected polygons with scaling, activating 2D drawing routines, height DRC settings, etc.

GEDRULE (GED) -- Layout Rule Assignment Utility

The **gedrule.ulc** **User Language** program is used to attach/detach rules to/from layout figure list elements and/or layout groups. The rules should be defined using the **RULECOMPNeural Rule System Compiler** to avoid undefined rule system errors.

GEDSETUP (GED) -- Layout Editor Setup

The **gedsetup.ulc** **User Language** program sets a series of default **Layout Editor** parameters and display modes.

GEDTEXT (GED) -- GED Text/Drill Functions

The **gedtext.ulc** **User Language** program provides a menu with a series of advanced text processing functions such as set text sizes, change text layers, upper case text strings, lower case text strings, convert texts to traces, areas or lines, plot text line width settings, write texts on arcs, delete texts with repetitive selection, etc.

GEDTRACE (GED) -- GED Trace and Routing Functions

The **gedtrace.ulc** **User Language** program activates a menu with a series of advanced trace, via and net list functions such as automatic rounding of trace corners, trace end cutting, (equidistant) parallel trace generation on alternate layers, trace to split power plane conversion, teardrop generation, trace width change, trace segment split, trace length adjustment, net data query, net highlight with optional zoom, net list output, trace length query, power layer re-definition, trace and/or unroutes report, pin/via statistics display, via placement, via movement, via type change, trace edit display mode selection, etc.

GEDVAR (GED) -- Layout Variant Selection

The **gedvar.ulc** **User Language** program activates a selectable layout variant.

GEDVIA (GED) -- GED Via Functions

The **gedvia.ulc** **User Language** program activates a menu with a series of advanced via functions such as pin/via statistics display, via placement, via movement, repetitive via selection, substitution of mouse-selectable or group-selected via types, via to part conversion, etc.

GENLMAC (GED) -- Layout Library Element Generator

The **genlmac.ulc** **User Language** program provides utilities for semi-automatic generation of layout library elements such as parts, padstacks and/or pads. The pad generator allows for the definition of circle, square, rectangular, finger, bullet, octagon, angular and drill symbol pads with pad names automatically derived from pad type and the pad size. The padstack generator allows for the definition of standard pins, SMD pins, plated or non-plated drill holes, standard vias and staggered vias. Padstack names are automatically retrieved from padstack type and pin size (selected with the pad shape and/or the drill size to be used on the corresponding padstack). Blind and buried vias can be defined covering adjacent signal layers in layer range 1 to 8. The part generator allows for the definition of part package types for resistors, capacitors, and electrolytic capacitors with different part body shapes (rectangular/block, cylindrical/tube, plate/drop, disk/drop) and part pin lead-out types (radial, axial, axial/stand-upright). Part package names are automatically derived from part type pin distance specification, part body shape and dimensions, and pin drill/wire diameters.

LAYDXFDI (GED) -- Layout AutoCAD/DXF Input

The **laydxfdi.ulc** **User Language** program loads AUTOCAD/DXF drawing data onto the currently loaded layout element. Input length units, input offsets and layer assignments can be adjusted as required.

LAYEDBAT (GED) -- Layout Library Batch Editor

The **layedbat.ulc** **User Language** program sets up a batch with a series of advanced layout edit actions to be applied to all layouts and/or layout macros of selectable layout library files of the current directory. **layedbat.ulc** provides functions for changing element boundaries, changing padstack and/or pad macros, changing layer assignments, assigning rules, defining, changing or deleting special texts, setting part origins and pick points, moving pin names, etc.

LCIFIN (GED) -- Layout CIF Data Import

The **lcifin.ulc User Language** program reads Caltech CIF data from a selectable CIF file onto the currently loaded layout element. The loaded CIF structures are placed on the current layout level. After successfully importing the CIF file, a group selection of the created elements is automatically applied.

LERRLIST (GED) -- Layout DRC Error List Display

The **lerrlist.ulc User Language** program displays a design rule check (DRC) error list with the option to zoom to selectable error markers.

LLIBDOC (GED) -- Layout Library Documentation

The **llibdoc.ulc User Language** program is a layout library management program intended for documentation purposes. **LLIBDOC** automatically places layout parts, layout padstacks or layout pads of a freely selectable DDB file onto superior layout database hierarchy level elements. Different standard sheet formats are supported including automatic legend box generation with library file name and sheet number denotation. The symbols to be included with the library documentation can be selected by symbol name pattern (wildcard). Symbols are aligned to horizontal base lines and text is placed with each symbol to denote the symbol name. The layout elements generated with **LLIBDOC** can be plotted (e.g., using the **LAYEPS User Language** program) to produce library documentation output.

LMACREAD (GED) -- Layout Macro Definition Import

The **lmacread.ulc User Language** imports layout macro definitions from a text file to the **Layout Editor**.

LPINTRC (GED) -- Layout Pin Trace Connection

The **lpintrc.ulc User Language** program automatically routes a trace connection to the origin of the pin currently under the graphic cursor. **LPINTRC** works only whilst manipulating either the start or the end point of a manually routed trace and must be called through a hotkey (e.g., through the p or the P key). When manipulating the trace start point, **LPINTRC** routes a trace corner to the pin currently under the graphic cursor. When manipulating the trace end point, **LPINTRC** creates a 45 degree segment from the last trace point and then connects to the pin origin using a straight segment. In either case, the current grid and angle lock modes stay in effect.

LSYMEDIT (GED) -- Layout Part Symbol Edit Functions

The **lsymedit.ulc User Language** program activates a menu with advanced layout symbol edit functions such as setting the part origin and/or the insertion pick, placing pin lists and/or pin rows, changing the element boundaries, etc.

LTXFIN (GED) -- TXF Layout Data Input

The **ltxfin.ulc User Language** program imports TXF layout data from a selectable ASCII file into the **Layout Editor**.

MT_ROUT (GED) -- Mikami-Tabuchi Router

The **mt_rout.ulc User Language** performs a Mikami Tabuchi line routing between two mouse selectable connection points. During the connection point selection, the right mouse button activates a context menu with router settings option. The calling sequence `mt_rout:'param'` activates the parameter settings dialog box directly without connection routing.

POLYRND (GED) -- Change Polygon Corners to Arcs or 45 Degree Segments

The **polyrnd.ulc User Language** program automatically converts polygon corners into arcs or 45 degree segments.

READLPLC (GED) -- Layout Placement Data Input

The **readlplc.ulc User Language** program reads placement data from a selectable ASCII file and automatically performs the therein defined part placement. **READLPLC** is capable of processing the placement data formats produced with the **DUMPPLC User Language** program.

TEARDROP (GED) -- Teardrop Functions

The **teardrop.ulc** **User Language** program automatically generates teardrops for all trace ends on vias and/or part pins. Teardrops are trace ends fluently widened to the diameter of the matching via. Teardrop generation can optionally be restricted to (selectable) vias or part pins. Additional functions are provided for selecting and/or deleting teardrops.

TRACERND (GED) -- Change Trace Corners to Arcs

The **tracernd.ulc** **User Language** program automatically converts trace corners of selectable traces of the currently loaded layout to trace arc segments with a freely selectable radius.

TRCPUSH (GED) -- Push Trace Segments

The **trcpush.ulc** **User Language** program automatically shifts mouse-selectable trace segments by a freely selectable shift vector. **TRCPUSH** tries to push away neighboring trace segments on the same layer to provide enough space for shoving the selected trace segment, i.e., **TRCPUSH** can also be used to push trace segment bundles. When shifting a certain trace segment, **TRCPUSH** automatically shifts, shortens and/or lengthens adjacent trace segments in order to leave trace segment orientations and/or directions unchanged, thus providing powerful push'n'shove features for automatically shifting trace bunches.

VHDLOUT (GED) -- VHDL Netlist Data Output

The **vhdlout.ulc** **User Language** program is generates a VHDL netlist output for use with external PLD layout/fitter programs. **VHDLOUT** reads a (logical) netlist and automatically creates a layout worksheet to extract the required data. In hierachical designs, the VHDL netlist contains module ports. The output is directed to a file with the file name extension `.vhdl`.

4.2.5 Autorouter Programs

The following **User Language** programs are compatible to the **Autorouter** interpreter environment (i.e., they can be called from the **Autorouter**).

AR_MS (AR) -- Autorouter Mouse Action

The **ar_ms.ulc User Language** program is automatically activated when pressing the left mouse button in the workarea whilst the **Autorouter** is idle. **AR_MS** provides a context-sensitive function menu for the object at the current mouse position. The Load Element function is provided if no element is loaded.

ARDISP (AR) -- Autorouter Display Functions

The **ardisp.ulc User Language** program provides a series of advanced **Autorouter** display management functions.

ARIO (AR) -- Autorouter Data I/O Functions

The **ario.ulc User Language** program provides an **Autorouter** menu with a series of advanced layout-specific data input/output functions. User-specific import/export functions can easily be added through `addioitem` commands in the `bae.ini` file of the BAE programs directory.

ARSETUP (AR) -- Autorouter Setup

The **arsetup.ulc User Language** program sets a series of default **Autorouter** parameters and display modes.

4.2.6 CAM Processor Programs

The following **User Language** programs are compatible to the **CAM Processor** interpreter environment (i.e., they can be called from the **CAM Processor**).

CAM_MS (CAM) -- CAM Processor Mouse Action

The **cam_ms.ulc** **User Language** program is automatically activated when pressing the left mouse button in the workarea whilst the **CAM Processor** is idle. **CAM_MS** provides a context-sensitive function menu for the object at the current mouse position. The Load Element function is provided if no element is loaded.

CAMBATCH (CAM) -- CAM Batch Output

The **cambatch.ulc** **User Language** program provides a tool for batch driven CAM data output intended for customization. **CAMBATCH** automatically generates Gerber data output for a series of predefined signal, power and documentary layers as well as drilling data output.

CAMBATDB (CAM) -- CAM Batch Database

The **cambatdb.ulc** **User Language** program provides utilities for managing a database with user-defined CAM data output batch functions in Windows and Motif enviroments.

CAMIO (CAM) -- CAM Processor Data I/O Functions

The **camio.ulc** **User Language** program provides a menu with a series of advanced **CAM Processor** data input/output functions. User-specific import/export functions can easily be added through `addioitem` commands in the `bae.ini` file of the BAE programs directory.

CAMSETUP (CAM) -- CAM Processor Setup

The **camsetup.ulc** **User Language** program sets a series of default **CAM Processor** and Gerber plot parameters and display modes and activates a table-defined pen-assignment for multi-layer plots.

GAPTUTIL (CAM) -- Gerber Aperture Table Management

The **gaptutil.ulc** **User Language** program provides Gerber aperture table management utilities such as input/output of BAE and/or ECAM ASCII formatted Gerber aperture table files.

GBALLSIG (CAM) -- CAM Signal Layers Gerber Output Batch

The **gballsig.ulc** **User Language** program accomplishes a batch driven Gerber data output for all signal layers of the currently loaded layout.

GINSOUT (CAM) -- Generic Insertion Data Output

The **ginsout.ulc** **User Language** program produces generic insertion data output for the currently loaded layout. The output format is programmed through the definitions loaded from an insertion format specification file (file name extension .ifs). The insertion data coordinates are generated relative to the current CAM origin.

POWDCHK (CAM) -- Power Layer Heat-trap Check

The **powdchk.ulc** **User Language** program lists the positions of all heat traps possibly isolated by surrounding drill isolations. It is strongly recommended to check and correct the definition of any heat trap listed by **POWDCHK** before generating manufacturing data.

4.2.7 CAM View Programs

The following **User Language** programs are compatible to the **CAM View** interpreter environment (i.e., they can be called from the **CAM View** module).

CV_MS (CV) -- CAM View Mouse Action

The **cv_ms.ulc User Language** program is automatically activated when pressing the left mouse button in the workarea whilst the **CAM View** module is idle. **CV_MS** provides a context-sensitive function menu for the object at the current mouse position. Executable functions from the File menu are provided if no CAM data is loaded.

CVSETUP (CV) -- CAM View Setup

The **cvbatld.ulc User Language** program provides utilities for reading and writing CAM data output file sets generated with the **CAMBATDB** batch output utility.

CVSETUP (CV) -- CAM View Setup

The **cvsetup.ulc User Language** program sets a series of default **CAM View** parameters and display modes.

4.2.8 IC Design Programs

The following **User Language** programs are compatible to the **Bartels AutoEngineer IC Design** system interpreter environment (i.e., they can be called from the **Chip Editor**).

CHKIMAC (ICD) -- List undefined IC Design Macro References

The **chkimac.ulc** **User Language** program lists the names of all undefined macros (library elements) referenced from the currently loaded **IC Design** element to a popup menu with file output option.

CHECKINL (ICD) -- Check IC Design against Netlist

The **checkinl.ulc** **User Language** program checks the currently loaded **IC Design** against the corresponding net list data and reports unplaced parts, wrong cell types and missing pin definitions in a popup menu with file output option.

ICDPCR (ICD) -- IC Design Report

The **icdpcr.ulc** **User Language** program provides detailed information about the currently loaded **IC Design** element such as DDB file name, element name, element origin coordinates, element boundary coordinates, element size, DRC checking parameter settings, reference listing (cells on IC layout, pins/cells on cell). The report output is displayed in a popup menu with file output option.

4.2.9 Chip Editor Programs

The following **User Language** programs are compatible to the **Chip Editor** interpreter environment (i.e., they can be called from the **Chip Editor**).

CED_MS (CED) -- Chip Editor Mouse Action

The **ced_ms.ulc User Language** program is automatically activated when pressing the left mouse button in the workarea whilst the **Chip Editor** is idle. **CED_MS** provides a context-sensitive function menu for the object at the current mouse position. The Load Element, New Element and Mouse Mode functions are provided if no element is loaded.

CEDDISP (CED) -- Chip Editor Display Functions

The **ceddisp.ulc User Language** program provides a series of advanced **Chip Editor** display management functions.

CEDGROUP (CED) -- Chip Editor Group Functions

The **cedgroup.ulc User Language** program provides a menu with advanced **Chip Editor** group functions such as automatic selection and/or de-selection of specially defined groups of elements of the currently loaded IC Design element (e.g. all objects of a selectable type, all fixed/unfixed, all mirrored/unmirrored, all on selectable layer, etc.). There are also functions available for automatic group copy, for layer-selective group delete or for changing text size and/or trace width of all group-selected texts and/or traces.

CEDMACL (CED) -- IC Design Macro Load

The **cedmacl.ulc User Language** program loads the **IC Design** macro of a mouse-selectable cell or pin from the currently loaded **IC Design** element to the **Chip Editor**.

CEDPART (CED) -- Chip Editor Macro and Placement Functions

The **cedpart.ulc User Language** program activates a menu with a series of advanced macro and placement functions such as place cells specified by macro name pattern, swap macros, mirror and/or rotate selectable macros, placement data input and/or output, delete constructive macros, placement data query, etc.

CEDPOLY (CED) -- Chip Editor Polygon Functions

The **cedpoly.ulc User Language** program provides a menu with a series of advanced polygon processing functions such as changing polygon layers, defining polygon types, copying group-selected polygons with scaling, activating 2D drawing routines, etc.

CEDSETUP (CED) -- Chip Editor Setup

The **cedsetup.ulc User Language** program sets a series of default **Chip Editor** parameters and display modes.

4.3 User Language Program Installation

This section provides information on how to install and/or compile BAE **User Language** programs are installed and how to apply key bindings and menu assignments.

The BAE software installs close to 200 pre-compiled **User Language** programs to the **ulcprog.vdb** file of the BAE programs directory. The corresponding source files are are also provided in the **User Language** directory (**baeulc**). See chapter 4.2 for a complete listing and short descriptions of the BAE **User Language** programs.

4.3.1 Program Compilation

Usually, it is *not* necessary to (re-)compile the **User Language** programs delivered with the BAE software since the compiled programs are installed to the **ulcprog.vdb** file of the BAE programs directory. Nevertheless, the **User Language** directory provides several batch files for automatically compiling all BAE **User Language** programs. The **CPLSLL** (ComPiLe with Static Link Library) batch file is recommended for compilation. The compile batch can be started from the **User Language** directory (**baeulc**) by entering

```
> cplsll ↵
```

to an MS-DOS-Prompt (with the **PATH** variable pointing to the BAE programs directory) or with the

```
> cplsll.bat ↵
```

command from a Linux or UNIX shell. The compilation process might last some time according to the power of your computer.

4.3.2 Menu Assignments and Key Bindings

Some of the BAE **User Language** programs define implicit **User Language** program calls for activating a modified BAE user interface with many additional functions (startups, toolbars, menu assignments, key bindings). You can add even more functions, or you can modify and/or reset the predefined menu assignments and key bindings.

The **User Language** startup program **BAE_ST** is automatically started when entering a **User Language Interpreter** environment (**Schematic Editor**, **Layout Editor**, **Autorouter**, **CAM Processor**, **CAM View** or **Chip Editor**). **BAE_ST** calls the **UIFSETUP** **User Language** program which activates predefined menu assignments and key bindings for the current BAE program module. Any changes to the menu assignments and key bindings require modification and re-compilation of the **UIFSETUP** source code only. The **HLPKEYS** **User Language** program can be used to list the current key bindings. With the predefined menu assignments of **UIFSETUP** activated, **HLPKEYS** can be called from the **Key Bindings** function of the **Help** menu. Menu assignments, although obvious from the BAE user interfaces, can be listed with the **UIFDUMP** **User Language** program. The **UIFRESET** **User Language** program can be used to reset all currently defined menu assignments and key bindings of the currently active BAE program module. Simply give it a try and run the **UIFRESET** program; we bet you'll be surprised to see the difference (the predefined menu layout can always be restored by calling **UIFSETUP**). The **UIFSETUP**, **UIFDUMP** and **UIFRESET** programs can also be called from the menu of the **KEYPROG** **User Language** program which provides additional facilities for online key programming and **User Language** program help info management.

.

Appendix A
Conventions and Definitions

This appendix describes the conventions and the valid parameter value ranges for accessing the **User Language** index variable types and system functions. Definitions are given for the terms interpreter environment and caller type. The valid value range definitions are listed according to the corresponding caller types.

Contents

Tables

A.1 Conventions

This section describes the basic conventions for accessing the **User Language** index variable types and system functions.

A.1.1 Interpreter Environment

Each **Bartels AutoEngineer** program module providing the **User Language Interpreter** facilities is called an interpreter environment. Different sets of **User Language** index variable types and/or system functions are implemented and/or available in different interpreter environments. Each **User Language** program can be called just from those interpreter environments containing all of the **User Language** index types and system functions referenced by the program.

A.1.2 Caller Type

The **User Language** caller type is defined to support **User Language** program compatibility checks. A caller type is a coded value which represents a set of compatible **User Language** interpreter environments. Each **User Language** index variable type and each **User Language** system function is assigned to a certain caller type, which designates the compatible interpreter environment(s). The **User Language Compiler** combines the caller types of the program's system references (unless the `#pragma ULCALLERSTD` preprocessor statement is used; see chapter 2.6.5 for details). The resulting program caller type must match a valid set of interpreter environments (otherwise no valid program can be generated). When calling a **User Language** machine program, the **User Language Interpreter** checks whether the caller type associated with the program is compatible to the current interpreter environment (otherwise the program cannot be executed).

Table A-1 contains the list of **Bartels User Language** caller types.

Table A-1: User Language Caller Types

Code	Caller Type Designator
STD	Standard
CAP	Schematic Capture Data Access
SCM	**Schematic Editor**
LAY	Layout Data Access
GED	**Layout Editor**
AR	**Autorouter**
CAM	**CAM Processor**
CV	**CAM View**
ICD	IC Design Data Access
CED	**Chip Editor**

Table A-2 contains the information about the compatibility of the **Bartels User Language** caller types. According to this table the **User Language Compiler** performs a compatibility test for the system references of a program. All **User Language** index variable types and system functions referenced by a program must be assigned to a set of compatible caller types.

Table A-2: Compatibility Caller Type versus Caller Type

Caller Type	STD	CAP	SCM	LAY	GED	AR	CAM	CV	ICD	CED
STD	x	x	x	x	x	x	x	x	x	x
CAP	x	x	x	-	-	-	-	-	-	-
SCM	x	x	x	-	-	-	-	-	-	-
LAY	x	-	-	x	x	x	x	-	-	-
GED	x	-	-	x	x	-	-	-	-	-
AR	x	-	-	x	-	x	-	-	-	-
CAM	x	-	-	x	-	-	x	-	-	-
CV	x	-	-	-	-	-	-	x	-	-
ICD	x	-	-	-	-	-	-	-	x	x
CED	x	-	-	-	-	-	-	-	x	x

Table A-3 contains the information about the compatibility of the **Bartels User Language** caller types to the **Bartels User Language Interpreter** environments. According to this table the **User Language Interpreter** accomplishes compatibility checks on the **User Language** programs to be called. A **User Language** program can only be executed, if the program's caller type is compatible to the current interpreter environment.

Table A-3: Compatibility Caller Type versus Interpreter

Caller Type	Interpreter					
	SCM	**GED**	**AR**	**CAM**	**CV**	**CED**
STD	x	x	x	x	x	x
CAP	x	-	-	-	-	-
SCM	x	-	-	-	-	-
LAY	-	x	x	x	-	-
GED	-	x	-	-	-	-
AR	-	-	x	-	-	-
CAM	-	-	-	x	-	-
CV	-	-	-	-	x	-
ICD	-	-	-	-	-	x
CED	-	-	-	-	-	x

A.2 Value Range Definitions

User Language value ranges are defined for a series of index variable type elements and system function parameters. This section contains a complete survey over these definitions. In the description for the index variable types (see appendix B) and system functions (see appendix C) the herein defined value ranges are referenced by the corresponding designation of the value range.

A.2.1 Standard Value Ranges (STD)

The following codes are valid for all caller types. I.e., they define valid value ranges for special elements of index variable types and/or system function parameters in the **Schematic Editor**, the **Layout Editor**, the **Autorouter**, the **CAM Processor** and the **Chip Editor** interpreter environments. The naming of the value range definitions emerges from the keyword **STD** and a continuous number.

STD1 - DDB Element Class:

```
(-1) = Unknown/invalid DDB class
 100 = Layout Plan
 101 = Layout Part
 102 = Layout Padstack
 103 = Layout Pad
 150 = Layout Paths
 151 = Layout Editor Connectivity
 160 = Layout DRC Parameters
 200 = Physical Connection List
 201 = Connection Assignments
 202 = Backannotation Request
 300 = Autorouter Data
 301 = Autorouter Parameter
 400 = BAE Character Font
 401 = BAE Setup Data
 402 = Parameter Setup Data
 500 = Gerber Table
 501 = Layout Color Table
 502 = SCM Color Table
 510 = IC Color Table
 511 = GDS Structure
 700 = BAE Messages
 800 = SCM Sheet
 801 = SCM Symbol
 802 = SCM Marker
 803 = SCM Label
 850 = SCM Part List
 900 = Logical Library Part
 901 = Logical Connection List
 902 = Physical Pin Info
 903 = Packager Parameters
1000 = IC Layout
1001 = IC Cell
1002 = IC Pin
1050 = IC Paths
1051 = IC Connectivity
1052 = IC Autorouter Data
1200 = User Language Program
1201 = User Language Library
1400 = Design Rule
1401 = Design Rule List
1402 = Design Rule Source
4096 = SQL Structure Table Info
4097 = SQL Free Table Info
4352 = SQL user-defined Table
   : = :
8191 = SQL user-defined Table
```

STD2 - Length Units:

```
Length values are given in meters (unless otherwise mentioned).
```

STD3 - Angle Units:

```
Angle values are given in radians (unless otherwise mentioned).
```

STD4 - Menu Item Numbers:

```
[   0,  99] = Main menu items
[ 100, 199] = Menu 1 items
[ 200, 299] = Menu 2 items
[ 300, 399] = Menu 3 items
[ 400, 499] = Menu 4 items
[ 500, 599] = Menu 5 items
[ 600, 699] = Menu 6 items
[ 700, 799] = Menu 7 items
[ 800, 899] = Menu 8 items
[ 900, 999] = Menu 9 items
[1000,1099] = Menu 10 items
       9003 = Menu Function Undo
       9004 = Menu Function Redo
       9005 = Close currently loaded element
       9006 = Jump to Schematic Editor
       9007 = Jump to Packager
       9008 = Jump to Packager, run Packager and
              jump to Schematic Editor if no Packager error occurred
       9009 = Jump to Packager, run Packager and
              jump to Layout Editor if no Packager error occurred
       9010 = Jump to Layout Editor
       9012 = Jump to Autorouter
       9013 = Jump to CAM Processor
       9014 = Jump to CAM View
       9015 = BAE HighEnd/IC Design: Jump to IC Design Chip Editor
       9016 = BAE HighEnd/IC Design: Jump to IC Design CIF View
       9017 = BAE HighEnd/IC Design: Jump to IC Design GDS View
       9018 = BAE HighEnd/IC Design: Jump to IC Design Cell Placer
       9020 = Jump to BAE Setup
       9021 = Jump to BAE Main Menu
       9022 = BAE HighEnd: Start Schematic Editor instance
       9023 = BAE HighEnd: Start Packager instance
       9024 = BAE HighEnd: Start and run Packager instance
       9025 = BAE HighEnd: Start Layout Editor instance
       9027 = BAE HighEnd: Start Autorouter instance
       9028 = BAE HighEnd: Start CAM Processor instance
       9029 = BAE HighEnd: Start CAM View instance
       9030 = BAE HighEnd: Start IC Design Chip Editor instance
       9031 = BAE HighEnd: Start IC Design CIF View instance
       9032 = BAE HighEnd: Start IC Design GDS View instance
       9033 = BAE HighEnd: Start IC Design Cell Placer instance
       9035 = BAE HighEnd: Start BAE Main Menu instance
       9036 = Menu function Exit
       9038 = Menu function Help
       9039 = Menu function Help to
       9041 = Jump to Packager, run Packager and
              jump to Chip Editor if no Packager error occurred
       9042 = Cut to Clipboard
       9043 = Copy to Clipboard
       9044 = Paste from Clipboard
       9048 = Start program and wait for its termination
       9049 = Start program and return
```

Windows/Motif dialogs:

```
5000 = Schematic Editor: Display parameters dialog
5001 = Schematic Editor: General Schematic Editor parameters dialog
5002 = Schematic Editor: SCM Plot parameters dialog
5005 = Layout Editor: Display parameters dialog
5006 = Layout Editor: General Layout Editor parameters
5008 = Layout Editor: Copper fill parameters dialog
5010 = Layout Editor: Autoplacement parameters dialog
5000 = Autorouter: Display parameters dialog
5001 = Autorouter: General Autorouter parameters dialog
5002 = Autorouter: Automatic placement parameters dialog
5003 = Autorouter: Autorouting options dialog
5004 = Autorouter: Autorouting control parameters dialog
5005 = Autorouter: Autorouting strategy parameters dialog
5006 = Autorouter: Autorouting batch setup dialog
5000 = CAM Processor: Display parameters dialog
5001 = CAM Processor: Control plot parameters dialog
5002 = CAM Processor: Gerber photoplot parameters dialog
5003 = CAM Processor: Drilling data output parameters dialog
5004 = CAM Processor: General CAM/Plot parameters dialog
5000 = CAM View: Display parameters dialog
5001 = CAM View: General CAM View parameters dialog
```

Standard menu item numbers are calculated using the

```
100 × main menu number + submenu number
```

formula with numbering starting at zero. The main menu number 0 is reserved for the Undo, Redo menu, i.e., the Undo and Redo functions must be called through menu numbers 9003 and/or 9004, respectively.

STD5 - Dialog Element Parameter Type:

```
                    Parameter values:
     0 = 0x000000 = String value
     1 = 0x000001 = Boolean value (check box)
     2 = 0x000002 = Integer value
     3 = 0x000003 = Double value
                    Display element types:
     4 = 0x000004 = Label/title text/string
     5 = 0x000005 = Horizontal seperator graphic
     6 = 0x000006 = Vertical seperator graphic
                    Multiple choice dialog element types:
     7 = 0x000007 = Radio box first option
     8 = 0x000008 = Radio box next option
     9 = 0x000009 = Selection box base entry
    10 = 0x00000A = Selection box next entry
 65536 = 0x010000 = List box base entry
 65537 = 0x010001 = List box next entry
                    Dialog button types:
    11 = 0x00000B = Action button
    12 = 0x00000C = OK button
    13 = 0x00000D = Abort button
                    Special parameter type codes:
    14 = 0x00000E = Dummy dialog element
983055 = 0x0F000F = Parameter type mask (for parameter type queries)
                    Numerical value types:
    16 = 0x000010 = Signed numerical value type
    32 = 0x000020 = Distance/length numerical value type
    64 = 0x000040 = Rotation angle numerical value type
                    Value range checking parameter types:
   128 = 0x000080 = Lower boundary checking parameter type
   256 = 0x000100 = Upper boundary checking parameter type
   512 = 0x000200 = Immediate lower boundary checking parameter type
  1024 = 0x000400 = Immediate upper boundary checking parameter type
                    Miscellaneous parameter types:
  2048 = 0x000800 = Empty string display field parameter type
  4096 = 0x001000 = Bold typeface font parameter type
  8192 = 0x002000 = Fixed width font parameter type
 16384 = 0x004000 = Double click confirmation parameter type
 32768 = 0x008000 = Edit disabled parameter type
1048576 = 0x100000 = Single click confirmation parameter type
```

STD6 - Interaction Mode:

```
0 = Input interactive
1 = Input automatic
```

STD7 - Coordinate Display Mode:

```
0 = Display/input in mm units (micrometer units in IC Design)
1 = Display/input in Inch units (mm units in IC Design)
```

STD8 - Grid Lock Flag:

```
0 = Grid unlocked
1 = Grid locked
```

STD9 - Angle Lock Flag:

```
0 = Angle unlocked
1 = Angle locked
```

STD10 - Workspace Flag:

```
0 = Element/object is out of workspace
1 = Element/object is in workspace
```

STD11 - Fixed Flag:

```
0 = Element/object is unfixed
1 = Element/object is fixed
```

STD12 - Element Glued Mode:

```
0 = Element/object is not glued
2 = Element/object is glued
```

STD13 - Group Flag:

```
0 = Element/object is not selected to group
1 = Element/object is selected to group
2 = Toggle group selection status
```

STD14 - Mirror Mode:

```
0 = Element/object is not mirrored
1 = Element/object is mirrored
```

STD15 - Polygon Point Type:

```
0 = Normal point
1 = Left arc center point
2 = Right arc center point
```

STD16 - Macro Completion Status:

```
xxxxxxx1 = Macro is completed (bit mask)
xxxxxx1x = Macro is missing (bit mask)
    else = internal
```

STD17 - Mouse Button Key Codes:

```
0 = Keyboard Input
1 = Left Mouse Button
2 = Middle Mouse Button
3 = Right Mouse Button
```

STD18 - Color Codes:

```
  0 = Black (no color)
  1 = Blue
  2 = Green
  3 = Cyan
  4 = Red
  5 = Magenta
  6 = Brown
  7 = Light Gray
  8 = Dark Gray
  9 = Light Blue
 10 = Light Green
 11 = Light Cyan
 12 = Light Red
 13 = Light Magenta
 14 = Yellow
 15 = White
 -1 = Black faded-out
 -2 = Blue faded-out
 -3 = Green faded-out
 -4 = Cyan faded-out
 -5 = Red faded-out
 -6 = Magenta faded-out
 -7 = Brown faded-out
 -8 = Light Gray faded-out
 -9 = Dark Gray faded-out
-10 = Light Blue faded-out
-11 = Light Green faded-out
-12 = Light Cyan faded-out
-13 = Light Red faded-out
-14 = Light Magenta faded-out
-15 = Yellow faded-out
-16 = White faded-out
```

STD19 - Drawing Mode:

```
0 = Replace
1 = Clear
2 = Set
3 = Complement
```

STD20 - Polygon Fill/Drawing Mode:

```
 0 = Closed line polygon
 1 = Closed fill polygon
 2 = Open line polygon
 3 = Closed pattern fill polygon
+4 = Dashed polygon outline
+8 = Dotted polygon outline
```

STD21 - Interaction Item Store Mode:

```
0 = Append interaction placeholder to interaction queue end
1 = Append automatic interaction to interaction queue end
2 = Insert interaction placeholder at interaction queue start
3 = Insert automatic interaction at interaction queue start
```

A.2.2 Schematic Capture Value Ranges (CAP)

The following codes are valid for caller types CAP and SCM. I.e., they define valid value ranges for special elements of index variable types and/or system function parameters in the **Schematic Editor** interpreter environment. The naming of the value range definitions emerges from the keyword **CAP** and a continuous number.

CAP1 - Schematic Capture Text Mode:

```
0 = Standard Text (bit mask to be combined with CAP7)
1 = Comment Text (bit mask to be combined with CAP7)
```

CAP2 - Schematic Capture Polygon Type:

```
0 = Line Cosmetic
1 = Filled Cosmetic
2 = internal
3 = internal
4 = Connection Point
5 = Dotted Line
```

CAP3 - Schematic Capture Figure Element Type:

```
1 = Polygon
2 = Connection
3 = Named Reference
4 = internal
5 = Text
6 = Name pattern
7 = internal
8 = Polygon corner pick
9 = Named symbol reference pick
10 = Named label reference pick
11 = Named reference attribute pick
```

CAP4 - Schematic Capture Pool Element Type:

```
  -1 = Unknown/undefined Element
   1 = Element Type Macro (C_MACRO)
   3 = Element Type Named Reference (C_NREF)
   6 = Element Type Attribute Value (C_ATTRIBUTE.VALUE)
   7 = Element Type Attribute Name (C_ATTRIBUTE.NAME)
  16 = Element Type Polygon (C_POLY)
  17 = Element Type Text (C_TEXT)
  18 = Element Type Connection Segment List (C_CONBASE)
  19 = Element Type Bus Tap (C_BUSTAP)
  20 = Element Type Part Name Pattern (C_MACRO.PNAMEPAT)
  32 = Element Type Font Name
else = internal
```

CAP5 - Schematic Tag Symbol/Label Mode:

```
1 = Standard Symbol / Standard Label
2 = Virtual Tag Symbol
3 = Netlist Tag Symbol / Net Attribute Label
```

CAP6 - Schematic Capture Tag Pin Type:

```
0 = Standard Pin or Label
1 = Symbol Destination Tag Pin
2 = Pin Destination Tag Pin
3 = Net Destination Tag Pin
4 = Net Pin Destination Tag Pin
3 = Net Area Destination Tag Pin
```

CAP7 - Schematic Capture Text Style Bit Mask:

```
xxxx000x = Standard text style (no frame)
xxxxxx1x = Frame 1; surrounding box at 1/8 text height distance from text
xxxxx1xx = Frame 2; surrounding box at 1/4 text height distance from text
xxxx1xxx = Open frame(s); surrounding box(es) open at text origin side
xxx1xxxx = No text rotation
xx1xxxxx = Horizontally centered text flag
x1xxxxxx = Vertically centered text flag
1xxxxxxx = Right-aligned text flag
```

A.2.3 Schematic Editor Ranges (SCM)

The following codes are valid for caller type SCM. I.e., they define valid value ranges for special system function parameters in the **Schematic Editor** interpreter environment. The naming of the value range definitions emerges from the keyword **SCM** and a continuous number.

SCM1 - SCM Display Item Types:

```
(-16384) = -0x4000 = Invalid Display Item
       0 = Documentation
       1 = Connections
       2 = Symbols
       3 = Markers
       4 = Symbol Borders
       5 = internal
       6 = internal
       7 = Connect Area
       8 = Work Area
       9 = Origin
      10 = Highlight
      11 = Commentary Text
      12 = Tag Symbol
      13 = Tag Link
      14 = Variant Attribute
      15 = Plot disabled (elements which are excluded from plot outputs)
```

SCM2 - Schematic Display Item Class Level Bit Mask:

```
xxxxxxxxxxx1 = Display item on plan level
xxxxxxxxxx1x = Display item on symbol level
xxxxxxxxx1xx = Display item on label level
xxxxxxxx1xxx = Display item on marker level
```

A.2.4 Layout Value Ranges (LAY)

The following codes are valid for caller types LAY, GED, AR and CAM. I.e., they define valid value ranges for special elements of index variable types and/or system function parameters in the **Layout Editor**, the **Autorouter** and the **CAM Processor** interpreter environments. The naming of the value range definitions emerges from the keyword **LAY** and a continuous number.

LAY1 - Layout Layer Number:

```
(-16384) = -0x4000 = Invalid Layer
    (-6) = Inside Signal Layers
    (-5) = Part Side Layer
    (-4) = Unroutes Layer
    (-3) = Part Plan Layer
    (-2) = Board Outline Layer
    (-1) = All Signal Layers
       0 = Signal Layer 1
       1 = Signal Layer 2
       2 = Signal Layer 3
       : = Signal Layer :
      99 = Signal Layer 100
     768 = 0x300 = Power Layer 1
     769 = 0x301 = Power Layer 2
     770 = 0x302 = Power Layer 3
       : =     : = Power Layer :
     777 = 0x309 = Power Layer 10
     778 = 0x30A = Power Layer 11
     779 = 0x30B = Power Layer 12
    1024 = 0x400 = Documentary Layer 1 Side 1
    1025 = 0x401 = Documentary Layer 1 Side 2
    1026 = 0x402 = Documentary Layer 1 Both Sides
    1040 = 0x410 = Documentary Layer 2 Side 1
    1041 = 0x411 = Documentary Layer 2 Side 2
    1042 = 0x412 = Documentary Layer 2 Both Sides
       : =     : = Documentary Layer :
    1168 = 0x490 = Documentary Layer 10 Side 1
    1169 = 0x491 = Documentary Layer 10 Side 2
    1170 = 0x492 = Documentary Layer 10 Both Sides
    1184 = 0x4A0 = Documentary Layer 11 Side 1
    1185 = 0x4A1 = Documentary Layer 11 Side 2
    1186 = 0x4A2 = Documentary Layer 11 Both Sides
    1200 = 0x4B0 = Documentary Layer 12 Side 1
    1201 = 0x4B1 = Documentary Layer 12 Side 2
    1202 = 0x4B2 = Documentary Layer 12 Both Sides
       : =     : = Documentary Layer :
    2592 = 0xA20 = Documentary Layer 99 Side 1
    2593 = 0xA21 = Documentary Layer 99 Side 2
    2594 = 0xA22 = Documentary Layer 99 Both Sides
    2608 = 0xA30 = Documentary Layer 100 Side 1
    2609 = 0xA31 = Documentary Layer 100 Side 2
    2610 = 0xA32 = Documentary Layer 100 Both Sides
```

LAY2 - Layout Text Mode:

```
0 = Physical
1 = Logical
2 = Norotate
```

LAY3 - Layout Polygon Mirror Visibility:

```
0 = Visible always
1 = Visible if not mirrored
2 = Visible if mirrored
17 = Fixed visible if not mirrored
18 = Fixed visible if mirrored
```

LAY4 - Layout Polygon Type:

```
1 = Copper
2 = Forbidden Area
3 = Border
4 = Connected Copper
5 = Line Cosmetic
6 = Filled Cosmetic
7 = Copperfill Workarea
8 = Hatched Copper
9 = Split Power Plane Area
```

LAY5 - Layout Drilling Class:

```
 0 = - (standard default)
 1 = A (standard/unmirrored)
 2 = B (standard/unmirrored)
 3 = C (standard/unmirrored)
 4 = D (standard/unmirrored)
 5 = E (standard/unmirrored)
 6 = F (standard/unmirrored)
 7 = G (standard/unmirrored)
 8 = H (standard/unmirrored)
 9 = I (standard/unmirrored)
10 = J (standard/unmirrored)
11 = K (standard/unmirrored)
12 = L (standard/unmirrored)
13 = M (standard/unmirrored)
14 = N (standard/unmirrored)
15 = O (standard/unmirrored)
16 = P (standard/unmirrored)
17 = Q (standard/unmirrored)
18 = R (standard/unmirrored)
19 = S (standard/unmirrored)
20 = T (standard/unmirrored)
21 = U (standard/unmirrored)
22 = V (standard/unmirrored)
23 = W (standard/unmirrored)
24 = X (standard/unmirrored)
25 = Y (standard/unmirrored)
26 = Z (standard/unmirrored)
0x0080 =  0 × 256 + 128 = - (mirrored default)
0x0180 =  1 × 256 + 128 = A (mirrored)
0x0280 =  2 × 256 + 128 = B (mirrored)
0x0380 =  3 × 256 + 128 = C (mirrored)
0x0480 =  4 × 256 + 128 = D (mirrored)
0x0580 =  5 × 256 + 128 = E (mirrored)
0x0680 =  6 × 256 + 128 = F (mirrored)
0x0780 =  7 × 256 + 128 = G (mirrored)
0x0880 =  8 × 256 + 128 = H (mirrored)
0x0980 =  9 × 256 + 128 = I (mirrored)
0x0A80 = 10 × 256 + 128 = J (mirrored)
0x0B80 = 11 × 256 + 128 = K (mirrored)
0x0C80 = 12 × 256 + 128 = L (mirrored)
0x0D80 = 13 × 256 + 128 = M (mirrored)
0x0E80 = 14 × 256 + 128 = N (mirrored)
0x0F80 = 15 × 256 + 128 = O (mirrored)
0x1080 = 16 × 256 + 128 = P (mirrored)
0x1180 = 17 × 256 + 128 = Q (mirrored)
0x1280 = 18 × 256 + 128 = R (mirrored)
0x1380 = 19 × 256 + 128 = S (mirrored)
0x1480 = 20 × 256 + 128 = T (mirrored)
0x1580 = 21 × 256 + 128 = U (mirrored)
0x1680 = 22 × 256 + 128 = V (mirrored)
0x1780 = 23 × 256 + 128 = W (mirrored)
0x1880 = 24 × 256 + 128 = X (mirrored)
0x1980 = 25 × 256 + 128 = Y (mirrored)
0x1A80 = 26 × 256 + 128 = Z (mirrored)
```

Mirrored drill classes are intended for mirrored blind and buried via padstack definitions. Standard and mirrored drill class codes can be combined into a single drill class specification by adding or bit-or-ing the desired drill classes as in

```
6 + 0x0880
```

for standard drill class **F** (6) and mirrored drill class **H** (0x0880).

LAY6 - Layout Figure Element Type:

```
1 = Polygon
2 = Path
3 = Named Reference
4 = Unnamed Reference
5 = Text
6 = Drill
7 = internal
8 = Polygon corner pick
9 = Trace corner/pick
10 = Fill area polygon pick
```

LAY7 - Layout Level Type:

```
>= 0 = Single Tree Level
(-1) = Multiple Trees Level (Short Circuit)
(-2) = Changed Level, no Tree
(-3) = Assigned Level (internal)
```

LAY8 - Layout Pool Element Type:

```
  -1 = Unknown/undefined Element
   1 = Element Type Macro (L_MACRO)
   5 = Element Type Unnamed Reference (L_UREF)
   6 = Element Type Named Reference (L_NREF)
   8 = Element Type Attribute Value (L_ATTRIBUTE.VALUE)
   9 = Element Type Attribute Name (L_ATTRIBUTE.NAME)
  16 = Element Type Polygon (L_POLY)
  17 = Element Type Path (L_LINE)
  18 = Element Type Text (L_TEXT)
  19 = Element Type Drill (L_DRILL)
  21 = Element Type Hatched Polygon Path
  32 = Element Type Part Side Layer
  33 = Element Type Power Layer Net
  34 = Element Type DRC Parameter
  35 = Element Type Font Name
  48 = Element Type DRC Error Marker (L_DRCERROR)
else = internal
```

LAY9 - Layout Display Item Types (additional to LAY1):

```
 (-7) = Drill Holes (transfered to classes '-', 'A'-'Z')
 (-8) = Work Area
 (-9) = Origin
(-10) = Error
(-11) = Highlight
(-12) = Drill Holes Class '-'
(-13) = Drill Holes Class 'A'
    : = :
(-38) = Drill Holes Class 'Z'
(-39) = Fixed
(-40) = Glued
```

LAY10 - Layout Mincon Function Type:

```
0 = Mincon Off
1 = Pins Horizontal
2 = Pins Vertical
3 = Pins Horizontal+Vertical Sum
4 = Pins Airline
5 = Corners Horizontal
6 = Corners Vertical
7 = Corners Horizontal+Vertical Sum
8 = Corners Airline
```

LAY11 - Layout Input Item Type:

```
>= 0 = Pool element input
(-1) = Rubberband input
(-2) = Window input
(-3) = Circle center input
(-4) = Anti clockwise arc input
(-5) = Clockwise arc input
(-6) = Segment move input
(-7) = Segment cut input
(-8) = Segment marker input
(-9) = Rubberband input type 2
```

LAY12 - Layout Variant Visibility:

```
0-99 = Visible for given variant number
100  = Visible for all variants
101  = Visible for unplaced variant
```

LAY13 - Layout DRC Error Display Mode:

```
1 = DRC copper distance violation
2 = DRC documentary layer forbidden area violation
3 = DRC documentary layer forbidden area height violation
4 = DRC HF design rule violation
5 = DRC invalid dropped polygon range
7 = DRC part side violation
8 = Fill polygon error hint
|65536 = DRC error marked as hidden
```

LAY14 - Layout Text Style Bit Mask:

```
xxx000xxxxxx = Standard text style (no frame)
xxxxx1xxxxxx = Frame 1; surrounding box at 1/8 text height distance from text
xxxx1xxxxxxx = Frame 2; surrounding box at 1/4 text height distance from text
xxx1xxxxxxxx = Open frame(s); surrounding box(es) open at text origin side
xx1xxxxxxxxx = Horizontally centered text flag
x1xxxxxxxxxx = Vertically centered text flag
1xxxxxxxxxxx = Right-aligned text flag
```

LAY15 - Layout Display Item Class Level Bit Mask:

```
xxxxxxxxxxx1 = Display item on layout level
xxxxxxxxxx1x = Display item on part level
xxxxxxxxx1xx = Display item on padstack level
xxxxxxxx1xxx = Display item on pad level
```

A.2.5 CAM Processor Value Ranges (CAM)

The following codes are valid for caller type CAM. I.e., they define valid value ranges for special system function parameters in the **CAM Processor** interpreter environment. The naming of the value range definitions emerges from the keyword **CAM** and a continuous number.

CAM1 - CAM Processor Mirror Mode:

```
0 = Mirroring off
1 = Mirroring on
2 = X-Backside (Mirroring off)
3 = X-Backside (Mirroring on)
4 = Y-Backside (Mirroring off)
5 = Y-Backside (Mirroring on)
```

CAM2 - CAM Processor Gerber Output Plotter Unit Length (STD2):

```
0.0000254     = 2.3 Inch Format
0.00000254    = 2.4 Inch Format
0.000000254   = 2.5 Inch Format
0.0000000254  = 2.6 Inch Format
0.00000000254 = 2.7 Inch Format
or any other value greater than 0.00000000053
```

CAM3 - CAM Processor Gerber Output Format:

```
    0 = 2.3 Inch Format
    1 = 2.4 Inch Format
    2 = 2.5 Inch Format
    3 = 2.6 Inch Format
    4 = 3.3 Metric Format
    5 = 3.4 Metric Format
    6 = 3.5 Metric Format
    7 = 3.6 Metric Format
```

CAM4 - CAM Processor HP-GL Plot Pen Number:

```
    1 = Enabled Pen 1
    2 = Enabled Pen 2
    3 = Enabled Pen 3
    4 = Enabled Pen 4
    5 = Enabled Pen 5
    6 = Enabled Pen 6
    7 = Enabled Pen 7
    8 = Enabled Pen 8
    9 = Enabled Pen 9
   10 = Enabled Pen 10
    : = Enabled Pen :
  100 = Enabled Pen 100
 (-1) = Disabled/Invalid Pen Number
 (-2) = Disabled Pen 1
 (-3) = Disabled Pen 2
 (-4) = Disabled Pen 3
 (-5) = Disabled Pen 4
 (-6) = Disabled Pen 5
 (-7) = Disabled Pen 6
 (-8) = Disabled Pen 7
 (-9) = Disabled Pen 8
(-10) = Disabled Pen 9
(-11) = Disabled Pen 10
   :  = Disabled Pen :
(-101) = Disabled Pen 100
```

A.2.6 IC Design Value Ranges (ICD)

The following codes are valid for caller types ICD and CED. I.e., they define valid value ranges for special elements of index variable types and/or system function parameters in the **Chip Editor** interpreter environment. The naming of the value range definitions emerges from the keyword **ICD** and a continuous number.

ICD1 - IC Design Layer Number:

```
(-16384) = -0x4000 = Invalid Layer
    (-3) = Unroutes Layer
    (-2) = Border Layer
    (-1) = All Layers
       0 = IC Layer 1
       1 = IC Layer 2
       2 = IC Layer 3
       : = IC Layer :
      99 = IC Layer 100
```

ICD2 - IC Design Text Mode:

```
0 = Physical
1 = Logical
2 = Norotate
```

ICD3 - IC Design Polygon Mirror Visibility:

```
0 = Visible always
1 = Visible if not mirrored
2 = Visible if mirrored
```

ICD4 - IC Design Polygon Type:

```
1 = Active Area
2 = Forbidden Area
3 = Line Cosmetic
4 = Border
```

ICD5 - IC Design Figure List Element Type:

```
1 = Polygon
2 = Path
3 = Named Reference
4 = Unnamed Reference
5 = Text
6 = internal
7 = Polygon corner pick
8 = Trace corner/pick
```

ICD6 - IC Design Level Type:

```
>= 0 = Single Tree Level
(-1) = Multiple Trees Level (Short Circuit)
(-2) = Changed Level, no Tree
(-3) = Assigned Level (internal)
```

ICD7 - IC Design Pool Element Type:

```
  -1 = Unknown/undefined Element
   1 = Element Type Macro (I_MACRO)
   2 = Element Type Unnamed Reference (I_UREF)
   3 = Element Type Named Reference (I_NREF)
   6 = Element Type Attribute Value (I_ATTRIBUTE.VALUE)
   7 = Element Type Attribute Name (I_ATTRIBUTE.NAME)
  16 = Element Type Polygon (I_POLY)
  17 = Element Type Path (I_LINE)
  18 = Element Type Text (I_TEXT)
else = internal
```

ICD8 - IC Design Display Item Types (additional to ICD1):

```
(-6) = Work Area
(-7) = Origin
(-8) = Error
(-9) = Highlight
```

ICD9 - IC Design Layer Display Mode Bit Mask:

```
xxxxx00 = Display layer items with outline
xxxxx01 = Display layer items with filled area
xxxxx10 = Display layer items with dash outline
xxxxx11 = Display layer items with pattern filled area
00000xx = Display layer items with pattern 0 filled area
.....xx = Display layer items with pattern : filled area
11111xx = Display layer items with pattern 31 filled area
```

ICD10 - IC Design Mincon Function Type:

```
0 = Mincon Off
1 = Pins Horizontal
2 = Pins Vertical
3 = Pins Horizontal+Vertical Sum
4 = Pins Airline
5 = Corners Horizontal
6 = Corners Vertical
7 = Corners Horizontal+Vertical Sum
8 = Corners Airline
```

Appendix B
Index Variable Types

This appendix describes the **Bartels User Language** index variable types definitions, providing alphabetically sorted reference lists and index type descriptions which are grouped according to the corresponding caller types.

Contents

B.1 Index Reference

Each **Bartels User Language** index variable type is assigned to one of the caller types STD, CAP, LAY or ICD. This section lists the index variable types for each caller type.

B.1.1 Standard Index Variable Types (STD)

The following index variable types are assigned to caller type STD; i.e., they can be accessed from any of the **Bartels AutoEngineer** interpreter environments:

BAEPARAM	Bartels AutoEngineer Parameter
GLOBALVAR	Global User Language Variable

B.1.2 Schematic Capture Index Variable Types (CAP)

The following index variable types are assigned to caller type CAP; i.e., they can be accessed from the **Schematic Editor** interpreter environment:

C_ATTRIBUTE	SCM Part Attribute
C_BUSTAP	SCM Bus Tap
C_CNET	SCM Logical Net List
C_CONBASE	SCM Connection Segment Group
C_CONSEG	SCM Connection Segment
C_FIGURE	SCM Figure Element
C_LEVEL	SCM Signal Level
C_MACRO	SCM Library Element
C_NREF	SCM Named Macro Reference
C_POINT	SCM Polygon Point
C_POLY	SCM Polygon
C_POOL	SCM Pool Element
C_TEXT	SCM Text
CL_ALTPLNAME	Layout Net List Alternate Part Package
CL_ATTRIBUTE	Layout Net List Attribute
CL_CNET	Layout Net List
CL_CPART	Layout Net List Part Entry
CL_CPIN	Layout Net List Part Pin Entry

B.1.3 Layout Index Variable Types (LAY)

The following index variable types are assigned to caller type LAY; i.e., they can be accessed from the **Layout Editor**, the **Autorouter** and the **CAM Processor** interpreter environments:

L_ALTPLNAME	Layout Net List Alternate Part Package Type
L_ATTRIBUTE	Layout Net List Attribute
L_CNET	Layout Net List
L_CPART	Layout Net List Part Entry
L_CPIN	Layout Net List Part Pin Entry
L_DRCERROR	Layout DRC Error Marker
L_DRCERROROK	Layout DRC Error Acceptance
L_DRILL	Layout Drill Hole
L_FIGURE	Layout Figure Element
L_LEVEL	Layout Signal Level
L_LINE	Layout Trace
L_MACRO	Layout Library Element
L_NREF	Layout Named Macro Reference
L_POINT	Layout Polygon Point
L_POLY	Layout Polygon
L_POOL	Layout Pool Element
L_POWLAYER	Layout Power Layer
L_TEXT	Layout Text
L_UREF	Layout Unnamed Macro Reference

B.1.4 CAM View Index Variable Types (CV)

The following index variable types are assigned to caller type CV; i.e., they can be accessed from the **CAM View** interpreter environment:

CV_DATASET	CAM View Data Set

B.1.5 IC Design Index Variable Types (ICD)

The following index variable types are assigned to caller type ICD; i.e., they can be accessed from the **Chip Editor** interpreter environment:

I_ATTRIBUTE	IC Design Net List Attribute
I_CNET	IC Design Net List
I_CPART	IC Design Net List Part Entry
I_CPIN	IC Design Net List Part Pin Entry
I_FIGURE	IC Design Figure Element
I_LEVEL	IC Design Signal Level
I_LINE	IC Design Trace
I_MACRO	IC Design Library Element
I_NREF	IC Design Named Macro Reference
I_POINT	IC Design Polygon Point
I_POLY	IC Design Polygon
I_POOL	IC Design Pool Element
I_TEXT	IC Design Text
I_UREF	IC Design Unnamed Macro Reference

B.2 Standard Index Description (STD)

This section describes the **Bartels User Language** index variable types for general **User Language** data access (STD).

BAEPARAM - Bartels AutoEngineer Parameter

The **BAEPARAM** index variable type provides access to the list of the currently activated BAE parameters. The structure definition of **BAEPARAM** is:

```
index BAEPARAM {              // BAE Parameter Index
      int IDCODE;             // Parameter Type/Ident. Code
      int TYP;                // Parameter Data Type:
                              //    1 = int
                              //    2 = double
                              //    4 = string
      int VALINT;             // Parameter Integer Value (if TYP is 1)
      double VALDBL;          // Parameter Double Value (if TYP is 2)
      string VALSTR;          // Parameter String Value (if TYP is 4)
      };
```

The source code of the **baeparam.ulh User Language** include file provides the list of valid parameter type codes (i.e., valid values for **IDCODE**) as well as function templates for retrieving specific BAE parameter values.

GLOBALVAR - Global User Language Variable

The **GLOBALVAR** index variable type provides access to the list of the global **User Language** values currently defined with the **varset User Language** system function. The structure definition of **GLOBALVAR** is:

```
index GLOBALVAR {             // Global User Language Variable Index
      string NAME;            // Global Variable Name
      int TYP;                // Global Variable Data Type:
                              //    1 = int
                              //    2 = double
                              //    3 = char
                              //    4 = string
      };
```

Once a global **User Language** variable has been scanned with the **GLOBALVAR** index, the **varget User Language** system function can be used to retrieve its value.

B.3 Schematic Capture Index Description (CAP)

This section describes the **Bartels User Language** index variable types for the Schematic Capture data access (CAP).

C_ATTRIBUTE - SCM Part Attribute

The **C_ATTRIBUTE** index variable type provides access to the part attributes defined on the currently loaded SCM sheet. The structure definition of **C_ATTRIBUTE** is:

```
index C_ATTRIBUTE {           // Attribute Index
      string NAME;            // Attribute Name
      string VALUE;           // Attribute Value
      };
```

The **C_ATTRIBUTE** index can only be applied as **of**-index for the attribute list of **C_NREF**.

C_BUSTAP - SCM Bus Tap

The **C_BUSTAP** index variable type provides access to the bus connectors (bus taps) of the currently loaded SCM sheet. The structure definition of **C_BUSTAP** is:

```
index C_BUSTAP {              // Bustap Index
      string NAME;            // Bustap Name
      double X;               // Bustap X Coordinate (STD2)
      double Y;               // Bustap Y Coordinate (STD2)
      double ANGLE;           // Bustap Rotation Angle (STD3)
      int MIRROR;             // Bustap Mirror Mode (STD14)
      index C_MACRO MACRO;    // Link to Bus Tap Macro
      index C_CONBASE CON;    // Link to Segment Group
      index C_CONSEG SEG;     // Link to Connection Segment
      };
```

Each bus tap is placed on a bus connection segment. The bus tap rotation angle is a multiple of 90 degree. The **CON** index variable is a backward link to the connection segment group containing the bus tap. The **SEG** index variable provides a cross link to the connection segment on which the bus tap is placed.

C_CNET - SCM Logical Net List

The **C_CNET** index variable type provides access to the logical net list of the currently loaded SCM sheet. The structure definition of **C_CNET** is:

```
index C_CNET {                // Logical Net List Index
      string NAME;            // Net Name
      int NUMBER;             // Net Tree Number
      int NETGLO;             // Net Global Flag:
                              //    0 = Local Net Definition
                              //    1 = Global Net Definition
      int BUSNET;             // Bus Net Number
      };
```

C_CONBASE - SCM Connection Segment Group

The **C_CONBASE** index variable type provides access to the connection segment group of a specific column and/or row of the currently loaded SCM sheet. The structure definition of **C_CONBASE** is:

```
index C_CONBASE {             // Connection Segment Group Index
      int ORI;                // Orientation:
                              //    0 = Horizontal
                              //    1 = Vertical
      double X;               // Base X Coordinate (STD2)
      double Y;               // Base Y Coordinate (STD2)
      int SN;                 // Connection Segment Count
      index C_CONSEG;         // Connection Segment List
      index C_BUSTAP;         // Bus Tap List
      };
```

A **C_CONBASE** index contains all connection segments of a specific column (orientation vertical) and/or row (orientation horizontal), including all bus taps defined on these connections. The group's connection segment and bus tap lists can be accessed through corresponding **forall-of** loops.

C_CONSEG - SCM Connection Segment

The **C_CONSEG** index variable type provides access to the connection segments of the currently loaded SCM sheet. The structure definition of **C_CONSEG** is:

```
index C_CONSEG {              // Connection Segment Index
      double X1;              // Segment X Coordinate 1 (STD2)
      double Y1;              // Segment Y Coordinate 1 (STD2)
      double X2;              // Segment X Coordinate 2 (STD2)
      double Y2;              // Segment Y Coordinate 2 (STD2)
      int BUSFLAG;            // Segment Bus Flag:
                              //    0 = Normal Segment
                              //    1 = Bus Segment
      int GROUP;              // Segment Group Flag (STD13)
      index C_CONBASE CON;    // Link to Segment Group
      };
```

Connection segments always are placed orthogonal. This means that either the X coordinates are identical (orientation vertical) or, otherwise, the Y coordinates are identical (orientation horizontal). The **CON** index variable provides a backward link to the connection segment group which contains the corresponding connection segment.

C_FIGURE - SCM Figure Element

The **C_FIGURE** index variable type provides access to all placed figure elements (polygons, connections, macro references, texts) of the currently loaded SCM element. The structure definition of **C_FIGURE** is:

```
index C_FIGURE {              // Figure Element Index
      int TYP;                // Element Type (CAP3)
      string NAME;            // Element Name
      double SIZE;            // Element Size (STD2)
      double X;               // Element X Coordinate (STD2)
      double Y;               // Element Y Coordinate (STD2)
      double ANGLE;           // Element Rotation Angle (STD3)
      int MIRROR;             // Element Mirror Mode (STD14)
      int GROUP;              // Element Group Flag (STD13)
      index C_POOL POOL;      // Link to Pool Element
      index C_POLY POLY;      // Link to Polygon Element
      index C_CONBASE CONBASE;// Link to Connection Segment Group
      index C_NREF NREF;      // Link to Named Reference Element
      index C_TEXT TEXT;      // Link to Text Element
      };
```

The **NAME** variable either denotes the name for named macro references or denotes the string of a text element. On SCM symbol level, **NAME** holds the SCM symbol part name pattern. The **POOL** variable provides a cross link to the library pool element which builds up the figure element. The figure element attributes can be changed with the **scm_elem*chg** functions. A feature for scanning the complete figure element data with all hierarchy levels is provided with the **cap_scanfelem** function.

C_LEVEL - SCM Signal Level

The **C_LEVEL** index variable type provides access to the connectivity levels, i.e., the net list and/or signal levels of the currently loaded SCM sheet. The structure definition of **C_LEVEL** is:

```
index C_LEVEL {               // Connectivity Level Index
      int IDNUM;              // Level Identification Number
      int BUSFLAG;            // Level Bus Flag
      int SEGFLAG;            // Level Segment Connection Mode (Bit Pattern):
                              //    1 = Segment connected to level
                              //    2 = Contact areas connected to level
      int ERRFLAG;            // Level Error Flag
      int HIGHLIGHT;          // Level Highlight Flag
      int DISPLAY;            // Level Display Attributes
      int CNN;                // Level Net Count
      index C_CNET;           // Level Net List
      };
```

The signal level's net list can be accessed through a corresponding **forall-of** loop.

C_MACRO - SCM Library Element

The **C_MACRO** index variable type provides access to the macros, i.e., the library elements (symbol, label, marker) used on the currently loaded SCM element. The structure definition of **C_MACRO** is:

```
index C_MACRO {               // Macro Definition Index
      string NAME;            // Macro Name
      double MLX;             // Left Macro Border (STD2)
      double MLY;             // Lower Macro Border (STD2)
      double MUX;             // Right Macro Border (STD2)
      double MUY;             // Upper Macro Border (STD2)
      double MNX;             // Macro Origin X Coordinate (STD2)
      double MNY;             // Macro Origin Y Coordinate (STD2)
      int CLASS;              // Macro Class Code (STD1)
      int TAGSYM;             // Macro Tag Symbol/Label Mode (CAP5)
      int COMP;               // Macro Status (STD16)
      string PNAMEPAT;        // Macro Part Name Pattern
      };
```

The **PNAMEPAT** variable holds the part name pattern defined for SCM symbol macros.

C_NREF - SCM Named Macro Reference

The **C_NREF** index variable type provides access to the named macro references, i.e., the name-specified library elements placed on the currently loaded SCM element. These are parts and/or labels on SCM sheet level or pins on symbol and/or label hierarchy level. The structure definition of **C_NREF** is:

```
index C_NREF {                // Named Reference Index
      string NAME;            // Reference Name
      double X;               // Reference X Coordinate (STD2)
      double Y;               // Reference Y Coordinate (STD2)
      double ANGLE;           // Reference Rotation Angle (STD3)
      int MIRROR;             // Reference Mirror Mode (STD14)
      int TAGPTYP;            // Reference Tag Pin Type (CAP6)
      index C_MACRO MACRO;    // Link to Macro
      index C_ATTRIBUTE;      // Attribute List
      };
```

The **MACRO** variable provides a cross link for accessing the referenced library element. The part attribute list can be accessed through a corresponding **forall-of** loop.

C_POINT - SCM Polygon Point

The **C_POINT** index variable type provides access to the polygon points of a specific polygon. The structure definition of **C_POINT** is:

```
index C_POINT {               // Polygon Point Index
      double X;               // Polygon Point X Coordinate (STD2)
      double Y;               // Polygon Point Y Coordinate (STD2)
      int TYP;                // Polygon Point Type (STD15)
      };
```

The **C_POINT** index can only be applied as **of**-index for the point list in **C_POLY**.

C_POLY - SCM Polygon

The **C_POLY** index variable type provides access to the polygons (areas, lines) defined on the currently loaded SCM element. The structure definition of **C_POLY** is:

```
index C_POLY {                  // Polygon Index
      int TYP;                  // Polygon Type (CAP2)
      double WIDTH;             // Polygon Line Width (STD2)
      double DASHLEN;           // Polygon Dash Length (STD2)
      double DASHSPC;           // Polygon Dash Relative Spacing
      int DASH;                 // Polygon Dash Mode
      int PN;                   // Polygon Point Count
      index C_POINT;            // Polygon Point List
      };
```

The polygon point list of the polygon can be accessed through a corresponding **forall-of** loop.

C_POOL - SCM Pool Element

The **C_POOL** index variable type provides access to the currently loaded pool elements. The structure definition of **C_POOL** is:

```
index C_POOL {                  // Pool Element Index
      int TYP;                  // Pool Element Typ (CAP4)
      int REFCNT;               // Pool Element Reference Count
      index C_POOL NXT;         // Link to Next Pool Element
      index C_POOL REF;         // Link to Reference Pool Element
      index C_POLY POLY;        // Link to Polygon Element
      index C_CONBASE CONBASE;// Link to Connection Segment Group
      index C_NREF NREF;        // Link to Named Reference Element
      index C_TEXT TEXT;        // Link to Text Element
      index C_MACRO MACRO;      // Link to Library Element
      index C_BUSTAP BUSTAP;    // Link to Bustap Element
      };
```

The **C_POOL** index is used for processing library definitions with the **cap_scanpool** system function. The **REFCNT** variable specifies, how often the pool element is currently referenced. The **NXT** and **REF** variables allow for fast pool element list traversal.

C_TEXT - SCM Text

The **C_TEXT** index variable type provides access to the text data defined on the currently loaded SCM element. The structure definition of **C_TEXT** is:

```
index C_TEXT {                  // Text Index
      string STR;               // Text String
      double X;                 // Text X Coordinate (STD2)
      double Y;                 // Text Y Coordinate (STD2)
      double ANGLE;             // Text Rotation Angle (STD3)
      double SIZE;              // Text Size (STD2)
      double WIDTH;             // Text Line Width (STD2)
      int MIRROR;               // Text Mirror Mode (STD14)
      int MODE;                 // Text Mode/Style (CAP1|CAP7)
      int CLASS;                // Text Class Bits
      };
```

CL_ALTPLNAME - Layout Net List Alternate Part Package Type

The **CL_ALTPLNAME** index variable type provides access to the alternate part package type list of the currently loaded layout net list. The structure definition of **CL_ALTPLNAME** is:

```
index L_ALTPLNAME {          // Alternate Part Package Type Index
      string PLNAME;          // Layout Library Name
      };
```

CL_ATTRIBUTE - Layout Net List Attribute

The **CL_ATTRIBUTE** index variable type provides access to the part and net attributes of the currently loaded layout net list. The structure definition of **CL_ATTRIBUTE** is:

```
index CL_ATTRIBUTE {          // Attribute Index
      string NAME;            // Attribute Name
      string VALUE;           // Attribute Value
      };
```

CL_CNET - Layout Net List

The **CL_CNET** index variable type provides access to the nets of the currently loaded layout net list. The structure definition of **CL_CNET** is:

```
index CL_CNET {               // Layout Net Index
      string NAME;            // Net Name
      int NUMBER;             // Net Tree Number
      int PRIOR;              // Net Routing Priority
      double RDIST;           // Net Minimum Distance (STD2)
      int PINN;               // Net Pin Count
      index CL_CPIN;          // Net Pin List
      index CL_ATTRIBUTE;     // Net Attribute List
      };
```

The net tree number is used for identifying the net. The **cap_getlaytreeidx** function provides access to the **CL_CNET** index for a given net tree number. The minimum distance applies to the traces of the net; this distance must at minimum be kept to copper structures not belonging to the corresponding net. The net pin and/or attribute lists can be accessed through corresponding **forall-of** loops.

CL_CPART - Layout Net List Part Entry

The **CL_CPART** index variable type provides access to the parts of the currently loaded layout net list. The structure definition of **CL_CPART** is:

```
index CL_CPART {              // Layout Netlist Part Index
      string NAME;            // Part Name
      string PLNAME;          // Part Physical Library Name
      int PEQUC;              // Part Equivalence Code
      int PINN;               // Part Pin Count
      int FPINN;              // Part Free Pin Count
      index CL_CPIN;          // Part Pin List
      index CL_ALTPLNAME;     // Part Alternate Physical Names List
      index CL_ATTRIBUTE;     // Part Attribute List
      };
```

The part pin and/or attribute lists can be accessed through corresponding **forall-of** loops. Component swaps can be applied on parts with identical equivalence codes to optimize the placement.

CL_CPIN - Layout Net List Part Pin Entry

The **CL_CPIN** index variable type provides access to the part pins of the currently loaded layout net list. The structure definition of **CL_CPIN** is:

```
index CL_CPIN {               // Layout Netlist Part Pin Index
      string NAME;            // Pin Name
      double RWIDTH;          // Pin Routing Width (STD2)
      int TREE;               // Pin Net Tree Number
      int GATE;               // Pin Gate Number
      int GEQUC;              // Pin Gate Equivalence Code
      int GEQUP;              // Pin Equivalence Code
      int GGRPC;              // Pin Gate Group Number
      int GPNUM;              // Pin Gate Relative Number
      index CL_CNET CNET;     // Link to Pin Net
      index CL_CPART CPART;   // Link to Pin Part
      index CL_ATTRIBUTE;     // Pin Attribute List
      };
```

The pin routing width defines the width for routing to the next connection point. The **CNET** and **CPART** variables provide backward links to the corresponding layout net list nets and parts, respectively. The **GATE**, **GEQUC**, **GEQUP**, **GGRPC** and **GPNUM** variables can be used to check pin/gate swap allowance.

B.4 Layout Index Description (LAY)

This section describes the **Bartels User Language** index variable types for the Layout data access (LAY).

L_ALTPLNAME - Layout Net List Alternate Part Package Type

The **L_ALTPLNAME** index variable type provides access to the connection list part alternate physical library name list of the currently loaded layout. The structure definition of **L_ALTPLNAME** is:

```
index L_ALTPLNAME {           // Alternate Phys. Lib. Name Index
      string PLNAME;          // Physical Library Name
      };
```

L_ATTRIBUTE - Layout Net List Attribute

The **L_ATTRIBUTE** index variable type provides access to the connection list part or net attributes of the currently loaded layout. The structure definition of **L_ATTRIBUTE** is:

```
index L_ATTRIBUTE {           // Attribute Index
      string NAME;            // Attribute Name
      string VALUE;           // Attribute Value
      };
```

L_CNET - Layout Net List

The **L_CNET** index variable type provides access to the connection list nets of the currently loaded layout. The structure definition of **L_CNET** is:

```
index L_CNET {                // Connection List Net Index
      string NAME;            // Net Name
      int NUMBER;             // Net Tree Number
      int PRIOR;              // Net Routing Priority
      double RDIST;           // Net Minimum Distance (STD2)
      int VIS;                // Net Visibility Flag
      int PINN;               // Net Pin Count
      index L_CPIN;           // Net Pin List
      index L_ATTRIBUTE;      // Net Attribute List
      index L_POOL UNRPOOL;   // Link to Unroutes Pool Element
      };
```

The net tree number is used for identifying the net. The **lay_gettreeidx** function provides access to the **L_CNET** index for a given net tree number. The minimum distance applies to the traces of the net; this distance must at minimum be kept to copper structures not belonging to the corresponding net. The net pin and/or attribute list can be accessed by applying a corresponding **forall-of** loop. The **UNRPOOL** variable provides access to the unrouted connections of the net; the corresponding airlines can be processed with the **lay_scanpool** system function.

L_CPART - Layout Net List Part Entry

The **L_CPART** index variable type provides access to the connection list parts of the currently loaded layout. The structure definition of **L_CPART** is:

```
index L_CPART {               // Connection List Part Index
      string NAME;            // Part Name
      string PLNAME;          // Part Physical Library Name
      int USED;               // Part Placement (and Group Selection) Code:
                              //    0 = Part not placed
                              //    1 = Part placed
                              //    2 = Part placed and selected to group
      int PEQUC;              // Part Equivalence Code
      int PINN;               // Part Pin Count
      int FPINN;              // Part Free Pin Count
      index L_MACRO MACRO;    // Link to Macro
      index L_CPIN;           // Part Pin List
      index L_ALTPLNAME;      // Part Alternate Physical Names List
      index L_ATTRIBUTE;      // Part Attribute List
      };
```

The part pin and/or attribute lists can be accessed through corresponding **forall-of** loops. Component swaps can be applied on parts with identical equivalence codes to optimize the placement.

L_CPIN - Layout Net List Part Pin Entry

The **L_CPIN** index variable type provides access to the connection list part pins of the currently loaded layout. The structure definition of **L_CPIN** is:

```
index L_CPIN {                // Net List Pin Index
      string NAME;            // Pin Name
      double RWIDTH;          // Pin Routing Width (STD2)
      int TREE;               // Pin Net Tree Number
      int GATE;               // Pin Gate Number
      int GEQUC;              // Pin Gate Equivalence Code
      int GEQUP;              // Pin Equivalence Code
      int GGRPC;              // Pin Gate Group Number
      int GPNUM;              // Pin Gate Relative Number
      index L_CNET CNET;      // Link to Pin Net
      index L_CPART CPART;    // Link to Pin Part
      };
```

The pin routing width defines the width for routing to the next connection point. The **CNET** and **CPART** variables provide backward links to the corresponding connection list net and part entries, respectively. The **GATE**, **GEQUC**, **GEQUP**, **GGRPC** and **GPNUM** variables can be used for checking pin/gate swap allowance.

L_DRCERROR - Layout DRC Error Marker

The **L_DRCERROR** index variable type provides access to the error markers displayed by the design rule check on the currently loaded layout element. The structure definition of **L_DRCERROR** is:

```
index L_DRCERROR {            // DRC Error Marker Index
      int TYP;                // DRC Error Type:
                              //    1 = Copper distance violation
                              //    2 = Doc. layer keepout area violation
                              //    3 = Doc. layer keepout area height violation
                              //    4 = HF Design Rule Violation
                              //    5 = Invalid dropped polygon range
      int LAYER;              // DRC Error Marker Layer (LAY1)
      double RLX;             // DRC Error Marker Left Border (STD2)
      double RLY;             // DRC Error Marker Lower Border (STD2)
      double RUX;             // DRC Error Marker Right Border (STD2)
      double RUY;             // DRC Error Marker Upper Border (STD2)
      double CHKDIST;         // DRC Error DRC Clearance Distance (STD2)
      double ERRDIST;         // DRC Error Current Clearance Distance (STD2)
      string IDSTR;           // DRC Error Id String
      index L_FIGURE FIG1;    // DRC Error Element 1
      index L_FIGURE FIG2;    // DRC Error Element 2
      };
```

L_DRCERROROK - Layout DRC Error Acceptance

The **L_DRCERROROK** index variable type provides access to the DRC error acceptance settings of the currently loaded layout element. The structure definition of **L_DRCERROROK** is:

```
index L_DRCERROROK {          // Layout DRC Error Acceptance Index
      string IDSTR;           // DRC Error Id String
      };
```

L_DRILL - Layout Drill Hole

The **L_DRILL** index variable type provides access to the drill holes defined of the currently loaded padstack. The structure definition of **L_DRILL** is:

```
index L_DRILL {               // Drilling Index
      double X;               // Drilling X Coordinate (STD2)
      double Y;               // Drilling Y Coordinate (STD2)
      double RAD;             // Drilling Radius (STD2)
      int CLASS;              // Drilling Class Code (LAY5)
      };
```

The **L_DRILL** index contains the placement data of the drilling on the corresponding padstack, when using **L_DRILL** in the drill scan functions of **lay_scanfelem**, **lay_scanall** or **lay_scanpool**. The coordinates on the layout and/or part can be retrieved from the transformed coordinates of the drill scan function.

L_FIGURE - Layout Figure Element

The **L_FIGURE** index variable type provides access to all placed figure elements (polygons, traces, macro references, texts, drills) of the currently loaded layout element. The structure definition of **L_FIGURE** is:

```
index L_FIGURE {              // Figure Element Index
      int TYP;                // Element Type (LAY6)
      string NAME;            // Element Name
      double SIZE;            // Element Size (STD2)
      double X;               // Element X Coordinate (STD2)
      double Y;               // Element Y Coordinate (STD2)
      double ANGLE;           // Element Rotation Angle (STD3)
      int MIRROR;             // Element Mirror Mode (STD14)
      int LAYER;              // Element Layer/Class (LAY1 | LAY5)
      int GROUP;              // Element Group Flag (STD13)
      int FIXED;              // Element Fixed Flag (STD11 | STD12)
      int TREE;               // Element Net Tree Number
      index L_POOL POOL;      // Link to Pool Element
      index L_POLY POLY;      // Link to Polygon Element
      index L_LINE LINE;      // Link to Trace Element
      index L_NREF NREF;      // Link to Macro Reference (named)
      index L_UREF UREF;      // Link to Macro Reference (unnamed)
      index L_TEXT TEXT;      // Link to Text Element
      };
```

The **NAME** variable holds either the name of a named macro reference or the string of a text element. For elements of **TYP 7** (intern), **NAME** return the standard via padstack macro name if the internal element is a standard via definition. The **LAYER** variable specifies the element layer number, except for drill elements where it denotes the drill class. The **POOL** variable provides a cross link to the library pool element which builds up the figure element. The figure element attributes can be changed with the **ged_elem*chg** functions. A feature for scanning the complete figure element data with all hierarchy levels is provided with the **lay_scanfelem** function.

L_LEVEL - Layout Signal Level

The **L_LEVEL** index variable type provides access to the connectivity levels, i.e., the net list and/or signal levels of the currently loaded layout. The structure definition of **L_LEVEL** is:

```
index L_LEVEL {               // Connectivity Level Index
      int LEVVAL;             // Level Value (LAY7)
      };
```

L_LINE - Layout Trace

The **L_LINE** index variable type provides access to the path and/or trace data defined on the currently loaded layout and/or part. The structure definition of **L_LINE** is:

```
index L_LINE {                // Line Path Index
      double WIDTH;           // Line Path Width (STD2)
      int LAYER;              // Line Path Layer (LAY1)
      int TREE;               // Line Path Tree Number
      int PN;                 // Line Path Point Count
      index L_POINT;          // Line Path Point List
      };
```

L_MACRO - Layout Library Element

The **L_MACRO** index variable type provides access to the macros, i.e., the library elements (part, padstack, pad) used on the currently loaded layout element. The structure definition of **L_MACRO** is:

```
index L_MACRO {               // Macro Definition Index
      string NAME;            // Macro Name
      double MLX;             // Left Macro Border (STD2)
      double MLY;             // Lower Macro Border (STD2)
      double MUX;             // Right Macro Border (STD2)
      double MUY;             // Upper Macro Border (STD2)
      double MNX;             // Macro Origin X Coordinate (STD2)
      double MNY;             // Macro Origin Y Coordinate (STD2)
      int CLASS;              // Macro Class Code (STD1)
      int COMP;               // Macro Status (STD16)
      };
```

L_NREF - Layout Named Macro Reference

The **L_NREF** index variable type provides access to the named macro references, i.e., the name-specified library elements placed on the currently loaded element. These are parts on layout hierarchy level, or padstacks on part hierarchy level. The structure definition of **L_NREF** is:

```
index L_NREF {                // Named Reference Index
      string NAME;            // Reference Name
      double X;               // Reference X Coordinate (STD2)
      double Y;               // Reference Y Coordinate (STD2)
      double ANGLE;           // Reference Rotation Angle (STD3)
      int LAYOFF;             // Reference Layer Offset (LAY1)
      int MIRROR;             // Reference Mirror Flag (STD14)
      index L_MACRO MACRO;    // Link to Macro
      };
```

The **MACRO** variable provides a cross link for accessing the referenced library element.

L_POINT - Layout Polygon Point

The **L_POINT** index variable type provides access to the polygon points of a specific polygon or trace element. The structure definition of **L_POINT** is:

```
index L_POINT {               // Polygon Point Index
      double X;               // Polygon Point X Coordinate (STD2)
      double Y;               // Polygon Point Y Coordinate (STD2)
      int TYP;                // Polygon Point Type (STD15)
      };
```

The **L_POINT** index can be applied just as **of**-index for the point lists in **L_POLY** and/or **L_LINE**.

L_POLY - Layout Polygon

The **L_POLY** index variable type provides access to the polygons (passive copper, forbidden areas, board outline, active copper, documentary lines, documentary areas, copper fill workareas, hatched areas, split power plane areas) defined on the currently loaded layout element. The structure definition of **L_POLY** is:

```
index L_POLY {                  // Polygon Index
      int LAYER;                // Polygon Layer (LAY1)
      int TREE;                 // Polygon Net Tree Number
      int TYP;                  // Polygon Type (LAY4)
      int MVIS;                 // Polygon Mirror Mode (LAY3) and/or:
                                //    LAY3 + 4 = Dashed Polygon
                                //    LAY3 + 8 = Dotted Polygon
      double WIDTH;             // Polygon Line Width (STD2)
      double DASHLEN;           // Polygon Dash Length (STD2)
      double DASHSPC;           // Polygon Dash Relative Spacing
      int PN;                   // Polygon Point Count
      index L_POINT;            // Polygon Point List
      };
```

The polygon net tree number is only valid for active copper, copper fill workareas, hatched areas and split power plane areas. The **lay_scanfelem** and/or **lay_scanall** function must be used to check the signal level of passive copper.

L_POOL - Layout Pool Element

The **L_POOL** index variable type provides access to the currently loaded pool elements. The structure definition of **L_POOL** is:

```
index L_POOL {                  // Pool Element Index
      int TYP;                  // Pool Element Type (LAY8)
      int REFCNT;               // Pool Element Reference Count
      int LAYER;                // Pool Element Layer (LAY1)
      index L_POOL NXT;         // Link to Next Pool Element
      index L_POOL REF;         // Link to Reference Pool Element
      index L_POLY POLY;        // Link to Polygon Element
      index L_LINE LINE;        // Link to Trace Element
      index L_NREF NREF;        // Link to Macro Reference (named)
      index L_UREF UREF;        // Link to Macro Reference (unnamed)
      index L_TEXT TEXT;        // Link to Text Element
      index L_DRILL DRILL;      // Link to Drill Element
      index L_DRCERROR DRCERR;// Link to DRC Error Element
      index L_MACRO MACRO;      // Link to Library Element
      };
```

The **L_POOL** index is used for processing library definitions with the **lay_scanpool** system function. The **REFCNT** variable specifies, how often the pool element is currently referenced. The **NXT** and **REF** variables allow for fast pool element list traversal.

L_POWLAYER - Layout Power Layer

The index variable type **L_POWLAYER** provides access to the power layers defined on the currently loaded layout. The structure definition of **L_POWLAYER** is:

```
index L_POWLAYER {              // Power Layer Index
      index L_CNET CNET;        // Link to Power Layer Net
      index L_LEVEL LEVEL;      // Link to Power Layer Level
      int LAYER;                // Power Layer Code (LAY1)
      };
```

The **CNET** variable provides access to the net defined with the corresponding power layer. The **LEVEL** variable can be used to check the signal level of the corresponding power layer.

L_TEXT - Layout Text

The **L_TEXT** index variable type provides access to text data defined on the currently loaded layout element. The structure definition of **L_TEXT** is:

```
index L_TEXT {                  // Text Index
      string STR;               // Text String
      double X;                 // Text X Coordinate (STD2)
      double Y;                 // Text Y Coordinate (STD2)
      double ANGLE;             // Text Rotation Angle (STD3)
      double SIZE;              // Text Size (STD2)
      double WIDTH;             // Text Line Width (STD2)
      int LAYER;                // Text Layer (LAY1)
      int MIRROR;               // Text Mirror Flag (STD14)
      int MODE;                 // Text Mode (LAY2)
      };
```

L_UREF - Layout Unnamed Macro Reference

The **L_UREF** index variable type provides access to the unnamed macro references, i.e., the library elements placed on the currently loaded element without name-specification. These are vias on layout and/or part hierarchy level or pads on padstack hierarchy level. The structure definition of **L_UREF** is:

```
index L_UREF {                  // Unnamed Reference Index
      int TREE;                 // Reference Net Tree Number
      double X;                 // Reference X Coordinate (STD2)
      double Y;                 // Reference Y Coordinate (STD2)
      double ANGLE;             // Reference Rotation Angle (STD3)
      int LAYOFF;               // Reference Layer Offset (LAY1)
      int MIRROR;               // Reference Mirror Flag (STD14)
      index L_MACRO MACRO;      // Link to Macro
      };
```

The **MACRO** variable provides a cross link for accessing the referenced library element. The **LAYOFF** variable is only valid for pads on padstack hierarchy level.

B.5 CAM View Index Description (CV)

This section describes the **Bartels User Language** index variable types for the **CAM View** data access (CV).

CV_DATASET - CAM View Data Set

The **CV_DATASET** index variable type provides access to the currently loaded **CAM View** data sets. The structure definition of **CV_DATASET** is:

```
index CV_DATASET {            // Data Set
      int IDX;                // Data Set Index
      int TYP;                // Data Set Type
      int LLAYER;             // Data Set Line Layer (LAY1)
      int FLAYER;             // Data Set Flash Layer (LAY1)
      double XOFF;            // Data Set X Offset (STD2)
      double YOFF;            // Data Set Y Offset (STD2)
      int MIRROR;             // Data Set Mirror Flag (STD14)
      string NAME;            // Data Set File Name
      };
```

B.6 IC Design Index Description (ICD)

This section describes the **Bartels User Language** index variable types for the **IC Design** data access (ICD).

I_ATTRIBUTE - IC Design Net List Attribute

The **I_ATTRIBUTE** index variable type provides access to the connection list part or net attributes of the currently loaded IC layout. The structure definition of **I_ATTRIBUTE** is:

```
index I_ATTRIBUTE {           // Attribute Index
      string NAME;            // Attribute Name
      string VALUE;           // Attribute Value
      };
```

I_CNET - IC Design Net List

The **I_CNET** index variable type provides access to the connection list nets of the currently loaded IC layout. The structure definition of **I_CNET** is:

```
index I_CNET {                // Connection List Net Index
      string NAME;            // Net Name
      int NUMBER;             // Net Tree Number
      int PRIOR;              // Net Routing Priority
      double RDIST;           // Net Minimum Distance (STD2)
      int PINN;               // Net Pin Count
      index I_CPIN;           // Net Pin List
      index I_ATTRIBUTE;      // Net Attribute List
      index I_POOL UNRPOOL;   // Link to Unroutes Pool Element
      };
```

The net tree number is used for identifying the net. The **icd_gettreeidx** function provides access to the **I_CNET** index for a given net tree number. The minimum distance applies to the traces of the net; this distance must at minimum be kept to copper structures not belonging to the corresponding net. The net pin and/or attribute list can be accessed through a corresponding **forall-of** loop. The **UNRPOOL** variable provides access to the unrouted connections of the net; the corresponding airlines can be processed with the **icd_scanpool** system function.

I_CPART - IC Design Net List Part Entry

The **I_CPART** index variable type provides access to the connection list parts of the currently loaded IC layout. The structure definition of **I_CPART** is:

```
index I_CPART {               // Connection List Part Index
      string NAME;            // Part Name
      string PLNAME;          // Part Physical Library Name
      int USED;               // Part Placement Code:
                              //    0 = Part not placed
                              //    1 = Part placed
      int PEQUC;              // Part Equivalence Code
      int PINN;               // Part Pin Count
      int FPINN;              // Part Free Pin Count
      index I_MACRO MACRO;    // Link to Macro
      index I_CPIN;           // Part Pin List
      index I_ATTRIBUTE;      // Part Attribute List
      };
```

The part pin and/or attribute lists can be accessed through corresponding **forall-of** loops. Component swaps can be applied on parts with identical equivalence codes to optimize the placement.

I_CPIN - IC Design Net List Part Pin Entry

The **I_CPIN** index variable type provides access to the connection list part pins of the currently loaded IC layout. The structure definition of **I_CPIN** is:

```
index I_CPIN {                  // Net List Pin Index
      string NAME;              // Pin Name
      double RWIDTH;            // Pin Routing Width (STD2)
      int TREE;                 // Pin Net Tree Number
      int GATE;                 // Pin Gate Number
      int GEQUC;                // Pin Gate Equivalence Code
      int GEQUP;                // Pin Equivalence Code
      int GGRPC;                // Pin Gate Group Number
      int GPNUM;                // Pin Gate Relative Number
      index I_CNET CNET;        // Link to Pin Net
      index I_CPART CPART;      // Link to Pin Part
      };
```

The pin routing width defines the width for routing to the next connection point. The **CNET** and **CPART** variables provide backward links to the corresponding connection list net and part entries, respectively. The **GATE**, **GEQUC**, **GEQUP**, **GGRPC** and **GPNUM** variables can be utilized for checking pin/gate swap allowance.

I_FIGURE - IC Design Figure Element

The index variable type **I_FIGURE** provides access to all placed figure elements (polygons, traces, macro references, texts) of the currently loaded **IC Design** element. The structure definition of **I_FIGURE** is:

```
index I_FIGURE {                // Figure Element Index
      int TYP;                  // Element Type (CAP3)
      string NAME;              // Element Name
      double SIZE;              // Element Size (STD2)
      double X;                 // Element X Coordinate (STD2)
      double Y;                 // Element Y Coordinate (STD2)
      double ANGLE;             // Element Rotation Angle (STD3)
      int MIRROR;               // Element Mirror Mode (STD14)
      int LAYER;                // Element Layer (ICD1)
      int GROUP;                // Element Group Flag (STD13)
      int FIXED;                // Element Fixed Flag (STD11)
      int TREE;                 // Element Net Tree Number
      int RULEOBJID;            // Element Rule System Object Id
      index I_POOL POOL;        // Link to Pool Element
      index I_POLY POLY;        // Link to Polygon Element
      index I_LINE LINE;        // Link to Trace Element
      index I_NREF NREF;        // Link to Macro Reference (named)
      index I_UREF UREF;        // Link to Macro Reference (unnamed)
      index I_TEXT TEXT;        // Link to Text Element
      };
```

The **NAME** variable holds either the name for a named macro reference or the string of a text element. The **LAYOUT** variable specifies the element layer number. The **POOL** variable provides a cross link to the library pool element which builds up the figure element. The figure element attributes can be changed with the **ced_elem*chg** functions. A feature for scanning the complete figure element data with all hierarchy levels is provided with the **icd_scanfelem** function.

I_LEVEL - IC Design Signal Level

The **I_LEVEL** index variable type provides access to the connectivity levels, i.e., the net list and/or signal levels of the currently loaded IC layout. The structure definition of **I_LEVEL** is:

```
index I_LEVEL {                 // Connectivity Level Index
      int LEVVAL;               // Level Value (ICD6)
      };
```

I_LINE - IC Design Trace

The **I_LINE** index variable type provides access to the path and/or trace data defined on the currently loaded IC layout. The structure definition of **I_LINE** is:

```
index I_LINE {                  // Line Path Index
      double WIDTH;             // Line Path Width (STD2)
      int LAYER;                // Line Path Layer (ICD1)
      int TREE;                 // Line Path Tree Number
      int PN;                   // Line Path Point Count
      index I_POINT;            // Line Path Point List
      };
```

I_MACRO - IC Design Library Element

The **I_MACRO** index variable type provides access to the macros, i.e., the library elements (cell, pin) used on the currently loaded **IC Design** element. The structure definition of **I_MACRO** is:

```
index I_MACRO {                 // Macro Definition Index
      string NAME;              // Macro Name
      double MLX;               // Left Macro Border (STD2)
      double MLY;               // Lower Macro Border (STD2)
      double MUX;               // Right Macro Border (STD2)
      double MUY;               // Upper Macro Border (STD2)
      double MNX;               // Macro Origin X Coordinate (STD2)
      double MNY;               // Macro Origin Y Coordinate (STD2)
      int CLASS;                // Macro Class Code (STD1)
      int COMP;                 // Macro Status (STD16)
      };
```

I_NREF - IC Design Named Macro Reference

The **I_NREF** index variable type provides access to the named macro references, i.e., the name-specified library elements placed on the currently loaded element. These are cells on IC layout hierarchy level or pin on cell hierarchy level. The structure definition of **I_NREF** is:

```
index I_NREF {                  // Named Reference Index
      string NAME;              // Reference Name
      double X;                 // Reference X Coordinate (STD2)
      double Y;                 // Reference Y Coordinate (STD2)
      double ANGLE;             // Reference Rotation Angle (STD3)
      double SCALE;             // Reference Scale Factor
      int MIRROR;               // Reference Mirror Flag (STD14)
      index I_MACRO MACRO;      // Link to Macro
      };
```

The **MACRO** variable provides a cross link for accessing the referenced library element.

I_POINT - IC Design Polygon Point

The **I_POINT** index variable type provides access to the polygon points of a specific polygon or trace element. The structure definition of **I_POINT** is:

```
index I_POINT {                 // Polygon Point Index
      double X;                 // Polygon Point X Coordinate (STD2)
      double Y;                 // Polygon Point Y Coordinate (STD2)
      int TYP;                  // Polygon Point Type (STD15)
      };
```

The **I_POINT** index can only be applied as **of**-index for the point lists in **I_POLY** and/or **I_LINE**.

I_POLY - IC Design Polygon

The **I_POLY** index variable type provides access to the polygons (active areas, forbidden areas, outline, documentary lines) defined on the currently loaded **IC Design** element. The structure definition of **I_POLY** is:

```
index I_POLY {                  // Polygon Index
      int LAYER;                // Polygon Layer (ICD1)
      int TREE;                 // Polygon Net Tree Number
      int TYP;                  // Polygon Type (ICD4)
      int MVIS;                 // Polygon Mirror Mode (ICD3)
      int PN;                   // Polygon Point Count
      index I_POINT;            // Polygon Point List
      };
```

The polygon net tree number is only valid for active areas. The **icd_scanfelem** and/or **icd_scanall** function must be used to check the signal level of passive copper.

I_POOL - IC Design Pool Element

The **I_POOL** index variable type provides access to the currently loaded pool elements. The structure definition of **I_POOL** is:

```
index I_POOL {                  // Pool Element Index
      int TYP;                  // Pool Element Type (ICD7)
      int REFCNT;               // Pool Element Reference Count
      int LAYER;                // Pool Element Layer (ICD1)
      index I_POOL NXT;         // Link to Next Pool Element
      index I_POOL REF;         // Link to Reference Pool Element
      index I_POLY POLY;        // Link to Polygon Element
      index I_LINE LINE;        // Link to Trace Element
      index I_NREF NREF;        // Link to Macro Reference (named)
      index I_UREF UREF;        // Link to Macro Reference (unnamed)
      index I_TEXT TEXT;        // Link to Text Element
      index I_MACRO MACRO;      // Link to Library Element
      };
```

The **I_POOL** index is used for processing library definitions with the **icd_scanpool** system function. The **REFCNT** variable specifies, how often the pool element is currently referenced. The **NXT** and **REF** variables allow for fast pool element list traversal.

I_TEXT - IC Design Text

The **I_TEXT** index variable type provides access to text data defined on the currently loaded **IC Design** element. The structure definition of **I_TEXT** is:

```
index I_TEXT {                  // Text Index
      string STR;               // Text String
      double X;                 // Text X Coordinate (STD2)
      double Y;                 // Text Y Coordinate (STD2)
      double ANGLE;             // Text Rotation Angle (STD3)
      double SIZE;              // Text Size (STD2)
      int LAYER;                // Text Layer (ICD1)
      int MIRROR;               // Text Mirror Flag (STD14)
      int MODE;                 // Text Mode (ICD2)
      };
```

I_UREF - IC Design Unnamed Macro Reference

The **I_UREF** index variable type provides access to the unnamed macro references, i.e., the library elements placed on the currently loaded element without name specification. These are vias on IC layout hierarchy level. The structure definition of **I_UREF** is:

```
index I_UREF {                // Unnamed Reference Index
      int TREE;               // Reference Net Tree Number
      double X;               // Reference X Coordinate (STD2)
      double Y;               // Reference Y Coordinate (STD2)
      double ANGLE;           // Reference Rotation Angle (STD3)
      double SCALE;           // Reference Scale Factor
      int MIRROR;             // Reference Mirror Flag (STD14)
      index I_MACRO MACRO;    // Link to Macro
      };
```

The **MACRO** variable provides a cross link for accessing the referenced library element.

Appendix C
System Functions

This appendix describes the system functions included with the **Bartels User Language**, providing reference listings which are grouped according to the corresponding caller types. The system function descriptions are sorted in alphabetical order.

Contents

C.1 Function Reference

Each **Bartels User Language** system function is assigned to one of the caller types STD, CAP, LAY, SCM, GED, AR, CAM, CV, ICD or CED, respectively. This section lists the **User Language** system functions for each caller type.

Function Description Notations

Each detailed function description provided with this appendix indicates the function caller type and provides a formal function and/or parameter declaration. The function data type defines the data type of the corresponding function return value; **void** functions do not provide return values. The mode of operation of a function is explained in detail and/or illustrated by examples wherever necessary.

Parameter declarations can contain valid value range definitions. Such value range definitions consist of a lower and an upper value range boundary specification. Valid lower value range boundary specifications are:

[L	value >= lower boundary L
] L	value > lower boundary L
]	no lower boundary

Valid upper value range boundary specifications are:

U]	value >= upper boundary L
U [	value > upper boundary L
[	no upper boundary

The value range boundaries are separated by comma (**,**). The declaration

```
double ]0.0,[;
```

e.g., defines a parameter of type **double**, which must be greater than 0.0. The **User Language Compiler** knows about the parameter value ranges and issues error messages if parameter values are out of range.

A parameter declaration preceded with **&** indicates, that the corresponding parameter value is set and/or changed by the system function; the **User Language Compiler** will issue a warning message if a constant value or a calculation result is passed to such a parameter.

A parameter declaration preceded by ***** indicates that the corresponding parameter must reference a user function. The system function description contains the required user function declaration as well. When running the program, the system function will automatically activate the corresponding user function. The reference to the user function is usually optional; the keyword **NULL** must be used for the function reference parameter if no user function should be referenced. It is strongly recommended to take great care at the declaration of referenced user functions, since the **User Language Compiler** cannot recognize erroneous user function reference declarations (relating to required function data type, return value conventions, required parameters, etc.). In case of wrong function reference declarations the **User Language Interpreter** might show up with unpredictable results or fatal side effects at runtime.

A **void** system function parameter type indicates, that the corresponding parameter can be of any data type. A **[]** parameter type specification indicates, that the function expects optional parameters of any (**void**) type at this place.

C.1.1 Standard System Functions (STD)

The following **User Language** system functions are assigned to caller type STD; i.e., they can be called from each **User Language Interpreter** environment of the **Bartels AutoEngineer** (**Schematic Editor**, **Layout Editor**, **Autorouter**, **CAM Processor**, **CAM View**, **IC Design** and **Chip Editor**):

abs	Absolute value of an integer
acos	Arc cosine
angclass	Classify an angle value
arylength	Get array length
asin	Arc sine
askcoord	Interactive X/Y coordinate value query
askdbl	Interactive double value query
askdist	Interactive distance value query
askint	Interactive integer value query
askstr	Interactive string value query
atan	Arc tangent
atan2	Arc tangent of the angle defined by a point
atof	Convert string to floating point value
atoi	Convert string to integer value
bae_askddbename	Interactive DDB element name query
bae_askddbfname	Interactive DDB file name query
bae_askdirname	Interactive directory name query
bae_askfilename	Interactive file name query
bae_askmenu	Interactive BAE menu query
bae_askname	Activate BAE name selection dialog
bae_asksymname	Interactive BAE library element query
bae_callmenu	BAE menu function call
bae_charsize	Get BAE text/character dimensions
bae_cleardistpoly	Clear internal BAE distance query polygon
bae_clearpoints	Clear internal BAE polygon buffer
bae_clriactqueue	Clear the BAE interaction queue
bae_crossarcarc	Determine cross point(s) of two arcs
bae_crosslineline	Determine cross point of wide line segments
bae_crosslinepoly	Determine cross point of wide line with polygon
bae_crosssegarc	Determine cross point(s) of segment with arc
bae_crosssegseg	Determine cross point of segments/lines
bae_dashpolyline	Vectorize dashed BAE polygon

bae_deffuncprog	Define BAE function key
bae_defkeyprog	Define BAE standard key
bae_defmenu	BAE standard menu definition start
bae_defmenuprog	Define BAE menu entry
bae_defmenusel	Set BAE menu default selection
bae_defmenutext	Define BAE menu item text
bae_defselmenu	BAE submenu definition start
bae_dialaddcontrol	BAE dialog element definition
bae_dialadvcontrol	Add advanced BAE dialog element
bae_dialaskcall	Activate BAE dialog with listbox element callback function
bae_dialaskparams	Activate BAE dialog
bae_dialbmpalloc	Create BAE dialog bitmap
bae_dialboxbufload	Restore BAE dialog box data from buffer
bae_dialboxbufstore	Store BAE dialog box data to buffer
bae_dialboxperm	Activate modeless BAE dialog
bae_dialclr	Clear BAE dialog elements
bae_dialgetdata	Get BAE dialog element parameter
bae_dialgettextlen	Get BAE dialog text length
bae_dialsetcurrent	Set current BAE dialog box
bae_dialsetdata	Set BAE dialog element parameter
bae_endmainmenu	BAE main menu definition end
bae_endmenu	BAE menu definition end
bae_fontcharcnt	Get BAE font character count
bae_fontname	Get BAE text font name
bae_getactmenu	Get active BAE menu entry number
bae_getanglelock	Get BAE angle lock flag
bae_getbackgrid	Get BAE display grid
bae_getcasstime	Get date/time of last project connection data update caused by Packager/Backannotation
bae_getclassbitfield	Get BAE DDB class processing key
bae_getcmdbuf	BAE command history query
bae_getcolor	Get BAE color value
bae_getcoorddisp	Get BAE coordinate display mode
bae_getdblpar	Get BAE double parameter
bae_getfuncprog	Get BAE function key definition
bae_getgridlock	Get BAE grid lock flag
bae_getgridmode	Get BAE grid dependency mode

bae_getinpgrid	Get BAE input grid
bae_getintpar	Get BAE integer parameter
bae_getinvcolor	Get BAE color inversion mode
bae_getkeyprog	Get BAE standard key definition
bae_getmenubitfield	Get BAE menu function processing key
bae_getmenuitem	BAE menu item query
bae_getmenuprog	Get BAE menu entry definition
bae_getmenutext	Get BAE menu text
bae_getmoduleid	Get BAE module id
bae_getmsg	Get BAE HighEnd message
bae_getpackdata	Get last project Packager run data
bae_getpacktime	Get last project Packager run date/time
bae_getpolyrange	Get internal BAE polygon range
bae_getstrpar	Get BAE string parameter
bae_inittextscreen	Clear/initialize the BAE text screen
bae_inpoint	Input BAE point/coordinates with mouse
bae_inpointmenu	Input BAE point/coordinates with mouse and right mouse button callback function
bae_language	Get BAE user interface language code
bae_loadcoltab	Load BAE color table
bae_loadelem	Load BAE element
bae_loadfont	Load BAE text font
bae_menuitemhelp	Display BAE menu item help
bae_msgbox	Activate BAE message popup
bae_msgboxverify	Activate BAE message popup with Yes/No verification
bae_msgboxverifyquit	Activate BAE message popup with Yes/No/Quit verification
bae_msgprogressrep	Activate/update BAE progress display
bae_msgprogressterm	Terminate BAE progress display
bae_mtpsize	Get BAE popup display area dimensions
bae_nameadd	Add BAE name selection list element
bae_nameclr	Clear BAE name selection list
bae_nameget	Get BAE name selection list element
bae_numstring	Create numeric string
bae_peekiact	BAE interaction queue query
bae_plainmenutext	BAE menu item text conversion
bae_planddbclass	Get BAE element DDB class code
bae_planename	Get BAE element name

bae_planfname	Get BAE element file name
bae_plannotsaved	Get BAE element not saved flag
bae_plansename	Get BAE destination element name
bae_plansfname	Get BAE destination element file name
bae_planwslx	Get BAE element left workspace boundary
bae_planwsly	Get BAE element lower workspace boundary
bae_planwsnx	Get BAE element origin X coordinate
bae_planwsny	Get BAE element origin Y coordinate
bae_planwsux	Get BAE element right workspace boundary
bae_planwsuy	Get BAE element upper workspace boundary
bae_popareachoice	Define choice field area in active BAE popup menu
bae_popcliparea	Define clipping area in active BAE popup menu
bae_popclrtool	Clear BAE toolbar popup area
bae_popcolbar	Define BAE popup menu color bar display
bae_popcolchoice	Define BAE popup menu color bar selector
bae_popdrawpoly	Display/draw polygon/graphic in active BAE popup menu
bae_popdrawtext	Display/draw text in active BAE popup menu
bae_popmouse	Get BAE popup/toolbar mouse position
bae_poprestore	Restore BAE popup menu area
bae_popsetarea	Activate/select BAE popup menu/area
bae_popshow	Activate BAE popup menu
bae_poptext	Define BAE popup menu text display
bae_poptextchoice	Define BAE popup menu text selector
bae_postprocess	Run BAE postprocess
bae_progdir	Get BAE program directory path name
bae_prtdialog	Print string to BAE dialogue line
bae_querydist	Query BAE point to polygon distance
bae_readedittext	BAE text input/display
bae_readtext	BAE text input with popup menu
bae_redefmainmenu	BAE main menu redefinition start
bae_redefmenu	Redefine BAE menu item
bae_resetmenuprog	Reset BAE menu definitions
bae_sendmsg	Send BAE HighEnd message
bae_setanglelock	Set BAE angle lock fag
bae_setbackgrid	Set BAE display grid
bae_setclipboard	Store text string to BAE clipboard

bae_setcolor	Set BAE color value
bae_setcoorddisp	Set BAE coordinate display mode
bae_setdblpar	Set BAE double parameter
bae_setgridlock	Set BAE grid lock flag
bae_setgridmode	Set BAE grid dependency mode
bae_setinpgrid	Set BAE input grid
bae_setintpar	Set BAE integer parameter
bae_setmoduleid	Set BAE module id
bae_setmousetext	Set BAE mouse click input text
bae_setplanfname	Set BAE project file name
bae_setpopdash	Set BAE popup/toolbar polygon dash line parameters
bae_setstrpar	Set BAE string parameter
bae_settbsize	Define/display BAE toolbar area
bae_storecmdbuf	Store BAE command to command history
bae_storedistpoly	Store internal BAE distance query polygon
bae_storeelem	Store BAE element
bae_storekeyiact	Store BAE key-press interaction to queue
bae_storemenuiact	Store BAE menu interaction to queue
bae_storemouseiact	Store BAE mouse interaction to queue
bae_storepoint	Store point to internal BAE polygon
bae_storetextiact	Store BAE text input interaction to queue
bae_swconfig	Get BAE software configuration
bae_swversion	Get BAE software version
bae_tbsize	Get BAE toolbar dimensions
bae_twsize	Get BAE text screen workspace size
bae_wsmouse	Get BAE workspace mouse position
bae_wswinlx	Get BAE workspace window left boundary
bae_wswinly	Get BAE workspace window lower boundary
bae_wswinux	Get BAE workspace window right boundary
bae_wswinuy	Get BAE workspace window upper boundary
catext	Concatenate file name extension
catextadv	Optionally concatenate file name extension
ceil	Ceiling function
clock	Get elapsed processor time
con_clear	Delete internal logical net list
con_compileloglib	Compile logical library definition

con_deflogpart	Define a logical library part entry
con_getddbpattrib	Get part/pin attribute from DDB file
con_getlogpart	Get a logical library part definition
con_setddbpattrib	Store part/pin attribute to DDB file
con_storepart	Store part to internal logical net list
con_storepin	Store pin to internal logical net list
con_write	Write internal logical net list to file
convstring	Convert string
cos	Cosine
cosh	Hyperbolic cosine
cvtangle	Convert an angle value
cvtlength	Convert a length value
ddbcheck	Check DDB file/element availability
ddbclassid	Get DDB class identifier
ddbclassscan	Scan DDB class elements
ddbcopyelem	Copy DDB file element
ddbdelelem	Delete DDB file element
ddbelemrefcount	Get DDB file element reference count
ddbelemrefentry	Get DDB file element reference entry
ddbgetelemcomment	Get DDB file element comment
ddbgetlaypartpin	Get DDB file layout part pin data
ddbrenameelem	Rename DDB file element
ddbupdtime	Get DDB file element update time
ddbsetelemcomment	Set DDB file element comment
dirscan	Scan directory
existddbelem	Check DDB file element
exit	Terminate a program immediately
exp	Exponential function
fabs	Absolute value of a double
fclose	Close a file
fcloseall	Close all files opened by the program
feof	Test for end-of-file
fgetc	Read next character from file
fgets	Read next line of text from file
filemode	Get file mode
filesize	Get file size

filetype	Get file type
floor	Floor function
fmod	Floating point remainder
fopen	Open a file
fprintf	Print to a file using format
fputc	Write a character to a file
fputs	Write a string to a file
frexp	Break double into fraction and exponent
fseterrmode	Set the file functions error handling mode
get_date	Get the current system date
get_time	Get the current system time
getchr	Get a character from standard input
getcwd	Get current working directory path name
getenv	Get environment variable value
getextprog	Get file type specific application
getstr	Get a line of text from standard input
isalnum	Test for alphanumeric character
isalpha	Test for alphabetic character
iscntrl	Test for control character
isdigit	Test for numeric character
isgraph	Test for visible character
islower	Test for lowercase alphabetic character
isprint	Test for printing character
ispunct	Test for punctuation character
isspace	Test for whitespace character
isupper	Test for uppercase alphabetic character
isxdigit	Test for hexadecimal numeric character
kbhit	Test if key hit
kbstate	Shift/control/alt key state query
launch	Pass command to operating system without waiting for completion
ldexp	Multiply by a power of 2
localtime	Get local processor date and time
log	Natural logarithm; base e
log10	Common logarithm; base ten
mkdir	Create directory
modf	Break double into integer and fractional

namestrcmp	Name string compare
numstrcmp	Numeric string compare
perror	Print error message
pow	Raise a double to a power
printf	Print to standard output using format
programid	Get current program name
putchr	Write a character to standard output
putenv	Set environment variable
puts	Write a string to standard output (append NL)
putstr	Write a string to standard output
quicksort	Sort index list
remove	Delete a file or directory
rename	Change the name of a file
rewind	Seek to the beginning of a file
rulecompile	Compile a rule definition
rulesource	Get rule definition source code
scanddbenames	Scan DDB file element names
scandirfnames	Scan directory file names
setprio	Set BAE process priority
sin	Sine
sinh	Hyperbolic sine
sprintf	Print to string using format
sqlcmd	SQL command execution
sqlerr	SQL error status query
sqlinit	SQL database initialization
sqrt	Square root
strcmp	String compare
strcspn	String prefix length not matching characters
strdelchr	Delete characters from string
strextract	Extract sub-string from another string
strextractfilepath	Extract directory name from a file path name string
strgetconffilename	Get environment variable expanded configuration file name
strgetvarfilename	Get environment variable expanded file name string
strgetpurefilename	Extract file name from file path name string
strlen	String length
strlistitemadd	Add string to string list

strlistitemchk	Search string in string list
strlower	Convert string to lowercase
strmatch	Test for string pattern match
strnset	Fill part or all of string with any character
strreverse	Reverse string
strscannext	Forward find characters in string
strscanprior	Backward find characters in string
strset	Fill string with any character
strspn	String prefix length matching characters
strupper	Convert string to uppercase
syngetintpar	Get BNF/scanner integer parameter
synparsefile	BNF/parser input file scan
synparseincfile	BNF/parser include file scan
synparsestring	BNF/Parser string scan
synscaneoln	BNF/scanner end-of-line recognition
synscanigncase	BNF/scanner keyword case-sensitivity mode setting
synscanline	BNF/scanner input line number
synscanstring	BNF/scanner input string
synsetintpar	Set BNF/scanner integer parameter
system	Pass command to operating system and wait for completion
tan	Tangent
tanh	Hyperbolic tangent
tolower	Convert uppercase to lowercase character
toupper	Convert lowercase to uppercase character
uliptype	Get User Language interpreter environment
ulipversion	Get User Language interpreter version
ulproginfo	Get User Language program info
ulsystem	Run another User Language program
ulsystem_exit	Run a User Language program after exiting current User Language program
vardelete	Delete global User Language variable
varget	Get global User Language variable value
varset	Set global User Language variable value

C.1.2 Schematic Capture System Functions (CAP)

The following **User Language** system functions are assigned to caller type CAP; i.e., they can be called from the **Schematic Editor** interpreter environment of the **Bartels AutoEngineer**:

cap_blockname	Get SCM sheet block name
cap_blocktopflag	Get SCM sheet block hierarchy level
cap_figboxtest	Check SCM element rectangle cross
cap_findblockname	Find SCM block circuit sheet with given block name
cap_findlayconpart	Get layout connection list part index
cap_findlayconpartpin	Get layout connection list pin index
cap_findlaycontree	Get layout connection list net name net index
cap_getglobnetref	Get global net name reference
cap_getlaytreeidx	Get layout connection list net number net index
cap_getpartattrib	Get SCM part attribute value
cap_getrulecnt	Get rule count for specific SCM object
cap_getrulename	Get rule name from specific SCM object
cap_getscbustapidx	Get currently scanned SCM bus tap
cap_getscclass	Get currently scanned SCM class
cap_getscrefpidx	Get currently scanned SCM library element
cap_getscstkcnt	Get SCM scan function stack depth
cap_gettagdata	Get SCM tag symbol destination data
cap_lastconseg	Get last modified SCM connection segment
cap_lastfigelem	Get last modified SCM figure list element
cap_layconload	Load layout net list
cap_maccoords	Get SCM (scanned) macro coordinates
cap_macload	Load SCM macro element to memory
cap_macrelease	Unload/release SCM macro element from memory
cap_mactaglink	Get SCM (scanned) macro tag link data
cap_nrefsearch	Search named SCM reference
cap_partplan	Get SCM part sheet name
cap_pointpoolidx	Get SCM junction point pool element
cap_ruleconatt	Attach rule(s) to SCM connection segment
cap_rulecondet	Detach rules from SCM connection segment
cap_ruleerr	SCM rule system error code query
cap_rulefigatt	Attach rule(s) to SCM figure list element
cap_rulefigdet	Detach rules from SCM figure list element

cap_ruleplanatt	Attach rule(s) to currently loaded SCM element
cap_ruleplandet	Detach rules from currently loaded SCM element
cap_rulequery	Perform rule query on specific SCM object
cap_scanall	Scan all SCM figure list elements
cap_scanfelem	Scan SCM figure list element
cap_scanpool	Scan SCM pool element
cap_vecttext	Vectorize SCM text

C.1.3 Schematic Editor System Functions (SCM)

The following **User Language** system functions are assigned to caller type SCM; i.e., they can be called from the **Schematic Editor** interpreter environment of the **Bartels AutoEngineer**:

scm_askrefname	SCM reference name selection
scm_asktreenam	SCM net name selection
scm_attachtextpos	Attach text position to SCM element
scm_checkbustapplot	Get SCM bus tap plot status
scm_checkjunctplot	Get SCM junction point plot status
scm_chkattrname	SCM attribute name validation
scm_conseggrpchg	Change SCM connection segment group flag
scm_deflibname	SCM setup default library name
scm_defloglname	SCM setup default packager library name
scm_defsegbus	SCM connection segment bus definition
scm_delconseg	Delete SCM connection segment
scm_delelem	Delete SCM figure list element
scm_drawelem	Redraw SCM figure list element
scm_elemangchg	Change SCM figure list element rotation angle
scm_elemgrpchg	Change SCM figure list element group flag
scm_elemmirrchg	Change SCM figure list element mirror mode
scm_elemposchg	Change SCM figure list element position
scm_elemsizechg	Change SCM figure list element size
scm_findpartplc	Layout part placement status query (BAE HighEnd)
scm_getdblpar	Get SCM double parameter
scm_getgroupdata	SCM group placement data query
scm_gethighlnet	Get SCM net highlight mode
scm_gethpglparam	SCM HP-GL plot parameter query
scm_getinputdata	SCM input data query
scm_getintpar	Get SCM integer parameter
scm_getstrpar	Get SCM string parameter
scm_highlnet	Set SCM net highlight mode
scm_pickanyelem	Pick any SCM figure list element
scm_pickbustap	Pick SCM bus tap
scm_pickconseg	Pick SCM connection segment
scm_pickelem	Pick SCM figure list element
scm_setdblpar	Set SCM double parameter

scm_setintpar	Set SCM integer parameter
scm_setpartattrib	Set SCM part attribute value
scm_setpickconseg	Set SCM default connection pick element
scm_setpickelem	Set SCM default pick element
scm_setstrpar	Set SCM string parameter
scm_settagdata	Set SCM tag symbol pin destination
scm_storecon	Place SCM connection
scm_storelabel	Place SCM label
scm_storepart	Place SCM part
scm_storepoly	Place SCM internal polygon
scm_storetext	Place SCM text

C.1.4 Layout System Functions (LAY)

The following **User Language** system functions are assigned to caller type LAY; i.e., they can be called from the **Layout Editor**, the **Autorouter** and the **CAM Processor** interpreter environment of the **Bartels AutoEngineer**:

lay_defelemname	Layout setup default element name
lay_deflibname	Layout setup default library name
lay_defusrunit	Layout setup default user units
lay_doclayindex	Layout documentary layer display index
lay_doclayname	Layout setup documentary layer name
lay_doclayside	Layout setup documentary layer side mode
lay_doclaytext	Layout setup documentary layer text mode
lay_figboxtest	Check layout element rectangle cross
lay_findconpart	Find layout part index of a named part
lay_findconpartpin	Find layout part pin index of a named part pin
lay_findcontree	Find layout net index of a named net
lay_getplanchkparam	Get layout distance check parameters
lay_getpowplanetree	Get layout power plane tree number
lay_getpowpolystat	Layout power layer polygon status query
lay_getrulecnt	Get rule count for specific layout object
lay_getrulename	Get rule name from specific layout object
lay_getscclass	Get currently scanned layout class
lay_getscpartrpidx	Get currently scanned layout part
lay_getscrefpidx	Get currently scanned layout library element
lay_getscstkcnt	Get layout scan function stack depth
lay_getsctextdest	Get scanned layout text line destination
lay_gettreeidx	Find layout net index of a tree
lay_grpdisplay	Layout setup group display layer
lay_lastfigelem	Get last modified layout figure list element
lay_maccoords	Get layout (scanned) macro coordinates
lay_macload	Load layout macro element to memory
lay_macrelease	Unload/release layout macro element from memory
lay_menulaylinecnt	Get the layer menu lines count
lay_menulaylinelay	Get layer number of specified layer menu line
lay_menulaylinename	Get name of specified layer menu line
lay_nrefsearch	Search named layout reference
lay_planmidlaycnt	Get layout inside layer count

lay_plantoplay	Get layout top layer
lay_pltmarklay	Layout setup plot marker layer
lay_ruleerr	Layout rule system error code query
lay_rulefigatt	Attach rule(s) to layout figure list element
lay_rulefigdet	Detach rules from layout figure list element
lay_rulelaysatt	Attach rule(s) to layout layer stackup
lay_rulelaysdet	Detach rules from layout layer stackup
lay_ruleplanatt	Attach rule(s) to currently loaded layout element
lay_ruleplandet	Detach rules from currently loaded layout element
lay_rulequery	Perform rule query on specific layout object
lay_scanall	Scan all layout figure list elements
lay_scanfelem	Scan layout figure list element
lay_scanpool	Scan layout pool element
lay_setfigcache	Fill layout figure list access cache
lay_setplanchkparam	Set layout distance check parameters
lay_toplayname	Layout setup top layer name
lay_vecttext	Vectorize layout text

C.1.5 Layout Editor System Functions (GED)

The following **User Language** system functions are assigned to caller type GED; i.e., they can be called from the **Layout Editor** interpreter environment of the **Bartels AutoEngineer**:

ged_asklayer	GED layer selection
ged_askrefname	GED reference name selection
ged_asktreeidx	GED net selection
ged_attachtextpos	Attach text position to layout element
ged_delelem	Delete GED figure list element
ged_drawelem	Redraw GED figure list element
ged_drcerrorhide	Set/reset GED DRC error acceptance mode
ged_drcpath	GED trace test placement design rule check
ged_drcpoly	GED polygon test placement design rule check
ged_drcvia	GED via test placement design rule check
ged_elemangchg	Change GED figure list element rotation angle
ged_elemfixchg	Change GED figure list element fixed flag
ged_elemgrpchg	Change GED figure list element group flag
ged_elemlaychg	Change GED figure list element layer
ged_elemmirrchg	Change GED figure list element mirror mode
ged_elemposchg	Change GED figure list element position
ged_elemsizechg	Change GED figure list element size
ged_getautocornins	Get GED auto corner insertion input mode
ged_getdblpar	Get GED double parameter
ged_getdrcmarkmode	Get GED DRC error display mode
ged_getdrcstatus	Get GED DRC completion status
ged_getgroupdata	GED group placement data query
ged_gethighlnet	Get GED net highlight mode/color
ged_getinputdata	GED input data query
ged_getintpar	Get GED integer parameter
ged_getlaydefmode	Get GED default layer mode
ged_getlayerdefault	Get GED default layer
ged_getmincon	Get GED Mincon function type
ged_getpathwidth	Get GED path standard widths
ged_getpickmode	Get GED element pick mode
ged_getpickpreflay	Get GED pick preference layer
ged_getpowlayerrcnt	Get GED power layer error count

ged_getsegmovmode	Get GED trace segment move mode
ged_getstrpar	Get GED string parameter
ged_getviaoptmode	Get GED trace via optimization mode
ged_getwidedraw	Get GED wide line display start width
ged_groupselect	GED group selection
ged_highlnet	Set GED net highlight mode/color
ged_layergrpchg	Select GED group by layer
ged_partaltmacro	Change GED net list part package type
ged_partnamechg	Change GED part name
ged_pickanyelem	Pick any GED figure list element
ged_pickelem	Pick GED figure list element
ged_setautocornins	Set GED auto corner insertion input mode
ged_setdblpar	Set GED double parameter
ged_setdrcmarkmode	Set GED DRC error display mode
ged_setintpar	Set GED integer parameter
ged_setlaydefmode	Set GED default layer mode
ged_setlayerdefault	Set GED default layer
ged_setmincon	Set GED Mincon function type
ged_setnetattrib	Set GED net attribute value
ged_setpathwidth	Set GED path standard widths
ged_setpickelem	Set GED default pick element
ged_setpickmode	Set GED element pick mode
ged_setpickpreflay	Set GED pick preference layer
ged_setplantoplay	Set GED layout top layer
ged_setsegmovmode	Set GED trace segment move mode
ged_setstrpar	Set GED string parameter
ged_setviaoptmode	Set GED trace via optimization mode
ged_setwidedraw	Set GED wide line display start width
ged_storedrill	Place GED drill hole
ged_storepart	Place GED part or padstack
ged_storepath	Place GED internal polygon as path
ged_storepoly	Place GED internal polygon
ged_storetext	Place GED text
ged_storeuref	Place GED unnamed reference (via or pad)

C.1.6 Autorouter System Functions (AR)

The following **User Language** system functions are assigned to caller type AR; i.e., they can be called from the **Autorouter** interpreter environment of the **Bartels AutoEngineer**:

ar_asklayer	Autorouter layer selection
ar_delelem	Delete Autorouter figure list element
ar_drawelem	Redraw Autorouter figure list element
ar_elemangchg	Change Autorouter figure list element rotation angle
ar_elemfixchg	Change Autorouter figure list element fixed flag
ar_elemmirrchg	Change Autorouter figure list element layer
ar_elemposchg	Change Autorouter figure list element mirror mode
ar_elemsizechg	Change Autorouter figure list element position
ar_getdblpar	Get Autorouter double parameter
ar_getintpar	Get Autorouter integer parameter
ar_getmincon	Change Autorouter figure list element size
ar_getpickpreflay	Get Autorouter Mincon function type
ar_getstrpar	Get Autorouter string parameter
ar_getwidedraw	Get Autorouter pick preference layer
ar_highlnet	Get Autorouter wide line display start width
ar_partnamechg	Set Autorouter net highlight mode
ar_pickelem	Change Autorouter net list part name
ar_setdblpar	Set Autorouter double parameter
ar_setintpar	Set Autorouter integer parameter
ar_setmincon	Pick Autorouter figure list element with mouse
ar_setnetattrib	Set Autorouter Mincon function type
ar_setpickpreflay	Set Autorouter net attribute value
ar_setplantoplay	Set Autorouter pick preference layer
ar_setstrpar	Set Autorouter string parameter
ar_setwidedraw	Set Autorouter wide line display start width
ar_storepart	Place Autorouter part or padstack
ar_storepath	Place Autorouter internal polygon as path
ar_storeuref	Place Autorouter unnamed reference (via or pad)

C.1.7 CAM Processor System Functions (CAM)

The following **User Language** system functions are assigned to caller type CAM; i.e., they can be called from the **CAM Processor** interpreter environment of the **Bartels AutoEngineer**:

cam_askplotlayer	CAM plot layer selection
cam_getdblpar	Get CAM double parameter
cam_getdrlaccuracy	CAM drill tool tolerance query
cam_getgenpltparam	CAM general plot parameter query
cam_getgerberapt	CAM Gerber aperture definition query
cam_getgerberparam	CAM Gerber plot parameter query
cam_gethpglparam	CAM HP-GL plot parameter query
cam_getintpar	Get CAM integer parameter
cam_getplotlaycode	CAM plot layer code query
cam_getpowpltparam	CAM power layer plot parameter query
cam_getwidedraw	CAM wide line display start width query
cam_plotgerber	CAM Gerber photo plot output
cam_plothpgl	CAM HP-GL pen plot output
cam_setdblpar	Set CAM double parameter
cam_setdrlaccuracy	Set CAM drill tool tolerance
cam_setgenpltparam	Set CAM general plot parameters
cam_setgerberapt	Set CAM Gerber aperture definition
cam_setintpar	Set CAM integer parameter
cam_setplotlaycode	Set CAM plot layer code
cam_setpowpltparam	Set CAM power layer plot parameters
cam_setwidedraw	Set CAM wide line display start width

C.1.8 CAM View System Functions (CV)

The following **User Language** system functions are assigned to caller type CV; i.e., they can be called from the **CAM View** interpreter environment of the **Bartels AutoEngineer**:

cv_aptgetcolor	Get CAM View aperture color
cv_aptsetcolor	Set CAM View aperture color
cv_deldataset	Delete CAM View data set
cv_getdblpar	Get CAM View double parameter
cv_getintpar	Get CAM View integer parameter
cv_movedataset	Move CAM View data set
cv_setdblpar	Set CAM View double parameter
cv_setintpar	Set CAM View integer parameter

C.1.9 IC Design System Functions (ICD)

The following **User Language** system functions are assigned to caller type ICD; i.e., they can be called from the **Chip Editor** interpreter environment of the **Bartels AutoEngineer**:

icd_altpinlay	IC Design setup alternate pin layer
icd_cellconlay	IC Design setup intern. cell connection layer
icd_cellscan	IC Design setup DRC on cell level mode
icd_cellshr	IC Design setup cell keepout area shrink
icd_ciflayname	IC Design setup CIF output layer name
icd_cstdsiz	IC Design setup standard cell height
icd_defelemname	IC Design setup default element name
icd_deflibname	IC Design setup default library name
icd_drcarc	IC Design setup DRC arc mode
icd_drcgrid	IC Design setup DRC grid
icd_drclaymode	IC Design setup layer DRC mode
icd_drcmaxpar	IC Design setup DRC parallel check length
icd_drcminwidth	IC Design setup DRC layer minimal dimensions
icd_drcrect	IC Design setup DRC orthogonal mode
icd_ecnlaymode	IC Design setup layer connectivity check
icd_findconpart	Find IC Design part index of a named part
icd_findconpartpin	Find IC Design part pin index of a named part pin
icd_findcontree	Find IC Design net index of a named net
icd_getrulecnt	Get rule count for specific object
icd_getrulename	Get rule name from specific object
icd_gettreeidx	Find IC Design net index of a tree
icd_grpdisplay	IC Design setup group display layer
icd_lastfigelem	Get last modified IC Design figure list element
icd_maccoords	Get IC Design (scanned) macro coordinates
icd_nrefsearch	Search named IC Design reference
icd_outlinelay	IC Design setup cell outline layer
icd_pindist	IC Design setup pin keepout distance
icd_plcxgrid	IC Design setup placement grid
icd_plcxoffset	IC Design setup placement offset
icd_routcellcnt	IC Design setup number of power supply cells
icd_routcellname	IC Design setup name of power supply cell
icd_ruleerr	Rule System error code query

icd_rulefigatt	Attach rule(s) to figure list element
icd_rulefigdet	Detach rules from figure list element
icd_ruleplanatt	Attach rule(s) to currently loaded element
icd_ruleplandet	Detach rules from currently loaded element
icd_rulequery	Perform rule query on specific object
icd_scanall	Scan all IC Design figure list elements
icd_scanfelem	Scan IC Design figure list element
icd_scanpool	Scan IC Design pool element
icd_stdlayname	IC Design setup standard layer name
icd_stdpinlay	IC Design setup standard pin layer
icd_vecttext	Vectorize IC Design text

C.1.10 Chip Editor System Functions (CED)

The following **User Language** system functions are assigned to caller type CED; i.e., they can be called from the **Chip Editor** interpreter environment of the **Bartels AutoEngineer**:

ced_asklayer	CED layer selection
ced_delelem	Delete CED figure list element
ced_drawelem	Redraw CED figure list element
ced_elemangchg	Change CED figure list element rotation angle
ced_elemfixchg	Change CED figure list element fixed flag
ced_elemgrpchg	Change CED figure list element group flag
ced_elemlaychg	Change CED figure list element layer
ced_elemmirrchg	Change CED figure list element mirror mode
ced_elemposchg	Change CED figure list element position
ced_elemsizechg	Change CED figure list element size
ced_getlaydispmode	Get CED layer display mode
ced_getmincon	Get CED Mincon function type
ced_getpathwidth	Get CED path standard widths
ced_getpickpreflay	Get CED pick preference layer
ced_getwidedraw	Get CED wide line display start width
ced_groupselect	CED group selection
ced_highlnet	Set CED net highlight mode
ced_layergrpchg	Select CED group by layer
ced_partaltmacro	Change CED net list part cell type
ced_partnamechg	Change CED net list part name
ced_pickelem	Pick CED figure list element
ced_setlaydispmode	Set CED layer display mode
ced_setmincon	Set CED Mincon function type
ced_setpathwidth	Set CED path standard widths
ced_setpickpreflay	Set CED pick preference layer
ced_setwidedraw	Set CED wide line display start width
ced_storepart	Place CED part or pin
ced_storepath	Place CED internal polygon as path
ced_storepoly	Place CED internal polygon
ced_storetext	Place CED text
ced_storeuref	Place CED unnamed reference (via or subpart)

C.2 Standard System Functions

This section describes (in alphabetical order) the standard system functions of the **Bartels User Language**. See Appendix C.1 for function description notations.

abs - Absolute value of an integer (STD)

Synopsis

```
int abs(                    // Returns result value
      int;                  // Input integer value
      );
```

Description

The **abs** function calculates and returns the absolute value of the given integer value.

acos - Arc cosine (STD)

Synopsis

```
double acos(                // Returns result value (STD3)
      double [-1.0,1.0];    // Input cosine value
      );
```

Description

The **acos** function calculates and returns the arc cosine value of the given double value. The resulting angle value is in radians.

angclass - Classify an angle value (STD)

Synopsis

```
int angclass(               // Returns angle class code
      double;               // Input angle value (STD3)
      );
```

Description

The **angclass** function determines and returns a class code for the given angle value. Possible return values are 0 for 0 degree angle, 1 for 90 degree angle, 2 for 180 degree angle, 3 for 270 degree angle or (-1) for other angle values. The input angle value must be in radians.

arylength - Get array length (STD)

Synopsis

```
int arylength(              // Returns array element count
      void;                 // Any input value
      );
```

Description

The **arylength** function returns the array element count for the given (array) input value.

asin - Arc sine (STD)

Synopsis

```
double asin(                // Returns result value (STD3)
      double [-1.0,1.0];    // Input sine value
      );
```

Description

The **asin** function calculates and returns the arc sine value of the given double value. The resulting angle value is in radians.

askcoord - Interactive X/Y coordinate value query (STD)

Synopsis

```
int askcoord(                  // Returns status
      & double;                // Returns X coordinate value (STD2)
      & double;                // Returns Y coordinate value (STD2)
      int [0,1];               // Input mode:
                               //    0 = Coordinates relative to last position
                               //    1 = Coordinates absolute
      );
```

Description

The **askcoord** function activates a dialog X and Y coordinate value input. The input mode specifies whether absolute (Jump absolute) or relative (Jump relative) coordinates are to be queried. The user input is returned with the first two function parameters. For absolute coordinate queries, these parameters are also input parameters for preset coordinate values. The function returns zero if the query was successful or nonzero if the query was aborted.

askdbl - Interactive double value query (STD)

Synopsis

```
int askdbl(                    // Returns status
      & double;                // Returns double value
      string;                  // Prompt string
      int;                     // Maximum input string length
      );
```

Description

The **askdbl** function asks the user for a double value, indicating the required interaction with the given prompt string. The user input double value is returned with the first parameter. The function return value is nonzero, if an invalid input value has been specified.

askdist - Interactive distance value query (STD)

Synopsis

```
int askdist(                   // Returns status
                               //    -1 = Eingabe ungültig/abgebrochen
                               //     0 = Valid distance input value entered
                               //     1 = Valid distance input value entered,
                               //          corner button pressed
      & double;                // Returns distance value
      string;                  // Prompt string
      int [0,15];              // Input control:
                               //    1 = Negative input allowed
                               //    2 = Circular input allowed
                               //    4 = String item prompt message
                               //    8 = Round corner button
      );
```

Description

The **askdist** function asks the user for a distance value, indicating the required interaction with the given prompt string. The third parameter determines the type of valid inputs. The user input value is interpreted in default user units, and is returned with the first parameter. The function returns zero for valid distance value inputs, 1 for valid distance value inputs with corner button pressed, or (-1) for invalid inputs or if the input function was aborted.

askint - Interactive integer value query (STD)

Synopsis

```
int askint(                    // Returns status
      & int;                   // Returns integer value
      string;                  // Prompt string
      int;                     // Maximum input number length
      );
```

Description

The **askint** function asks the user for an integer value, indicating the required interaction with the given prompt string. The user input integer value is returned with the first parameter. The function returns nonzero if an invalid input value has been specified.

askstr - Interactive string value query (STD)

Synopsis

```
string askstr(                 // Returns string
      string;                  // Prompt string
      int;                     // Maximum input string length
      );
```

Description

The **askstr** function asks the user for a string value, indicating the required interaction with the given prompt string. The user input string value is passed with the function return value.

atan - Arc tangent (STD)

Synopsis

```
double atan(                   // Returns result value
      double;                  // Input angle value (STD3)
      );
```

Description

The **atan** function calculates and returns the arc tangent value of the given angle value. The input angle value must be in radians.

atan2 - Arc tangent of the angle defined by a point (STD)

Synopsis

```
double atan2(                  // Returns result value (STD3)
      double;                  // Input point Y coordinate
      double;                  // Input point X coordinate
      );
```

Description

The **atan2** function calculates and returns the arc tangent value of the angle defined by the given input point coordinates (and the origin point). The resulting angle value is in radians.

atof - Convert string to floating point value (STD)

Synopsis

```
double atof(                   // Returns floating point value
      string;                  // Input string
      );
```

Description

The **atof** function converts and returns the double value represented by the given string value. The result is undefined if the double value cannot be represented.

atoi - Convert string to integer value (STD)

Synopsis

```
int atoi(                    // Returns integer value
      string;                // Input string
      );
```

Description

The **atoi** function converts and returns the integer value represented by the given string value. The result is undefined if the integer value cannot be represented.

bae_askddbename - Interactive DDB element name query (STD)

Synopsis

```
int bae_askddbename(         // Returns status
      & string;              // Returns element name
      string;                // DDB file name
      int ]0,[;              // DDB element class (STD1)
      string;                // Prompt string
                             //    empty string: Standard file dialog
                             //    ! prefix: Save file dialog
                             //    otherwise: Load file dialog
      );
```

Description

The **bae_askddbename** function allows the user to select a DDB element name by either keyboard input or mouse-selection in a request popup window showing the list of available elements. If the prompt string is an empty string the function uses the standard prompt for DDB element name queries. Any name passed with the first parameter is used as default element name selection. I.e., an empty string must be passed to the first parameter if no default element name is required and/or allowed. The function returns nonzero if no valid element name was selected or zero otherwise.

bae_askddbfname - Interactive DDB file name query (STD)

Synopsis

```
int bae_askddbfname(         // Returns status
      & string;              // Returns file name
      int [0,1];             // Existence check
      int [0,3];             // File name dialog mode (bit flags):
                             //    1 = Check if file exists
                             //    2 = specified file name is default
      string;                // Prompt string
                             //    empty string: Standard file dialog
                             //    ! prefix: Save file dialog
                             //    otherwise: Load file dialog
      );
```

Description

The **bae_askddbfname** function allows the user to select a DDB file name by either keyboard input or mouse-selection in a request popup window showing the list of available DDB files. The standard BAE DDB file name prompt is used if an empty prompt string is specified. A file save dialog is activated instead of a file open dialog if the first character of the file name prompt string is an exclamation mark (**!**), i.e., the exclamation mark is faded out and the confirmation button of the Windows file name dialog changes from Open to Save. The function returns nonzero if no valid file name was selected and/or existence check is nonzero and the file doesn't exist or zero otherwise.

bae_askdirname - Interactive directory name query (STD)

Synopsis

```
int bae_askdirname(           // Returns status
      & string;               // Returns directory name
      string;                 // Directory name for scan start
      string;                 // Prompt string
      );
```

Description

The **bae_askdirname** function allows the user to select a directory name by either keyboard input or mouse-selection in a request popup window showing the list of available directories. The directory name for scan start specifies the top level directory for the popup window directory selection display. If the prompt string is an empty string the function uses the standard prompt for directory name queries. The function returns nonzero if no valid directory name was selected or zero otherwise.

bae_askfilename - Interactive file name query (STD)

Synopsis

```
int bae_askfilename(          // Returns status
      & string;               // Returns file name
      string;                 // File name extension string
      string;                 // Prompt string
                              //    empty string: Standard file dialog
                              //    ! prefix: Save file dialog
                              //    otherwise: Load file dialog
      );
```

Description

The **bae_askfilename** function allows the user to select a file name by either keyboard input or mouse-selection in a popup window showing the list of available files. The file name extension string can be used to set a file name extension filter. On empty string input, all files are scanned/displayed. On extension specification (e.g., **.ddb**, **.dat**, **.txt**, **.**, **.***, etc.), only those files matching the extension are scanned and/or displayed. The **-** tag works for file name exclusion. On **-** input all files matching BAE system or data file extensions (**.ass**, **.con**, **.ddb**, **.def**, **.exe**, **.fre**, **.ulc** and **.usf**, respectively) are faded-out from display (this feature can be used for output/plot file queries where system/library/project files must not be selected). The standard BAE file name prompt is used if an empty prompt string is specified. A file save dialog is activated instead of a file open dialog if the first character of the file name prompt string is an exclamation mark (**!**), i.e., the exclamation mark is faded out and the confirmation button of the Windows file name dialog changes from Open to Save. The function returns nonzero if no valid file name was selected or zero otherwise.

bae_askmenu - Interactive BAE menu query (STD)

Synopsis

```
int bae_askmenu(              // Returns selected menu item index (0..49),
                              //  or (-1) on menu selection abort
      int [1,50];             // Menu item count
      string;                 // First menu item string
      []                      // Subsequent menu item strings
      );
```

Description

The **bae_askmenu** function activates a user-specific menu with up to 48 mouse-selectable menu items. The function returns the number of the selected menu item or (-1) if the menu selection was aborted. Menu item numbering starts at 0.

See also

Function **bae_defmenusel**.

bae_askname - Activate BAE name selection dialog (STD)

Synopsis

```
int bae_askname(              // Status
      & string;               // Returns selected name
      string;                 // Prompt string (or empty string)
      int;                    // Maximum input string length
      );
```

Description

The **bae_askname** function activates a dialog for selecting a name from the name list which is currently defined with the **bae_nameadd** function. The second parameter specifies a non-standard input prompt. The system uses a predefined standard prompt if an empty string is passed as prompt string. The third parameter sets the maximum user input string length. The selected name is returned through the first parameter. The function returns zero if a name was selected or nonzero if the function was aborted without valid name selection.

See also

Functions **bae_nameadd**, **bae_nameclr**, **bae_nameget**.

bae_asksymname - Interactive BAE library element query (STD)

Synopsis

```
int bae_asksymname(           // Returns status
      & string;               // Returns library element name
      & string;               // Returns DDB library file name
      int ]0,[;               // Database class (STD1)
      string;                 // Library file directory
      string;                 // Default library path name
      string;                 // Default symbol/element name
      );
```

Description

The **bae_asksymname** function activates a dialog for selecting a library element of the specified database class from a selectable library file. The function returns zero if a library element was successfully selected or non-zero if the user aborted the dialog without valid element selection.

bae_callmenu - BAE menu function call (STD)

Synopsis

```
int bae_callmenu(             // Returns status
      int [0,9999];           // Menu function number (STD4)
      );
```

Description

The **bae_callmenu** function calls the specified BAE menu function passing the interactions defined with the **bae_store*iact** functions. The function returns nonzero on menu function errors or invalid menu function numbers.

bae_charsize - Get BAE text/character dimensions (STD)

Synopsis

```
void bae_charsize(
      & double;               // Returns character width (pixels)
      & double;               // Returns character height (pixels)
      );
```

Description

The **bae_charsize** function determines the current BAE character dimensions and returns the corresponding pixel values with its parameters.

bae_cleardistpoly - Clear internal BAE distance query polygon (STD)

Synopsis

```
void bae_cleardistpoly(
      );
```

Description

The **bae_cleardistpoly** function deletes the internal distance query polygon created with the **bae_storedistpoly** function.

See also

Functions **bae_storedistpoly**.

bae_clearpoints - Clear internal BAE polygon buffer (STD)

Synopsis

```
void bae_clearpoints(
      );
```

Description

The **bae_clearpoints** function deletes the internally stored polygon point list. This function should be called before the first **bae_storepoint** call to delete previously stored points.

See also

Functions **bae_getpolyrange**, **bae_storedistpoly**, **bae_storepoint**.

bae_clriactqueue - Clear the BAE interaction queue (STD)

Synopsis

```
void bae_clriactqueue(
      );
```

Description

The **bae_clriactqueue** function deletes all interactions stored in the interaction queue. The interaction queue is used for passing interactions to the BAE menu functions activated through **bae_callmenu**.

bae_crossarcarc - Determine cross point(s) of two arcs (STD)

Synopsis

```
int bae_crossarcarc(          // Crosspoint count
      double;                 // Arc 1 start point X coordinate (STD2)
      double;                 // Arc 1 start point Y coordinate (STD2)
      double;                 // Arc 1 center point X coordinate (STD2)
      double;                 // Arc 1 center point Y coordinate (STD2)
      int [1,2];              // Arc 1 center point type code (STD15)
      double;                 // Arc 1 end point X coordinate (STD2)
      double;                 // Arc 1 end point Y coordinate (STD2)
      double;                 // Arc 2 start point X coordinate (STD2)
      double;                 // Arc 2 start point Y coordinate (STD2)
      double;                 // Arc 2 center point X coordinate (STD2)
      double;                 // Arc 2 center point Y coordinate (STD2)
      int [1,2];              // Arc 2 center point type code (STD15)
      double;                 // Arc 2 end point X coordinate (STD2)
      double;                 // Arc 2 end point Y coordinate (STD2)
      & double;               // Crosspoint 1 X coordinate (STD2)
      & double;               // Crosspoint 1 Y coordinate (STD2)
      & double;               // Crosspoint 2 X coordinate (STD2)
      & double;               // Crosspoint 2 Y coordinate (STD2)
      );
```

Description

The **bae_crossarcarc** function determines the crossing points for the specified arcs. The function returns the number of crosspoints (0, 1, or 2). The coordinates of existing crosspoints are also returned through the crosspoint functions parameters.

See also

Functions **bae_crosslineline**, **bae_crosslinepoly**, **bae_crosssegarc**, **bae_crosssegseg**.

bae_crosslineline - Determine cross point of wide line segments (STD)

Synopsis

```
int bae_crosslineline(        // Crossing flag
      double;                 // Line 1 start point X coordinate (STD2)
      double;                 // Line 1 start point Y coordinate (STD2)
      double;                 // Line 1 end point X coordinate (STD2)
      double;                 // Line 1 end point Y coordinate (STD2)
      double ]0.0,[;          // Line 1 width (STD2)
      double;                 // Line 2 start point X coordinate (STD2)
      double;                 // Line 2 start point Y coordinate (STD2)
      double;                 // Line 2 end point X coordinate (STD2)
      double;                 // Line 2 end point Y coordinate (STD2)
      double ]0.0,[;          // Line 2 width (STD2)
      );
```

Description

The **bae_crosslineline** function checks whether the specified wide line segments are crossing each other. The function returns 1 if the segments are crossing each other or zero otherwise.

See also

Functions **bae_crossarcarc**, **bae_crosslinepoly**, **bae_crosssegarc**, **bae_crosssegseg**.

bae_crosslinepoly - Determine cross point of wide line with polygon (STD)

Synopsis

```
int bae_crosslinepoly(        // Crossing flag
      double;                 // Line start point X coordinate (STD2)
      double;                 // Line start point Y coordinate (STD2)
      double;                 // Line end point X coordinate (STD2)
      double;                 // Line end point Y coordinate (STD2)
      double ]0.0,[;          // Line width (STD2)
      );
```

Description

The **bae_crosslinepoly** function checks whether the specified wide line segment crosses the temporary polygon created with **bae_storepoint**. The function returns 1 if the segment crosses the polygon or zero otherwise.

See also

Functions **bae_crossarcarc**, **bae_crosslineline**, **bae_crosssegarc**, **bae_crosssegseg**, **bae_storepoint**.

bae_crosssegarc - Determine cross point(s) of segment with arc (STD)

Synopsis

```
int bae_crosssegarc(          // Crosspoint count
      double;                 // Segment start point X coordinate (STD2)
      double;                 // Segment start point Y coordinate (STD2)
      double;                 // Segment end point X coordinate (STD2)
      double;                 // Segment end point Y coordinate (STD2)
      double;                 // Arc start point X coordinate (STD2)
      double;                 // Arc start point Y coordinate (STD2)
      double;                 // Arc center point X coordinate (STD2)
      double;                 // Arc center point Y coordinate (STD2)
      int [1,2];              // Arc center point type code (STD15)
      double;                 // Arc end point X coordinate (STD2)
      double;                 // Arc end point Y coordinate (STD2)
      int [0,1];              // Crosspoint priority flag
      & double;               // Crosspoint 1 X coordinate (STD2)
      & double;               // Crosspoint 1 Y coordinate (STD2)
      & double;               // Crosspoint 2 X coordinate (STD2)
      & double;               // Crosspoint 2 Y coordinate (STD2)
      );
```

Description

The **bae_crosssegarc** function determines the crossing points for the specified segment and arc. The function returns the number of crosspoints (0, 1, or 2). The coordinates of existing crosspoints are also returned through the crosspoint functions parameters.

See also

Functions **bae_crossarcarc**, **bae_crosslineline**, **bae_crosslinepoly**, **bae_crosssegseg**.

bae_crosssegseg - Determine cross point of segments/lines (STD)

Synopsis

```
int bae_crosssegseg(          // Crossing flag
      int [0,1];              // Infinite line comparison flag
      double;                 // Line 1 start point X coordinate (STD2)
      double;                 // Line 1 start point Y coordinate (STD2)
      double;                 // Line 1 end point X coordinate (STD2)
      double;                 // Line 1 end point Y coordinate (STD2)
      double;                 // Line 2 start point X coordinate (STD2)
      double;                 // Line 2 start point Y coordinate (STD2)
      double;                 // Line 2 end point X coordinate (STD2)
      double;                 // Line 2 end point Y coordinate (STD2)
      & double;               // Crosspoint X coordinate (STD2)
      & double;               // Crosspoint Y coordinate (STD2)
      );
```

Description

The **bae_crosssegseg** function checks whether the specified segments and/or lines are crossing each other. The first function parameter specifies whether a segment comparison or a infinite line comparison should be carried out. The function returns 1 if a crosspoint was found or zero otherwise. The crosspoint coordinates are returned through the last two function parameters if a crosspoint was found.

See also

Functions **bae_crossarcarc**, **bae_crosslineline**, **bae_crosslinepoly**, **bae_crosssegarc**.

bae_dashpolyline - Vectorize dashed BAE polygon (STD)

Synopsis

```
int bae_dashpolyline(         // Returns status
      int;                    // Polygon dash mode:
                              //    0 = straight line (no dash)
                              //    1 = dashed line
                              //    2 = dotted line
                              //    3 = dashed/dotted line
      double ]0.0,[;          // Polygon dash base length (STD2)
      double ]-0.5,0.5[;      // Polygon dash relative spacing
      * int;                  // Polygon line scan function
      * int;                  // Polygon arc scan function
      );
```

Description

The **bae_dashpolyline** function vectorizes the polygon previously stored with **bae_storepoint** using the specified dash parameters. The polygon line and arc scan functions are automatically called for each polygon line and/or arc, respectively. The function returns zero if the vectorization was successful, or nonzero on error or if the scan was aborted.

Polygon line scan function

```
int polylinescanfuncname(
      double xs,              // Line start point X coordinate (STD2)
      double ys,              // Line start point Y coordinate (STD2)
      double xe,              // Line end point X coordinate (STD2)
      double ye               // Line end point Y coordinate (STD2)
      )
{
      // Polygon line scan function statements
      :
      return(errstat);
}
```

The return value of the polygon line scan function should be zero if the scan was ok or nonzero on error or if the scan should be aborted.

Polygon arc scan function

```
int polylinescanfuncname(
      double xs,              // Arc start point X coordinate (STD2)
      double ys,              // Arc start point Y coordinate (STD2)
      double xe,              // Arc end point X coordinate (STD2)
      double ye               // Arc end point Y coordinate (STD2)
      double xc,              // Arc center point X coordinate (STD2)
      double yc               // Arc center point Y coordinate (STD2)
      int cwflag;             // Arc clockwise flag:
                              //       0 = arc counter-clockwise
                              //    else = arc clockwise
      )
{
      // Polygon arc scan function statements
      :
      return(errstat);
}
```

The return value of the polygon arc scan function should be zero if the scan was ok or nonzero on error or if the scan should be aborted.

See also

Functions **bae_clearpoints**, **bae_storepoint**.

bae_deffuncprog - Define BAE function key (STD)

Synopsis

```
int bae_deffuncprog(          // Returns status
      int [1,128];            // Function key number
      string;                 // User Language program name or
                              //  # followed by menu item (STD4)
      );
```

Description

The **bae_deffuncprog** function assigns the given **User Language** program (or BAE menu function) to the given function key. An empty string program name specification can be used to reset the current assignment. The function returns zero if done or nonzero on error (i.e., invalid parameters or reset request for undefined key bindings).

See also

Functions **bae_getfuncprog**, **bae_resetmenuprog**.

bae_defkeyprog - Define BAE standard key (STD)

Synopsis

```
int bae_defkeyprog(           // Returns status
      int;                    // Key character
      string;                 // User Language program name or
                              //  # followed by menu item (STD4)
      );
```

Description

The **bae_defkeyprog** function assigns the specified **User Language** program (or BAE menu function) to the given standard key. An empty string program name specification can be used to reset the current assignment. The function returns zero if done, or nonzero on error (i.e., invalid parameters or reset request for undefined key bindings).

See also

Functions **bae_getkeyprog**, **bae_resetmenuprog**.

bae_defmenu - BAE menu definition start (STD)

Synopsis

```
int bae_defmenu(              // Returns status
      int [0,999];            // Menu code number
      int [0,999];            // Menu area code number:
                              //    1   = main menu area
                              //    101 = first submenu area
                              //    102 = second submenu area
                              //    :   = : submenu area
      );
```

Description

The **bae_defmenu** function starts the definition of a standard menu in the currently active BAE module. The function returns (-1) on error or zero otherwise. After calling **bae_defmenu**, the **bae_defmenutext** function should be applied for defining the menu entries. The menu definition initiated with **bae_defmenu** *must* be terminated by a call to the **bae_endmenu** function. The **bae_resetmenuprog** function can be used to reset *all* menu assignments, thus restoring the default menu configuration of the currently active BAE module.

See also

Functions **bae_defmenuprog**, **bae_defmenutext**, **bae_endmenu**, **bae_redefmenu**, **bae_resetmenuprog**.

bae_defmenuprog - Define BAE menu entry (STD)

Synopsis

```
int bae_defmenuprog(          // Selection code or (-1) on error
      int [0,999];            // Menu number
      int [0,99];             // Menu line
      string;                 // Menu text
      string;                 // User Language program name or
                              //  # followed by menu item (STD4)
      int;                    // Menu entry processing key:
                              //    8000000h = always available
                              //    7FFFFFFh = available for each
                              //                element type
                              //    else     = (combined) DDB class
                              //                processing key
      );
```

Description

The **bae_defmenuprog** function assigns the specified menu text and the named **User Language** program (or BAE menu function) to the given menu entry. The menu number specifies the number of the main menu, whilst the menu line designates the position in the according submenu. An empty string program name specification can be used to reset the current assignment. The menu entry processing key activates ghost menu configurations. The processing key is a coded integer value as retrieved and/or defined using the **bae_getclassbitfield** and **bae_getmenubitfield** functions (hex value 80000000h can be entered to allow for application in any case). The **bae_defmenuprog** function returns zero if done or nonzero on error (i.e., invalid parameters or reset request for undefined menu assignments). The **bae_resetmenuprog** function can be used to reset *all* menu assignments, thus restoring the default menu configuration of the currently active BAE module.

See also

Functions **bae_getclassbitfield**, **bae_getmenubitfield**, **bae_getmenuprog**, **bae_getmenutext**, **bae_redefmenu**, **bae_resetmenuprog**.

bae_defmenusel - Set BAE menu default selection (STD)

Synopsis

```
void bae_defmenusel(
      int [-1,29];            // Menu item index
                              //  or (-1) for selection text store
      );
```

Description

The **bae_defmenusel** function sets the default selection for the next **bae_askmenu** call. This allows for the indication of the currently selected menu option. BAE Windows versions indicate the default selection through a tick marker, Motif versions grey-shade the preselected menu item, and the DOS versions and/or the sidemenu configurations preselect the menu item specified through the menu item index parameter. The **bae_defmenusel** selection is only valid for the next **bae_askmenu** call. **bae_askmenu** resets this selection, i.e., **bae_defmenusel** must be used to re-activate any required default selection for subsequent **bae_askmenu** calls.

See also

Function **bae_askmenu**.

bae_defmenutext - Define BAE menu item text (STD)

Synopsis

```
int bae_defmenutext(          // Returns status
      int [0,99];             // Menu line
      string;                 // Menu text
      int;                    // Menu entry processing key:
                              //    8000000h = always available
                              //    7FFFFFFh = available for each element type
                              //    else     = (combined) DDB class processing key
      );
```

Description

The **bae_defmenutext** function must be called after **bae_defmenu** and/or **bae_defselmenu** to store the given menu line text at the specified menu line of the current menu definition. A pulldown menu seperator line is inserted prior to the menu entry if the menu line text starts with a percent character (**%**). The commercial and-character **&** can be used to define menu accelerator keys, with the character preceded by the **&** sign defining the key for selecting the menu item through the keyboard. The menu entry processing key activates ghost menu configurations. The processing key is a coded integer value as retrieved and/or defined using the **bae_getclassbitfield** and **bae_getmenubitfield** functions (hex value 80000000h can be entered to allow for application in any case). The function returns (-1) on error or zero otherwise. The **bae_resetmenuprog** function can be used to reset *all* menu assignments, thus restoring the default menu configuration of the currently active BAE module.

See also

Functions **bae_defmenu**, **bae_defselmenu**, **bae_getclassbitfield**, **bae_getmenubitfield**, **bae_plainmenutext**, **bae_redefmenu**, **bae_resetmenuprog**.

bae_defselmenu - BAE menu definition start (STD)

Synopsis

```
int bae_defselmenu(           // Returns status
      int [0,999];            // Menu code number
      int [0,999];            // Menu area code number:
                              //    1   = main menu area
                              //    101 = first submenu area
                              //    102 = second submenu area
                              //    :   = : submenu area
      );
```

Description

The **bae_defselmenu** function starts the definition of a standard menu in the currently active BAE module. The function returns (-1) on error or zero otherwise. After calling **bae_defselmenu**, the **bae_defmenutext** function should be applied for defining the menu entries. The menu definition initiated with **bae_defselmenu** *must* be terminated by a call to the **bae_endmenu** function. The **bae_resetmenuprog** function can be used to reset *all* menu assignments, thus restoring the default menu configuration of the currently active BAE module.

See also

Functions **bae_defmenuprog**, **bae_defmenutext**, **bae_endmenu**, **bae_redefmenu**, **bae_resetmenuprog**.

bae_dialaddcontrol - BAE dialog element definition (STD)

Synopsis

```
int bae_dialaddcontrol(       // Dialog element index or (-1) on error
      int [0,[;               // Parameter type (STD5)
      int;                    // Minimum int parameter value
      int;                    // Maximum int parameter value
      int;                    // Initial int parameter value
      double;                 // Minimum double parameter value
      double;                 // Maximum double parameter value
      double;                 // Initial double parameter value
      string;                 // Initial string parameter value
      int [0,[;               // Maximum string parameter value length
      double;                 // Dialog element X position [character units]
      double;                 // Dialog element Y position [character units]
      double;                 // Dialog element dimension [character units]
      string;                 // Parameter name/prompt
      );
```

Description

The **bae_dialaddcontrol** function defines a dialog element for the specified parameter type. Subsequent calls to the **bae_dialaskparams**, function activate a dialog with the dialog element displayed at the specified position and size. Any label and/or prompt to be displayed with the dialog element can be specified through the parameter name/prompt function parameter. The parameter value setting(s) for the new dialog element must be passed through the function parameter(s) matching the specified dialog element parameter type. The function returns a non-negative dialog element index if the dialog element creation was successful or (-1) otherwise. The dialog element index is used as dialog element selection parameter in subsequent calls to the **bae_dialgetdata** and **bae_dialsetdata** functions. Dialog elements created with **bae_dialaddcontrol** are valid and/or available until the next **bae_dialclr** call.

See also

Functions **bae_dialadvcontrol**, **bae_dialaskparams**, **bae_dialbmpalloc**, **bae_dialboxbufload**, **bae_dialboxbufstore**, **bae_dialclr**, **bae_dialgetdata**, **bae_dialsetdata**.

bae_dialadvcontrol - Add advanced BAE dialog element (STD)

Synopsis

```
int bae_dialadvcontrol(       // Returns new dialog control index or (-1) on error
      int [0,[;               // Parameter type (STD5)
      int;                    // Minimum int parameter value
      int;                    // Maximum int parameter value
      int;                    // Initial int parameter value
      double;                 // Minimum double parameter value
      double;                 // Maximum double parameter value
      double;                 // Initial double parameter value
      string;                 // Initial string parameter value
      int [0,[;               // Maximum string parameter value length
      double;                 // Dialog element X coordinate [character units]
      double;                 // Dialog element Y coordinate [character units]
      double;                 // Dialog element width [character units]
      double;                 // Dialog element height [character units]
      string;                 // Parameter name/prompt
      );
```

Description

The **bae_dialadvcontrol** function defines an advanced dialog element (with element height specification) for the specified parameter type. Subsequent calls to the **bae_dialaskparams**, function activate a dialog with the dialog element displayed at the specified position, width and height. Any label and/or prompt to be displayed with the dialog element can be specified through the parameter name/prompt function parameter. The parameter value setting(s) for the new dialog element must be passed through the function parameter(s) matching the specified dialog element parameter type. The function returns a non-negative dialog element index if the dialog element creation was successful or (-1) otherwise. The dialog element index is used as dialog element selection parameter in subsequent calls to the **bae_dialgetdata** and **bae_dialsetdata** functions. Dialog elements created with **bae_dialadvcontrol** are valid and/or available until the next **bae_dialclr** call.

See also

Functions **bae_dialaddcontrol**, **bae_dialaskparams**, **bae_dialbmpalloc**, **bae_dialboxbufload**, **bae_dialboxbufstore**, **bae_dialclr**, **bae_dialgetdata**, **bae_dialsetdata**.

bae_dialaskcall - Activate BAE dialog with listbox element callback function (STD)

Synopsis

```
int bae_dialaskcall(          // Returns positive action code, or
                              //    ( 0) on OK, or
                              //    (-1) on Abort or error
      string;                 // Dialog title
      int [0,3];              // Distance output units:
                              //    0 = mm
                              //    1 = Inch
                              //    2 = mil
                              //    3 = um
      double ]0.0,[;          // Dialog width [character units]
      double ]0.0,[;          // Dialog height [character units]
      * int;                  // Listbox element callback function
      );
```

Description

The **bae_dialaskcall** function activates a dialog with the dialog elements previously defined with **bae_dialaddcontrol**. The dialog title specified with the first function parameter is displayed in the title bar of the dialog window. The size of the dialog window can be specified through the dialog width and height function parameters. The function return value is set to zero if the OK dialog button is pressed. Pressing a non-default action button dialog element with a positive action code assignment causes **bae_dialaskcall** to return with the specified action code. A value of (-1) is returned on error or Abort dialog button activation. The **bae_dialgetdata**, function can be used to retrieve dialog parameter values after successfully completing **bae_dialaskcall**. Distance and/or length parameter values are automatically displayed and/or returned according to the distance output units mode function parameter passed to **bae_dialaskcall**. The last parameter allows for the specification of a user-defined callback function which is automatically called if an element of a listbox with **PA_MCALLBACK** type definition is selected.

Listbox Element Callback Function

```
int callbackfuncname(
      int reason,             // Callback reason
      int boxidx,             // Dialog box index
      int itemidx,            // List element index
      int itemid,             // List element id
      string itemstr          // List element text
      )
{
      // Function statements
      :
      return(errstat);
}
```

The callback function should return zero upon successfull completion, or non-zero for errors or abort requests.

See also

Functions **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialaskparams**, **bae_dialbmpalloc**, **bae_dialboxparam**, **bae_dialboxperm**, **bae_dialclr**, **bae_dialgetdata**, **bae_dialsetdata**.

bae_dialaskparams - Activate BAE dialog (STD)

Synopsis

```
int bae_dialaskparams(        // Returns positive action code, or
                              //    ( 0) on OK, or
                              //    (-1) on Abort or error
                              //    (-2) on dialog size change
      string;                 // Dialog title
      int [0,3];              // Distance output units:
                              //    0 = mm
                              //    1 = Inch
                              //    2 = mil
                              //    3 = um
      double ]0.0,[;          // Dialog width [character units]
      double ]0.0,[;          // Dialog height [character units]
      );
```

Description

The **bae_dialaskparams** function activates a dialog with the dialog elements previously defined with **bae_dialaddcontrol**. The dialog title specified with the first function parameter is displayed in the title bar of the dialog window. The size of the dialog window can be specified through the dialog width and height function parameters. The function return value is set to zero if the OK dialog button is pressed. Pressing a non-default action button dialog element with a positive action code assignment causes **bae_dialaskparams** to return with the specified action code. A value of (-1) is returned on error or Abort dialog button activation. A value of (-2) is returned if the dialog size is changed. The **bae_dialgetdata**, function can be used to retrieve dialog parameter values after successfully completing **bae_dialaskparams**. Distance and/or length parameter values are automatically displayed and/or returned according to the distance output units mode function parameter passed to **bae_dialaskparams**.

See also

Functions **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialaskcall**, **bae_dialbmpalloc**, **bae_dialboxperm**, **bae_dialclr**, **bae_dialgetdata**, **bae_dialsetdata**.

bae_dialbmpalloc - Create BAE dialog bitmap (STD)

Synopsis

```
int bae_dialbmpalloc(         // Returns non-negative bitmap id or (-1) on error
      double ]0.0,[;          // Requested bitmap width [character units]
      double ]0.0,[;          // Requested bitmap height [character units]
      int [2,31];             // Bitmap id
      & int;                  // Generated bitmap width [Pixel]
      & int;                  // Generated bitmap height [Pixel]
      );
```

Description

The **bae_dialbmpalloc** function creates a bitmap with the specified parameters in a dialog box to be activated with **bae_dialaskparams**. The function returns (-1) on error or a non-negative bitmap id (and the generated bitmap dimensions) if the bitmap was successfully created. Once the bitmap is generated, the **bae_popsetarea** function is used to select the bitmap for subsequent graphic output with the **bae_popdrawtext** and **bae_popdrawpoly** functions.

See also

Functions **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialaskcall**, **bae_dialaskparams**, **bae_dialboxperm**, **bae_dialclr**, **bae_popdrawpoly**, **bae_popdrawtext**, **bae_popsetarea**.

bae_dialboxbufload - Restore BAE dialog box data from buffer (STD)

Synopsis

```
int bae_dialboxbufload(       // Status
      int [1,[;               // Dialog box buffer id
      );
```

Description

The **bae_dialboxbufload** function is used to restore dialog box definitions which were previously saved with the **bae_dialboxbufstore** function. The function returns zero if the dialog box definitions were successfully loaded or nonzero on error.

See also

Functions **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialboxbufstore**, **bae_dialclr**.

bae_dialboxbufstore - Store BAE dialog box data to buffer (STD)

Synopsis

```
int bae_dialboxbufstore(      // Dialog box id (>0) or (-1) on error
      );
```

Description

The **bae_dialboxbufstore** function saves the current dialog box definitions to an internal buffer. The functions returns a buffer id or (-1) on error. The buffer id can be used in subsequent calls to the **bae_dialboxbufload** function for restoring these dialog box definitions.

See also

Functions **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialboxbufload**, **bae_dialclr**.

bae_dialboxperm - Activate modeless BAE dialog (STD)

Synopsis

```
int bae_dialboxperm(          // Returns positive dialog box id, or
                              //    (-1) on dialog box creation error, or
                              //    (-2) if maximum dialog box count reached
      string;                 // Dialog title
      int [0,3];              // Distance output units:
                              //    0 = mm
                              //    1 = Inch
                              //    2 = mil
                              //    3 = um
      double ]0.0,[;          // Dialog width [character units]
      double ]0.0,[;          // Dialog height [character units]
      );
```

Description

The **bae_dialboxperm** function activates a modeless dialog with the dialog elements previously defined with **bae_dialaddcontrol**. The dialog title specified with the first function parameter is displayed in the title bar of the dialog window. The size of the dialog window can be specified through the dialog width and height function parameters. The function returns the positive dialog box id if the dialog was successfully generated, or a negative value on error. The **bae_dialgetdata** function can be used to retrieve dialog parameter values after successfully completing **bae_dialboxperm** (and dialog activation with **bae_dialsetcurrent**). Distance and/or length parameter values are automatically displayed and/or returned according to the distance output units mode function parameter passed to **bae_dialboxperm**.

See also

Functions **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialaskcall**, **bae_dialaskparams**, **bae_dialbmpalloc**, **bae_dialclr**, **bae_dialgetdata**, **bae_dialsetcurrent**, **bae_dialsetdata**.

bae_dialclr - Clear BAE dialog elements (STD)

Synopsis

```
int bae_dialclr(              // Returns status
      );
```

Description

The **bae_dialclr** function clears/deletes all BAE dialog elements previously defined with **bae_dialaddcontrol**. The function return value is nonzero if dialogs are not supported in the current BAE user interface environment. To clear any dialog elements from previous dialog definitions, **bae_dialclr** should be called before starting a new BAE dialog definitions The function return value should be checked, and alternative user input facilities should be provided if dialogs are not supported.

See also

Functions **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialaskcall**, **bae_dialaskparams**, **bae_dialbmpalloc**, **bae_dialboxbufload**, **bae_dialboxbufstore**, **bae_dialboxperm**, **bae_dialgetdata**, **bae_dialsetdata**.

bae_dialgetdata - Get BAE dialog element parameter (STD)

Synopsis

```
int bae_dialgetdata(          // Returns status
      int [0,[;               // Dialog element index
      & int;                  // Dialog element integer value
      & double;               // Dialog element double value
      & string;               // Dialog element string value
      );
```

Description

The **bae_dialgetdata** function is used to retrieve dialog element parameter values after successful **bae_dialaskparams** calls. The query dialog element is selected through the dialog element index returned by **bae_dialaddcontrol** at the creation of the dialog element. The parameter value is returned through the function parameter matching the data type of the queried dialog element. The function returns nonzero if the dialog element parameter query failed.

See also

Functions **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialaskparams**, **bae_dialclr**, **bae_dialsetdata**.

bae_dialgettextlen - Get BAE dialog text length (STD)

Synopsis

```
double bae_dialgettextlen(    // Returns text length [character units]
      int [0,[;               // Dialog text (font) type
      string;                 // Dialog text string
      );
```

Description

The **bae_dialgettextlen** function calculates and returns the spacial requirements for displaying the specified dialog text string.

bae_dialsetcurrent - Set current BAE dialog box (STD)

Synopsis

```
int bae_dialsetcurrent(       // Returns status
      int [0,[;               // Dialog box id
      );
```

Description

The **bae_dialsetcurrent** activates the (modeless) dialog box specified through the dialog box id for subsequent dialog box operations. The functions returns zero if a dialog was successfully selected, or nonzero otherwise.

See also

Functions **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialaskcall**, **bae_dialaskparams**, **bae_dialbmpalloc**, **bae_dialboxperm**, **bae_dialclr**, **bae_dialgetdata**, **bae_dialsetdata**.

bae_dialsetdata - Set BAE dialog element parameter (STD)

Synopsis

```
int bae_dialsetdata(          // Returns status
      int [0,[;               // Dialog element index
      int [0,[;               // Dialog element parameter type (STD5)
      int;                    // Dialog element integer value
      double;                 // Dialog element double value
      string;                 // Dialog element string value
      );
```

Description

The **bae_dialsetdata** function is used to set dialog element parameter types and/or values previously defined with the **bae_dialaddcontrol** function. The dialog element to be changed is selected through the dialog element index returned by **bae_dialaddcontrol** at the creation of the dialog element. The new dialog element parameter value must be passed through the function parameter matching the specified dialog element parameter type. The function returns nonzero if the dialog element parameter value change failed.

See also

Functions **bae_dialaddcontrol**, **bae_dialadvcontrol**, **bae_dialaskparams**, **bae_dialclr**, **bae_dialgetdata**.

bae_endmainmenu - BAE main menu definition end (STD)

Synopsis

```
int bae_endmainmenu(          // Returns status
      );
```

Description

The **bae_endmainmenu** function terminates a main menu redefinition previously initiated with the **bae_redefmainmenu** function. The function returns (-1) on error or zero otherwise.

See also

Function **bae_redefmainmenu**.

bae_endmenu - BAE menu definition end (STD)

Synopsis

```
int bae_endmenu(              // Returns status
      );
```

Description

The **bae_endmenu** function terminates a menu definition previously initiated with either of the functions **bae_defmenu** or **bae_defselmenu**. The function returns (-1) on error or zero otherwise.

See also

Functions **bae_defmenu**, **bae_defselmenu**.

bae_fontcharcnt - Get BAE font character count (STD)

Synopsis

```
int bae_fontcharcnt(          // Returns font character count
      );
```

Description

The **bae_fontcharcnt** function returns the number of characters defined in the currently active BAE text font.

bae_fontname - Get BAE text font name (STD)

Synopsis

```
string bae_fontname(          // Returns text font name
      );
```

Description

The **bae_fontname** function returns the text font name of the currently loaded BAE element.

bae_getactmenu - Get active BAE menu entry number (STD)

Synopsis

```
int bae_getactmenu(           // Returns menu entry number (STD4)
      );
```

Description

The **bae_getactmenu** function returns the menu entry number (STD4) of the currently active BAE menu function or (-1) if no menu function is active. This feature is useful for key-called **User Language** programs.

bae_getanglelock - Get BAE angle lock flag (STD)

Synopsis

```
int bae_getanglelock(         // Returns angle lock flag (STD9)
      );
```

Description

The **bae_getanglelock** function returns the current BAE angle lock mode (0=angle unlocked, 1=angle locked).

See also

Function **bae_setanglelock**.

bae_getbackgrid - Get BAE display grid (STD)

Synopsis

```
void bae_getbackgrid(
      & double;               // Returns X display grid (STD2)
      & double;               // Returns Y display grid (STD2)
      );
```

Description

The **bae_getbackgrid** function returns the current BAE X/Y display grid values with its parameters. Zero grid values refer to switched-off grids.

See also

Function **bae_setbackgrid**.

bae_getcasstime - Get date/time of last project connection data update caused by Packager/Backannotation (STD)

Synopsis

```
string bae_getcasstime(       // Returns Packager net lsit name
      & int;                  // Returns time second
      & int;                  // Returns time minute
      & int;                  // Returns time hour
      & int;                  // Returns date day
      & int;                  // Returns date month
      & int;                  // Returns date year
      );
```

Description

The **bae_getcasstime** function can be used to retrieve the date and time of the last project netlist updated caused by **Packager** or Backannotation.

See also

Functions **bae_getpackdata**, **bae_getpacktime**.

bae_getclassbitfield - Get BAE DDB class processing key (STD)

Synopsis

```
int bae_getclassbitfield(     // Returns processing key
      int ]0,[;               // DDB database class code (STD1)
      );
```

Description

The **bae_getclassbitfield** function returns the processing key code assigned to the specified DDB database class. The function returns (-1) for invalid and/or unknown database class specifications.

The processing key is a coded integer value which - in a BAE module - is unique for each processable DDB database class. Such key codes can be combined using bit-or operations and can subsequently be assigned to specific menu entries using either of the **bae_defmenutext**, **bae_defmenuprog** or **bae_redefmenu** functions, thus allowing for the configuration of menu functions which can only be applied on certain database classes.

See also

Functions **bae_defmenuprog**, **bae_defmenutext**, **bae_getmenubitfield**, **bae_redefmenu**.

bae_getcmdbuf - BAE command history query (STD)

Synopsis

```
int bae_getcmdbuf(            // Returns status
      int [-50,2099];         // Command buffer index 0 to 49
                              //   or 1000 to 1099 for Undo items 1 to 100
                              //   or 2000 to 2099 for Redo items 1 to 100
                              //   or negative value for message history
      & string;               // Command (sequence) string
      & string;               // Command notification text
      );
```

Description

The **bae_getcmdbuf** function is used to query the current context menu command history, i.e., the list of commands activated through the right mouse button. The command buffer index parameter selects the command to be queried (zero being the most recent command). The command (sequence) string and the command notification text (as displayed in the title bar) are returned through the second and third function parameter. The function returns 1 if the query was successful or zero if an invalid command buffer index was specified.

See also

Function **bae_storecmdbuf**.

bae_getcolor - Get BAE color value (STD)

Synopsis

```
int bae_getcolor(             // Color value (STD18)
      int;                    // Display item type (SCM1|LAY9|ICD9)
      );
```

Description

The **bae_getcolor** function returns the color value currently set for the given display item type. The display item type value must be set according to the currently active **User Language Interpreter** environment.

See also

Function **bae_setcolor**.

bae_getcoorddisp - Get BAE coordinate display mode (STD)

Synopsis

```
int bae_getcoorddisp(         // Returns coordinate display mode (STD7)
      );
```

Description

The **bae_getcoorddisp** function returns the current BAE coordinate display mode. The return value is 0 for mm display units (micrometer units in **IC Design**) or 1 for Inch display units (mil units in **IC Design**).

See also

Function **bae_setcoorddisp**.

bae_getdblpar - Get BAE double parameter (STD)

Synopsis

```
int bae_getdblpar(            // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = maximum dialog box width
                              //    1 = maximum dialog box height
                              //    2 = display zoom factor
                              //    3 = Rubberband corner radius (STD2)
                              //    4 = Rubberband X vector coordinate (STD2)
                              //    5 = Rubberband Y vector coordinate (STD2)
                              //    6 = fixed X pick coordinate (STD2)
                              //    7 = fixed Y pick coordinate (STD2)
                              //    8 = Dialog box width:
                              //    9 = Dialog box height:
                              //    10 = Screen pick aperture (STD2)
                              //    11 = Element selection preview area
                              //          relative size [0.05, 0.95]
                              //    12 = Dialog box X unit pixels
                              //    13 = Dialog box Y unit pixels
      & double;               // Returns parameter value
      );
```

Description

The **bae_getdblpar** function is used to query **Bartels AutoEngineer**`double` parameter settings. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **bae_getintpar**, **bae_getstrpar**, **bae_setdblpar**, **bae_setintpar**, **bae_setstrpar**.

bae_getfuncprog - Get BAE function key definition (STD)

Synopsis

```
string bae_getfuncprog(       // Program name
      int [1,128];            // Function key number
      );
```

Description

The **bae_getfuncprog** function returns the name of the **User Language** program (or the hash-preceded BAE menu function number) assigned to the specified function key. An empty string is returned if no program is assigned to the key or if the function key number specification is invalid.

bae_getgridlock - Get BAE grid lock flag (STD)

Synopsis

```
int bae_getgridlock(          // Returns grid lock flag (STD8)
      );
```

Description

The **bae_getgridlock** function returns the current BAE grid lock mode (0=grid unlocked, 1=grid locked).

See also

Function **bae_setgridlock**.

bae_getgridmode - Get BAE grid dependency mode (STD)

Synopsis

```
int bae_getgridmode(          // Returns automatic grid setting mode
                              //    0x01: input grid = 0.25 × display grid
                              //    0x02: input grid = 0.50 × display grid
                              //    0x04: input grid = 1.00 × display grid
                              //    0x08: input grid = 2.00 × display grid
                              //    0x10: display grid = 0.25 × input grid
                              //    0x20: display grid = 0.50 × input grid
                              //    0x40: display grid = 1.00 × input grid
                              //    0x80: display grid = 2.00 × input grid
      );
```

Description

The **bae_getgridmode** function returns the current BAE grid dependency mode.

See also

Function **bae_setgridmode**.

bae_getinpgrid - Get BAE input grid (STD)

Synopsis

```
void bae_getinpgrid(
      & double;               // Returns X input grid (STD2)
      & double;               // Returns Y input grid (STD2)
      );
```

Description

The **bae_getinpgrid** function returns the current BAE X/Y input grid values with its parameters. Zero grid values refer to switched-off grids.

See also

Function **bae_setinpgrid**.

bae_getintpar - Get BAE integer parameter (STD)

Synopsis

```
int bae_getintpar(              // Returns status
      int [0,[;                 // Parameter type/number:
                                //    0 = Input coordinate range checking:
                                //        0 = range check enabled
                                //        1 = range check disabled
                                //    1 = Module change autosave mode:
                                //        0 = autosave without prompt
                                //        1 = prompt before autosave
                                //    2 = Display disable mode:
                                //        0 = display enabled
                                //        1 = display disabled
                                //    3 = User interface menu/mouse mode:
                                //        0 = Sidemenu configuration
                                //        1 = Pulldown menu, LMB context
                                //        2 = Pulldwon menu, RMB context
                                //    4 = Workspace text color mode:
                                //        0 = standard colors
                                //        1 = inverted standard colors
                                //        2 = workspace related colors
                                //    5 = Load display mode:
                                //        0 = display overview after load
                                //        1 = handle load display in bae_load
                                //    6 = File dialog view:
                                //        0 = old BAE style file selection
                                //        1 = Explorer style default view
                                //        2 = Explorer style list view
                                //        3 = Explorer style details
                                //        4 = Explorer style small icons
                                //        5 = Explorer style large icons
                                //        6 = Use default style and size
                                //    7 = Element selection box mode:
                                //        0 = display name only
                                //        1 = display name and date
                                //    8 = Element selection sort mode:
                                //        0|1 = sort by name
                                //        2 = sort numerically
                                //        3 = sort by date
                                //    9 = Placement visibility:
                                //        0 = placed elements visible
                                //        1 = unplaced elements visible
                                //    10 = Last file system error
                                //    11 = Command history disable flag:
                                //        0 = Command history enabled
                                //        1 = Command history disabled
                                //    12 = Popup menu mouse warp mode:
                                //        0 = No popup menu mouse warp
                                //        1 = First popup menu entry mouse warp
                                //        2 = Preselected menu entry mouse warp
                                //        +4 = Mouse position restore warp
                                //        +8 = Element pick position warp
                                //    13 = Save disable flag:
                                //        0 = save enabled
                                //        1 = save disabled
                                //    14 = Mouse rectangle min. size:
                                //        ]0,[ = min. rectangle size
                                //    15 = Mouse info display mode:
                                //        0 = Tooltip info display
                                //        1 = Continuous crosshair info display
                                //        2 = Crosshair info display with Ctrl
                                //    16 = Next dialog box id
                                //    17 = Last created tooltip id
                                //    18 = Polygon drop count
                                //    19 = Polygon check disabled flag:
                                //        0 = Polygon check enabled
                                //        1 = Polygon check disabled
```

```
//    20 = Cursor key grid mode:
//         0 = Input grid
//         1 = Pixel grid
//    21 = Unsaved plan flag
//    22 = Element batch load mode:
//         0 = no batch load
//         1 = Batch load
//         2 = batch load, restore zoom window
//    23 = Grid lines display:
//         0 = Dot grid
//         1 = Line grid
//    24 = Display mirroring flag
//    25 = Input grid display flag
//    26 = Mouse function repeat mode:
//         0 = Repeat menu function
//         +1 = Repeat keystroke function
//         +2 = Repeat context menu function
//    27 = Maximum Undo/Redo count
//    28 = Menu tree view mode:
//         0 = No menu tree view window
//         1 = Left attached menu tree view window
//         2 = Right attached menu tree view window
//    29 = Menu tree view pixel width
//    30 = Message history disabled flag
//    31 = Element load message mode:
//         0 = Standard message
//         1 = User message
//         2 = User error message
//    32 = Pick marker display mode:
//         0 = Circle marker
//         1 = Diamond marker
//    33 = Mouse drag status:
//         0 = No mouse drag
//         1 = Request mouse drag
//         2 = Mouse dragged
//         3 = Request mouse drag release
//    34 = Menu function repeat request flag
//    35 = Function aborted flag
//    36 = Plan selection preview flag
//    37 = File error display mode:
//         0 = Status message only
//         1 = Confirm message box
//    38 = Element selection reference display mode:
//         0 = Display project file references
//         1 = Display library file references
//    39 = Mouse double-click mode:
//         0 = Map double-click and
//              select 0 to right mouse button
//         1 = Ignore double-click
//         2 = Map double-click to right mouse button
//    40 = Mouse pick double-click mode:
//         0 = Map double-click to right mouse button
//         1 = Ignore double-click
//    41 = Dialog control support flags:
//         0 = No dialog control elements supported
//         |1 = Base dialog control set supported
//         |2 = List view supported
//         |4 = Progress display supported
//         |8 = Toolbar button supported
//    42 = Progress box display mode:
//         0 = No progress box window
//         1 = Display progress window
//    43 = Progress box abort request flag
//    44 = Middle mouse button disable flag
//    45 = Current undo items count
//    46 = File drag and drop operation flag
//    47 = Autoraise BAE window flag
//    48 = Menu function active count
```

```
                              //    49 = Internal polygon list point count
                              //    50 = Alternate configuration file priority
                              //    51 = Dialog position save mode:
                              //        0 = Store absolute coordinates
                              //        1 = Store main window relative coordinates
                              //        2 = Store main window monitor
                              //            absolute coordinates
                              //    52 = Message box default button index:
                              //        (-1) = No default (Abort or No)
                              //        0-2 = Default button index
      & int;                  // Returns parameter value
      );
```

Description

The **bae_getintpar** function is used to query **Bartels AutoEngineer** integer parameter settings. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **bae_getdblpar**, **bae_getstrpar**, **bae_setdblpar**, **bae_setintpar**, **bae_setstrpar**.

bae_getinvcolor - Get BAE color inversion mode (STD)

Synopsis

```
int bae_getinvcolor(          // Returns inverted color palette flag
      );
```

Description

The **bae_getinvcolor** function checks whether the BAE system color palette is "inverted". The system color palette is considered inverted if the darkest color in palette (as defined through a **bae.col** file in the BAE programs directory) is not black. The function returns nonzero if the BAE system color palette appears to be inverted or zero else.

bae_getmenubitfield - Get BAE menu function processing key (STD)

Synopsis

```
int bae_getmenubitfield(      // Returns processing key
      int [0,999];            // Menu code
      int [0,99];             // Menu line
      );
```

Description

The **bae_getmenubitfield** function returns the processing code assigned to the menu entry specified with the main menu code and the menu line number in the corresponding submenu. The function returns (-1) for invalid and/or unknown menu entry specifications.

The processing key is a coded integer value specifying a set of element types which can be subject to the menu function. This key is utilized throughout the **Bartels AutoEngineer** to activate and/or deactivate specific menu functions according to the type of element currently loaded (note that the ghost menu layouts of the BAE Windows versions are a typical application of this feature). With the hex value 80000000h assigned to its processing key, a menu function can *always* be applied (i.e., even if *no* element is loaded). With the hex value 7FFFFFFFh assigned to its processing key, a menu function can be applied to *any* currently loaded element type. Other processing key codes are defined in each BAE module according to the DDB database classes which can be processed. The processing key code for a specific database class can be retrieved with the **bae_getclassbitfield** function. Such processing key codes can be combined using bit-or operations and can subsequently be assigned to specific menu entries using either of the **bae_defmenutext**, **bae_defmenuprog** or **bae_redefmenu** functions, thus allowing for a user-defined ghost menu setup.

See also

Functions **bae_defmenuprog**, **bae_defmenutext**, **bae_getclassbitfield**, **bae_redefmenu**.

bae_getkeyprog - Get BAE standard key definition (STD)

Synopsis

```
string bae_getkeyprog(        // Program name
      int;                    // Key character
      );
```

Description

The **bae_getkeyprog** function returns the name of the **User Language** program (or the hash-preceded BAE menu function number) assigned to the specified standard key. An empty string is returned if no program is assigned to the key or if the key character specification is invalid.

See also

Functions **bae_defkeyprog**, **bae_resetmenuprog**.

bae_getmenuitem - BAE menu item query (STD)

Synopsis

```
int bae_getmenuitem(          // Return status
      & int;                  // Menu code
      & string;               // Menu text(s)
      & string;               // Menu command
      );
```

Description

The **bae_getmenuitem** function activates a menu function selection for menu item information query (i.e., the selected menu function is not executed). The menu code parameter returns the numeric menu call code of the selected menu item. The menu text parameter returns the selected hierarchical menu item text sequence (with menu texts separated by `->`). The menu command parameter returns the menu command string/sequence assigned to the selected menu item. The function returns zero if the query was successful or nonzero for failed or aborted queries.

See also

Functions **bae_callmenu**, **bae_defmenuprog**, **bae_getmenuprog**, **bae_getmenutext**.

bae_getmenuprog - Get BAE menu entry definition (STD)

Synopsis

```
string bae_getmenuprog(       // Program name
      int [0,999];            // Menu code
      int [0,99];             // Menu line
      );
```

Description

The **bae_getmenuprog** function returns the name of the **User Language** program (or the hash-preceded BAE menu function number) assigned to the specified menu entry. The menu code parameter specifies the main menu number, whilst the menu line parameter specifies the position in the corresponding submenu. An empty string is returned if no program is assigned to the menu entry or if the menu entry does not exist.

See also

Functions **bae_defmenuprog**, **bae_getmenuitem**, **bae_getmenutext**, **bae_resetmenuprog**.

bae_getmenutext - Get BAE menu text (STD)

Synopsis

```
string bae_getmenutext(       // Menu text
      int [0,999];            // Menu code
      int [0,99];             // Menu line
      );
```

Description

The **bae_getmenutext** function returns the menu text which is assigned to the given menu entry. The menu code parameter specifies the main menu number, whilst the menu line parameter specifies the position in the corresponding submenu. An empty string is returned if the specified menu entry does not exist.

See also

Functions **bae_defmenuprog**, **bae_getmenuitem**, **bae_getmenuprog**, **bae_plainmenutext**, **bae_resetmenuprog**.

bae_getmoduleid - Get BAE module id (STD)

Synopsis

```
string bae_getmoduleid(       // Returns module id
      );
```

Description

The **bae_getmoduleid** retrieves the module name/identification of the currently active BAE program module.

See also

Function **bae_setmoduleid**.

bae_getmsg - Get BAE HighEnd message (STD/HighEnd)

Synopsis

```
string bae_getmsg(            // Returns current message string
      );
```

Description

The **bae_getmsg** function is only available in **BAE HighEnd**. **bae_getmsg** is used for receiving pending messages from the **BAE HighEnd** message system. Messages are send to program instances of a **BAE HighEnd** session using the **bae_sendmsg** function. A **BAE HighEnd** session is started with a BAE call and includes any other BAE program instance subsequently started with the New Task function from the BAE main menu or with the Next SCM Window function from the **Schematic Editor**. Each **BAE HighEnd** module receiving a message automatically activates the **User Language** program named **bae_msg**. If **bae_msg** is not available, the system tries to start an interpreter-specific **User Language** program (**scm_msg** in the **Schematic Editor**, **ged_msg** in the **Layout Editor**, **ar_msg** in the **Autorouter**, etc.). The **bae_getmsg** function must be used in the ***_msg** **User Language** program to retrieve pending messages. Pending messages are only available during the execution of the ***_msg** **User Language** program, i.e., any message not retrieved by the ***_msg** program using the **bae_getmsg** function is lost. The message text string can be used for triggering certain actions in the destination program instances.

See also

Function **bae_sendmsg**.

bae_getpackdata - Get last project Packager run data (STD)

Synopsis

```
int bae_getpackdata(          // Returns status
      string;                 // Project file name
      & string;               // Returns layout library file name
      & string;               // Returns net list name
      );
```

Description

The **bae_getpackdata** retrieves the layout library file and net list name parameters used for the last **Packager** run on the specified project file. The function returns zero if the query was successful or non-zero if the paramater data was not found.

See also

Functions **bae_getcasstime**, **bae_getpacktime**.

bae_getpacktime - Get last project Packager run date/time (STD)

Synopsis

```
int bae_getpacktime(          // Returns status
      string;                 // Project file name
      & int;                  // Returns time second
      & int;                  // Returns time minute
      & int;                  // Returns time hour
      & int;                  // Returns date day
      & int;                  // Returns date month
      & int;                  // Returns date year
      );
```

Description

The **bae_getpacktime** retrieves the date and time of the last **Packager** run for the specified project file. The function returns zero if the query was successful or non-zero if the date/time information was not available.

See also

Functions **bae_getcasstime**, **bae_getpackdata**.

bae_getpolyrange - Get internal BAE polygon range (STD)

Synopsis

```
int bae_getpolyrange(         // Returns status
      & double;               // Returns polygon range left boundary
      & double;               // Returns polygon range lower boundary
      & double;               // Returns polygon range right boundary
      & double;               // Returns polygon range upper boundary
      );
```

Description

The **bae_getpolyrange** function is used to query the range of the internal BAE polygon defined with **bae_storepoint**. The function returns zero if the query was successful or non-zero otherwise.

See also

Functions **bae_clearpoints**, **bae_storepoint**.

bae_getstrpar - Get BAE string parameter (STD)

Synopsis

```
int bae_getstrpar(            // Rückgabe Status
      int [0,[;               // Parameter type/number:
                              //    0 = Current element comment text
                              //    1 = Current element specification
                              //    2 = Last file access error file
                              //    3 = Last file access error item
                              //  [ 4 = System parameter - no read access ]
                              //  [ 5 = System parameter - no read access ]
                              //    6 = Last loaded color table
                              //    7 = Menu text of last called function
                              //    8 = Current menu item element text
                              //    9 = Current element load user message
                              //    10 = Clipboard text string
                              //    11 = Next module call file argument
                              //    12 = Next module call element argument
                              //    13 = Next module call command/type argument
                              //    14 = Last output file name
                              //    15 = Host name
                              //  [ 16 = System parameter - no read access ]
                              //    17 = All users data directory
                              //    18 = Current user data directory
                              //    19 = Alternate configuration data directory
                              //    20 = Local data column
                              //    21 = Global data column
      & string;               // Returns parameter value
      );
```

Description

The **bae_getstrpar** function is used to query **Bartels AutoEngineer** string parameter settings. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **bae_getdblpar**, **bae_getintpar**, **bae_setdblpar**, **bae_setintpar**, **bae_setstrpar**.

bae_inittextscreen - Clear/initialize the BAE text screen (STD)

Synopsis

```
void bae_inittextscreen(
      );
```

Description

The **bae_inittextscreen** function clears the BAE graphic workarea and initializes the BAE text screen. The text cursor is set to the upper left corner of the text screen.

bae_inpoint - Input BAE point/coordinates with mouse (STD)

Synopsis

```
int bae_inpoint(              // Returns status
      double;                 // Start X coordinate (STD2)
      double;                 // Start Y coordinate (STD2)
      & double;               // Returns input X coordinate (STD2)
      & double;               // Returns input Y coordinate (STD2)
      int;                    // Drawing mode:
                              //    0 = no tracking display
                              //    1 = rubberband frame display
                              //    2 = rubberband line display
                              //    3 = rubberband circle display
                              //    4 = rubberband distance display
                              //    5 = rubberband zoom window display
                              //    6 = rubberband square display
                              //    7 = rubberband centered square display
                              //    8 = rubberband centered window display
                              //    9 = rubberband line polygon display
                              //    10 = rubberband outline polygon display
                              //    11 = rubberband circle center display
                              //    12 = rubberband fixed
                              //    13 = rubberband centered zoom window
                              //    14 = No window display, RMB immediate abort
                              //    15 = Fixed offset display
      );
```

Description

The **bae_inpoint** function activates an interactive coordinate point specification request (with mouse and optional submenu). The start coordinates specify the start point for Jump relative commands. The selected point coordinates are passed with the return parameters. A drawing mode value of 1 activates rubberband frame drawing from the start coordinate to the current mouse coordinate during coordinate selection. A drawing mode value of 2 activates rubberband line drawing from the start coordinate to the current mouse coordinate during coordinate selection. The function returns nonzero if a point has been selected or (-1) if the point selection has been aborted.

See also

Function **bae_inpointmenu**.

bae_inpointmenu - Input BAE point/coordinates with mouse and right mouse button callback function (STD)

Synopsis

```
int bae_inpointmenu(          // Returns status
      double;                 // Start X coordinate (STD2)
      double;                 // Start Y coordinate (STD2)
      & double;               // Returns input X coordinate (STD2)
      & double;               // Returns input Y coordinate (STD2)
      int;                    // Drawing mode:
                              //    0 = no tracking display
                              //    1 = rubberband frame display
                              //    2 = rubberband line display
                              //    3 = rubberband circle display
                              //    4 = rubberband distance display
                              //    5 = rubberband zoom window display
                              //    6 = rubberband square display
                              //    7 = rubberband centered square display
                              //    8 = rubberband centered window display
                              //    9 = rubberband line polygon display
                              //    10 = rubberband outline polygon display
                              //    11 = rubberband circle center display
                              //    12 = rubberband fixed
                              //    13 = rubberband centered zoom window
                              //    14 = unused
                              //    15 = Fixed offset display
      );
```

Description

The **bae_inpointmenu** function activates an interactive coordinate point specification request (with mouse and optional submenu). The start coordinates specify the start point for Jump relative commands. The selected point coordinates are passed with the return parameters. A drawing mode value of 1 activates rubberband frame drawing from the start coordinate to the current mouse coordinate during coordinate selection. A drawing mode value of 2 activates rubberband line drawing from the start coordinate to the current mouse coordinate during coordinate selection. The last parameter allows for the specification of a callback function (e.g., for displaying a specific options menu) to be activated if the right mouse button is pressed. The function returns nonzero if a point has been selected or (-1) if the point selection has been aborted.

Right mouse button callback function

```
int callbackfunction(         // Status
      double x                // Current input X coordinate
      double y                // Current input Y coordinate
      )
{
      // Right mouse button callback function statements
      :
      return(status);
}
```

The right mouse button callback function return value should be 0 to signal input completion, 1 to continue input, or any other value to abort input. Modified X/Y input coordinate parameters can be returned for zero callback function return values (input completed).

See also

Function **bae_inpoint**.

bae_language - Get BAE user interface language code (STD)

Synopsis

```
string bae_language(          // Returns language code:
                              //    DE = German
                              //    EN = English
                              //  for future use:
                              //    FR = French
                              //    IT = Italian
                              //    SP = Spanish
                              //    SV = Swedish
      );
```

Description

The **bae_language** function returns a code for identifying the language which is currently activated in the BAE user interface. This information can be used to support dynamic multi-language support.

bae_loadcoltab - Load BAE color table (STD)

Synopsis

```
int bae_loadcoltab(           // Returns status
      string;                 // Color table name
      );
```

Description

The **bae_loadcoltab** function loads the color table with the given name to the BAE. The function returns nonzero if the color table cannot be loaded.

bae_loadelem - Load BAE element (STD)

Synopsis

```
int bae_loadelem(             // Returns status
      string;                 // File name
      string;                 // Element name
      int [100,[;             // DDB class (STD1)
      );
```

Description

The **bae_loadelem** function loads a BAE element from DDB file to memory. The element is specified by the DDB file name, the element name and the DDB class. Possible **Schematic Editor** DDB class codes are 800, 801, 802 and 803 for SCM sheet, SCM symbol, SCM marker and SCM label, respectively. Possible BAE Layout DDB class codes are 100, 101, 102 and 103 for PCB layout, layout part, layout padstack and layout pad, respectively. Possible **IC Design** DDB class codes are 1000, 1001 and 1002 for IC layout, IC cell and IC pin, respectively. The function returns zero if the element was successfully loaded, (-1) on file access errors, 1 on text font load faults, 2 if referenced macros (library elements) are missing or 3 on unplaced parts and/or missing pins, i.e., if part symbols placed on a loaded layout do not match the net list specifications.

An empty element name can be specified with PCB Layout element class 100 for automatically loading the layout element with the default layout element name (see **LAYDEFELEMENT** command for **BSETUP**).

bae_loadelem automatically processes pending Backannotation requests when loading SCM plans to the **Schematic Editor**.

Warning

The **bae_loadelem** function changes all currently loaded BAE element data. As a result, any active index variable becomes invalid. It is strongly recommended to refrain from using **bae_loadelem** in index variable accessing program blocks such as **forall** loops. Since **bae_loadelem** reset the Undo facility, it is also strongly recommended to check (with **bae_plannotsaved**) whether the current element has been saved before calling **bae_loadelem**.

bae_loadfont - Load BAE text font (STD)

Synopsis

```
int bae_loadfont(             // Returns status
      string;                 // Text font name
      );
```

Description

The **bae_loadfont** function loads the name-specified BAE text font. The function returns nonzero on font load errors or zero otherwise.

bae_menuitemhelp - Display BAE menu item help (STD)

Synopsis

```
int bae_menuitemhelp(         // Status
      int [0,9999];           // Menu code (STD4)
      string;                 // Help file name (Windows .hlp file)
      );
```

Description

The **bae_menuitemhelp** displays the Windows online help file topic for the specified menu code. The function returns nonzero if invalid parameters are specified.

Limitations

bae_menuitemhelp operates only under Windows.

bae_msgbox - Activate BAE message popup (STD)

Synopsis

```
void bae_msgbox(
      int;                    // Message box style/icon:
                              //    0 = info (information)
                              //    1 = warning (exclamation-point)
                              //    2 = error (question-mark)
                              //    3 = fatal error (stop-sign)
                              //    else =  no icon
      string;                 // Message box text string
      string;                 // Message box title string
      );
```

Description

The **bae_msgbox** function activates a message box popup with a confirmation (OK) button. The appearance of the message popup window can be selected with the first parameter. The message box text string is displayed in the popup area. The message box title string is used as header string to be displayed in the message box popup window title bar. The appearance and layout of the message box (popup positioning, title bar display, confirmation button, text alignment, word wrapping, etc.) can differ depending on the host operating system platform.

See also

Functions **bae_msgboxverify**, **bae_msgboxverifyquit**.

bae_msgboxverify - Activate BAE message popup with Yes/No verification (STD)

Synopsis

```
int bae_msgboxverify(          // Returns answer code:
                               //    1 = Yes
                               //    0 = No (default)
      string;                  // Message box text string
      string;                  // Message box title string
      );
```

Description

The **bae_msgboxverify** function activates a message box popup with a Yes/No verification query. The function returns 1 on Yes input/selection or zero otherwise. The message box text string is displayed in the popup area. The message box title string is used as header string to be displayed in the message box popup window title bar. The appearance and layout of the message box (popup positioning, title bar display, yes/no buttons, text alignment, word wrapping, etc.) can differ depending on the host operating system platform.

See also

Functions **bae_msgbox**, **bae_msgboxverifyquit**.

bae_msgboxverifyquit - Activate BAE message popup with Yes/No/Quit verification (STD)

Synopsis

```
int bae_msgboxverifyquit(      // Returns answer code:
                               //    1 = Yes
                               //    0 = No
                               //    else = Cancel/Quit (default)
      string;                  // Message box text string
      string;                  // Message box title string
      );
```

Description

The **bae_msgboxverifyquit** function activates a message box popup with a Yes/No/Quit verification query. The function returns 1 on Yes input/selection or zero on No input/selection. Any other return value indicates a Cancel/Quit request. The message box text string is displayed in the popup area. The message box title string is used as header string to be displayed in the message box popup window title bar. The appearance and layout of the message box (popup positioning, title bar display, yes/no/quit buttons, text alignment, word wrapping, etc.) can differ depending on the host operating system platform.

See also

Functions **bae_msgbox**, **bae_msgboxverify**.

bae_msgprogressrep - Activate/update BAE progress display (STD)

Synopsis

```
int bae_msgprogressrep(        // Returns non-zero on error
      string;                  // Progress display text string
      int [0,258];             // Progress display type:
                               //    1 : Percentage progress
                               //    2 : Marquee progress slider
                               //    |256 : Display abort button
      int [0,10000];           // Progress display completion value [%*100]
      int [0,[;                // Progress display min. character length
      );
```

Description

The **bae_msgprogressrep** activates and/or updates the current BAE progress display with the specified parameters. The function returns zero on success or non-zero on error (if invalid parameters were specified). The **bae_msgprogressterm** function can be used to terminate the progress display.

See also

Function **bae_msgprogressterm**.

bae_msgprogressterm - Terminate BAE progress display (STD)

Synopsis

```
void bae_msgprogressterm(
      );
```

Description

The **bae_msgprogressterm** terminates the previously with **bae_msgprogressrep** activated BAE progress display.

See also

Function **bae_msgprogressrep**.

bae_mtpsize - Get BAE popup display area dimensions (STD)

Synopsis

```
void bae_mtpsize(
      & int;                  // Returns popup menu text columns
      & int;                  // Returns popup menu text rows
      );
```

Description

The **bae_mtpsize** function retrieves the size of the graphic workarea available for displaying popup menus or toolbars defined with **bae_popshow** and/or **bae_settbsize**. The display area width is returned with the popup menu text columns parameter, whilst the display area height is returned with the popup menu text rows parameter. The **bae_charsize** function can be utilized to convert these values to standard length units.

See also

Functions **bae_charsize**, **bae_popshow**, **bae_twsize**, **bae_settbsize**.

bae_nameadd - Add BAE name selection list element (STD)

Synopsis

```
int bae_nameadd(              // Name list index/ID or (-1) on error
      string;                 // Name
      string;                 // Date string
      string;                 // Date sort string
      int;                    // Sort mode:
                              //    0 = append (no sorting)
                              //    1 = sort alphanumerically
                              //    2 = sort numerically
                              //    3 = sort by date
                              //    4 = sort for ID creation
      );
```

Description

The **bae_nameadd** function adds a name entry to the BAE name selection list. This list is used by the **bae_askname** function for activating name selection dialogs. The **bae_nameclr** function can or/should be used prior to the first **bae_nameadd** call to clear any previously stored entries from the name selection list. The function returns a name list index/ID or (-1) on error.

See also

Functions **bae_askname**, **bae_nameclr**, **bae_nameget**.

bae_nameclr - Clear BAE name selection list (STD)

Synopsis

```
void bae_nameclr(
      );
```

Description

The **bae_nameclr** function clears the name list currently defined with **bae_nameadd**.

See also

Functions **bae_askname**, **bae_nameadd**, **bae_nameget**.

bae_nameget - Get BAE name selection list element (STD)

Synopsis

```
int bae_nameget(              // Status
      int;                    // Name list index
      & string;               // Returns name
      & string;               // Returns date string
      & string;               // Returns date sort string
      & string;               // Returns comment
      & int;                  // Returns name entry count or name ID
      );
```

Description

The **bae_nameget** can be used to query BAE name selection list entries defined by **bae_nameadd**. The function returns zero if the query was successfully or nonzero otherwise.

See also

Functions **bae_askname**, **bae_nameadd**, **bae_nameclr**.

bae_numstring - Create numeric string (STD)

Synopsis

```
string bae_numstring(         // Returns numeric string
      double;                 // Input value
      int [0,[;               // Maximum precision
      );
```

Description

The **bae_numstring** converts the input value into a string with the specified precision (number of decimal points.

bae_peekiact - BAE interaction queue query (STD)

Synopsis

```
int bae_peekiact(             // Returns pending interaction status/type:
                              //    0 = No automatic interaction pending
                              //    1 = Mouse interaction pending
                              //    2 = Text interaction pending
                              //    3 = Menu interaction pending
                              //    4 = Wait key interaction pending
      );
```

Description

The **bae_peekiact** can be used to check whether an automatic interaction is pending in the current interaction queue. The function returns a type code for the next automatic interaction queue item, or zero if no automatic interaction is pending.

See also

Functions **bae_storekeyiact**, **bae_storemenuiact**, **bae_storemouseiact**, **bae_storetextiact**.

bae_plainmenutext - BAE menu item text conversion (STD)

Synopsis

```
string bae_plainmenutext(     // Returns answer string
      string;                 // Menu item string
      );
```

Description

The **bae_plainmenutext** function strips all special characters for menu separator line specifications (**%**), key accelerators (**&**), etc. from the specified menu item string. The resulting plain menu item string is returned as function result.

See also

Functions **bae_defmenutext**, **bae_getmenutext**.

bae_planddbclass - Get BAE element DDB class code (STD)

Synopsis

```
int bae_planddbclass(         // Returns BAE element DDB class code (STD1)
      );
```

Description

The **bae_planddbclass** functions returns the DDB class code of the currently loaded BAE element. Possible **Schematic Editor** return values are 800 for SCM sheet, 801 for SCM symbol, 802 for SCM marker or 803 for SCM label. Possible BAE Layout return values are 100 for PCB layout, 101 for layout part, 102 for layout padstack or 103 for layout pad. Possible **IC Design** return values are 1000 for IC layout, 1001 for IC cell and 1001 for IC pin. The function returns (-1) if no element is loaded.

bae_planename - Get BAE element name (STD)

Synopsis

```
string bae_planename(         // Returns BAE element name
      );
```

Description

The **bae_planename** function returns the element name of the currently loaded BAE element or an empty string if no element is currently loaded.

See also

Function **bae_plansename**.

bae_planfname - Get BAE element file name (STD)

Synopsis

```
string bae_planfname(         // Returns BAE element file name
      );
```

Description

The **bae_planfname** function returns file name of the currently loaded BAE element or and empty string if no element is currently loaded.

See also

Functions **bae_plansfname**, **bae_setplanfname**.

bae_plannotsaved - Get BAE element not saved flag (STD)

Synopsis

```
int bae_plannotsaved(         // Returns BAE element not saved flag
      );
```

Description

The **bae_plannotsaved** function checks whether the currently loaded BAE element has been saved. The functions zero if the element has been saved or 1 if changes have been made since the last save.

bae_plansename - Get BAE destination element name (STD)

Synopsis

```
string bae_plansename(        // Returns BAE destination element name
      );
```

Description

The **bae_plansename** function returns the destination element name of the currently loaded BAE element or an empty string if no element is currently loaded. This function should be used for element name queries during Save As operations.

See also

Function **bae_planename**.

bae_plansfname - Get BAE destination element file name (STD)

Synopsis

```
string bae_plansfname(        // Returns BAE destination element file name
      );
```

Description

The **bae_plansfname** function returns the destination file name of the currently loaded BAE element or an empty string if no element is currently loaded. This function should be used for element file name queries during Save As operations.

See also

Function **bae_planfname**.

bae_planwslx - Get BAE element left workspace boundary (STD)

Synopsis

```
double bae_planwslx(          // Returns left workspace boundary (STD2)
      );
```

Description

The **bae_planwslx** function returns the left (lower X) workspace boundary of the currently loaded BAE element.

bae_planwsly - Get BAE element lower workspace boundary (STD)

Synopsis

```
double bae_planwsly(          // Returns lower workspace boundary (STD2)
      );
```

Description

The **bae_planwsly** function returns the lower (lower Y) workspace boundary of the currently loaded BAE element.

bae_planwsnx - Get BAE element origin X coordinate (STD)

Synopsis

```
double bae_planwsnx(          // Returns origin X coordinate (STD2)
      );
```

Description

The **bae_planwsnx** function returns the origin X coordinate of the currently loaded BAE element.

bae_planwsny - Get BAE element origin Y coordinate (STD)

Synopsis

```
double bae_planwsny(          // Returns origin Y coordinate (STD2)
      );
```

Description

The **bae_planwsny** function returns the origin Y coordinate of the currently loaded BAE element.

bae_planwsux - Get BAE element right workspace boundary (STD)

Synopsis

```
double bae_planwsux(          // Returns right workspace boundary (STD2)
      );
```

Description

The **bae_planwsux** function returns the right (upper X) workspace boundary of the currently loaded BAE element.

bae_planwsuy - Get BAE element upper workspace boundary (STD)

Synopsis

```
double bae_planwsuy(          // Returns upper workspace boundary (STD2)
      );
```

Description

The **bae_planwsuy** function returns the upper (upper Y) workspace boundary of the currently loaded BAE element.

bae_popareachoice - Define choice field in active BAE popup menu (STD)

Synopsis

```
int bae_popareachoice(        // Returns status
      double [0.0,[;          // From row number
      double [0.0,[;          // From column number
      double [0.0,[;          // To row number
      double [0.0,[;          // To column number
      string;                 // Selection return string
      );
```

Description

The **bae_popareachoice** function defines a rectangular selection field in the currently active popup menu area. The active popup menu area is selected with the **bae_popsetarea** function and can be either the standard popup menu area to be defined and/or displayed with **bae_popshow** or the toolbar menu area to be defined and/or displayed with **bae_settbsize**. The selection area is defined with the text row and column range function parameters, whereas row zero refers to the top of the popup area and column zero refers to the left of the popup area. The **bae_tbsize** function can be used to retrieve the current toolbar dimensions. The **bae_charsize** function can be utilized to convert text coordinates to standard length units. The selection return string parameter defines the answer string to be returned with the **bae_readtext** function on selections in the defined choice area. The function returns (-1) on missing and/or invalid parameter specifications, 1 if a button was created, or zero otherwise.

See also

Functions **bae_charsize**, **bae_popcolbar**, **bae_popcolchoice**, **bae_popsetarea**, **bae_popshow**, **bae_poptext**, **bae_poptextchoice**, **bae_readtext**, **bae_settbsize**, **bae_tbsize**.

bae_popcliparea - Define clipping area in active BAE popup menu (STD)

Synopsis

```
void bae_popcliparea(
      int [0,1];              // Clipping enable flag:
                              //    0 = disable clipping
                              //    1 = enable clipping
      double [0.0,[;          // From row number
      double [0.0,[;          // From column number
      double [0.0,[;          // To row number
      double [0.0,[;          // To column number
      );
```

Description

The **bae_popcliparea** function enables (clipping enable flag nonzero) and/or disables (clipping enable flag zero) clipping in the currently active BAE popup menu area. The active popup menu area is selected with the **bae_popsetarea** function and can be either the standard popup menu area to be defined and/or displayed with **bae_popshow** or the toolbar menu area to be defined and/or displayed with **bae_settbsize**. Clipping is most useful for restricting graphic output to certain areas when displaying polygons in popup areas and/or toolbars using the **bae_popdrawpoly** function. The clipping area is defined with the text row and column range function parameters, whereas row zero refers to the top of the popup area and column zero refers to the left of the popup area. The **bae_charsize** function can be utilized to convert text coordinates to standard length units.

See also

Functions **bae_charsize**, **bae_popdrawpoly**, **bae_popsetarea**, **bae_popshow**, **bae_settbsize**.

bae_popclrtool - Clear BAE toolbar popup area (STD)

Synopsis

```
void bae_popclrtool(
      );
```

Description

The **bae_popclrtool** function can be used to clear and/or deactivate all elements from the current toolbar area.

See also

Functions **bae_popsetarea**, **bae_settbsize**.

bae_popcolbar - Define BAE popup menu color bar display (STD)

Synopsis

```
int bae_popcolbar(            // Returns nonzero on error
      double [0.0,[;          // From row number
      double [0.0,[;          // From column number
      double [0.0,[;          // To row number
      double [0.0,[;          // To column number
      int [0,[;               // Color value (STD18)
      );
```

Description

The **bae_popcolbar** function defines a non-selectable color bar in the popup menu previously activated by **bae_popshow**. The color bar size and position emerge from the specified row and column parameters with the zero coordinate [0,0] referring to the top left corner of the popup area. The color bar's color is specified with the given color value parameter. The function returns nonzero on invalid parameter specifications.

See also

Functions **bae_popcolchoice**, **bae_popshow**, **bae_poptext**, **bae_poptextchoice**.

bae_popcolchoice - Define BAE popup menu color bar selector (STD)

Synopsis

```
int bae_popcolchoice(         // Returns nonzero on error
      double [0.0,[;          // From row number
      double [0.0,[;          // From column number
      double [0.0,[;          // To row number
      double [0.0,[;          // To column number
      int [0,[;               // Color value (STD18)
      string;                 // Answer string to be returned
      );
```

Description

The **bae_popcolchoice** function defines a mouse-selectable color bar in the popup menu previously activated by **bae_popshow**. The color bar size and position emerge from the specified row and column parameters with the zero coordinate [0,0] referring to the top left corner of the popup area. The color bar's color is specified with the given color value parameter. The function returns nonzero on invalid parameter specifications. The selection of a color bar defined with **bae_popcolchoice** can be enabled through a call to the **bae_readtext** function. The **bae_readtext** return value is then set to the **bae_popcolchoice** answer string parameter.

See also

Functions **bae_popcolbar**, **bae_popshow**, **bae_poptext**, **bae_poptextchoice**, **bae_readtext**.

bae_popdrawpoly - Display/draw polygon in active BAE popup menu (STD)

Synopsis

```
int bae_popdrawpoly(          // Returns status
      int [0,[;               // Polygon color (STD18)
      int [0,3];              // Polygon drawing mode (STD19)
      int [0,15];             // Polygon fill mode (STD20)
      );
```

Description

The **bae_popdrawpoly** function draws the internal polygon defined with the **bae_storepoint** function to the currently active popup menu area using the color, drawing mode and filling properties specified with the corresponding function parameters. The active popup menu area is selected with the **bae_popsetarea** function and can be either the standard popup menu area to be defined and/or displayed with **bae_popshow** or the toolbar menu area to be defined and/or displayed with **bae_settbsize**. The polygon display can be restricted to a certain popup menu region using the **bae_popcliparea** function. The **bae_popareachoice** function can be utilized for allowing the polygon display or certain parts of it to be selected with the **bae_readtext** function. The **bae_setpopdash** function can be used to set the base length for dash lines. The **bae_clearpoints** function can be used to clear the internal polygon data to prepare for subsequent polygon definitions. The function returns zero if the polygon was drawn without errors or nonzero otherwise.

See also

Functions **bae_clearpoints**, **bae_dialbmpalloc**, **bae_popareachoice**, **bae_popcliparea**, **bae_popcolbar**, **bae_popcolchoice**, **bae_popdrawtext**, **bae_popsetarea**, **bae_popshow**, **bae_readtext**, **bae_setpopdash**, **bae_settbsize**, **bae_storepoint**.

bae_popdrawtext - Display/draw text in active BAE popup menu (STD)

Synopsis

```
int bae_popdrawtext(          // Returns status
      int;                    // Text row
      int;                    // Text column
      int [0,[;               // Text display color (STD18)
      int [0,[;               // Text background color (STD18)
      string;                 // Text string
      );
```

Description

The **bae_popdrawtext** function displays the given text string at the specified row and column coordinates in the currently active popup menu area using the colors specified for text display and background. The active popup menu area is selected with the **bae_popsetarea** function and can be either the standard popup menu area to be defined and/or displayed with **bae_popshow** or the toolbar menu area to be defined and/or displayed with **bae_settbsize**. The **bae_charsize** function can be utilized to convert text coordinates to standard length units. The **bae_popareachoice** function can be utilized for allowing the text display or certain parts of it to be selected with the **bae_readtext** function. The function returns zero if the text was displayed without errors or nonzero otherwise.

See also

Functions **bae_charsize**, **bae_dialbmpalloc**, **bae_popareachoice**, **bae_popdrawpoly**, **bae_popsetarea**, **bae_popshow**, **bae_readtext**, **bae_settbsize**.

bae_popmouse - Get BAE popup/toolbar mouse position (STD)

Synopsis

```
void bae_popmouse(
      & double;               // Returns mouse X coordinate/column
      & double;               // Returns mouse Y coordinate/row
      & int;                  // Returns mouse state:
                              //    Bit 1 : Left mouse button pressed
                              //    Bit 2 : Right mouse button pressed
                              //    Bit 3 : Middle mouse button pressed
      );
```

Description

The **bae_wsmouse** function retrieves the current mouse coordinates (column and row) with the popup and/or toolbar area. The mouse state return parameter can be used to designate which mouse buttons are currently pressed.

See also

Function **bae_wsmouse**.

bae_poprestore - Restore BAE popup menu area (STD)

Synopsis

```
void bae_poprestore(
      );
```

Description

The **bae_poprestore** function releases and restores the popup graphic work area previously defined with **bae_popshow**.

See also

Function **bae_popshow**.

bae_popsetarea - Clear BAE toolbar popup area (STD)

Synopsis

```
void bae_popsetarea(
      int                     // Popup area code:
                              //    0 = Popup menu area
                              //    1 = Toolbar area
                              //    2..31 = Dialog bitmap area
      );
```

Description

The **bae_popsetarea** function activates the toolbar area (popup area code 1), the standard popup menu area (popup area code 2) or a dialog bitmap area (popup area code and/or popup bitmap number 2 through 31) for subsequent popup operations such as **bae_popareachoice**, **bae_popcliparea**, **bae_popdrawpoly** or **bae_popdrawtext**. The standard popup menu area is defined and/or displayed with the **bae_popshow** function. The toolbar popup menu area is defined and/or displayed with the **bae_settbsize** function. On default, the standard popup menu area is selected for popup operations.

See also

Functions **bae_dialbmpalloc**, **bae_popareachoice**, **bae_popcliparea**, **bae_popdrawpoly**, **bae_popdrawtext**, **bae_popshow**, **bae_settbsize**.

bae_popshow - Activate BAE popup menu (STD)

Synopsis

```
int bae_popshow(              // Returns nonzero on error
      & double [0.0,[;        // Popup rows count
      & double [0.0,[;        // Popup columns count
      & double [0,1.0];       // Popup percentage left boundary
      & double [0,1.0];       // Popup percentage bottom boundary
      & double [0,1.0];       // Popup percentage right boundary
      & double [0,1.0];       // Popup percentage top boundary
      );
```

Description

The **bae_popshow** function defines and generates a popup menu with the given parameters. The popup menu position and size emerge from the specified popup boundary parameters; the given values are interpreted relatively to the available graphic work area size; the boundary values range from 0.0 (minimum and/or left/lower boundary) to 1.0 (maximum and/or right/upper boundary). The **bae_mtpsize** function can be used for determining the number of columns and/or rows available for popup display. The requested popup size is specified with the popup row and column count input parameters. These parameters are automatically re-calculated by **bae_popshow** to denote the row and column counts actually visible in the given popup area boundaries. The function returns nonzero on invalid and/or inconsistent parameter specifications. The popup menu colors for texts, background and frame are taken from the corresponding BAE setup menu color values stored with the **BSETUP** utility program. The **bae_popareachoice**, **bae_popcolbar**, **bae_popcolchoice**, **bae_popdrawpoly**, **bae_popdrawtext**, **bae_poptext** and **bae_poptextchoice** functions can be used after calling **bae_popshow** to define selectable and non-selectable color bars, text buttons, graphic display and selection areas in the popup menu. The popup menu item selections can be enabled by calling the **bae_readtext** function.

Warning

The **bae_poprestore** function must be used to release and/or restore any graphic work area occupied by a popup menu.

See also

Functions **bae_mtpsize**, **bae_popareachoice**, **bae_popcliparea**, **bae_popcolbar**, **bae_popcolchoice**, **bae_popdrawpoly**, **bae_popdrawtext**, **bae_poprestore**, **bae_popsetarea**, **bae_poptext**, **bae_poptextchoice**, **bae_readtext**, **bae_settbsize**, and BAE utility program **BSETUP**.

bae_poptext - Define BAE popup menu text display (STD)

Synopsis

```
int bae_poptext(              // Returns nonzero on error
      double [0.0,[;          // Text row number
      double [0.0,[;          // Text column number
      string;                 // Text string
      );
```

Description

The **bae_poptext** function defines a non-selectable text in the popup menu previously activated by **bae_popshow**. The text display position emerges from the specified row and column parameters with the zero coordinate [0,0] referring to the top left corner of the popup area. The text to be displayed is specified with the given text string parameter. The function returns nonzero on invalid parameter specifications.

See also

Functions **bae_popcolbar**, **bae_popcolchoice**, **bae_popshow**, **bae_poptextchoice**.

bae_poptextchoice - Define BAE popup menu text selector (STD)

Synopsis

```
int bae_poptextchoice(        // Returns nonzero on error
      double [0.0,[;          // Text row number
      double [0.0,[;          // Text column number
      string;                 // Text string
      string;                 // Answer string to be returned
      );
```

Description

The **bae_poptextchoice** function defines a mouse-selectable text in the popup menu previously activated by **bae_popshow**. The text display position emerges from the specified row and column parameters with the zero coordinate [0,0] referring to the top left corner of the popup area. The text to be displayed is specified with the given text string parameter. The function returns nonzero on invalid parameter specifications. The selection of a text button defined with **bae_poptextchoice** can be enabled through a call to the **bae_readtext** function. The **bae_readtext** return value is then set to the **bae_poptextchoice** answer string parameter.

See also

Functions **bae_popcolbar**, **bae_popcolchoice**, **bae_popshow**, **bae_poptext**, **bae_readtext**.

bae_postprocess - Run BAE postprocess (STD)

Synopsis

```
void bae_postprocess(
      );
```

Description

The **bae_postprocess** function runs a BAE postprocess on the currently loaded element. On layout level, the BAE postprocess forces a connectivity update and a design rule check.

bae_progdir - Get BAE program directory path name (STD)

Synopsis

```
string bae_progdir(           // Returns BAE program directory path
      );
```

Description

The **bae_progdir** function returns the currently valid BAE programs directory path name. This information is useful for accessing data in the BAE programs directory (e.g., system color tables, aperture tables, etc.).

bae_prtdialog - Print string to BAE dialogue line (STD)

Synopsis

```
void bae_prtdialog(
      string;                 // Message string
      );
```

Description

The **bae_prtdialog** function displays the specified message string in the BAE status line.

See also

Function **perror**.

bae_querydist - Query BAE point to polygon distance (STD)

Synopsis

```
int bae_querydist(            // Returns status
      double;                 // Query X coordinate (STD2)
      double;                 // Query Y coordinate (STD2)
      & double;               // Query distance return value (STD2)
      );
```

Description

The **bae_querydist** gets the distance between the point specified through the X and Y coordinate parameters and the internal distance query polygon previously stored with the **bae_storedistpoly** function. The resulting distance is returned with the last function parameter. The distance value is positive for points outside the distance query polygon and negative for points inside the distance query polygon. The function returns zero if no error occurred or nonzero on error (missing distance query polygon).

See also

Function **bae_storedistpoly**.

bae_readedittext - BAE text input/display (STD)

Synopsis

```
string bae_readedittext(      // Returns answer string
      string;                 // Prompt string
                              //    ! prefix: Multiline text input/edit
                              //    otherwise: Single line text input
      string;                 // Default answer string
      int [0,[;               // Maximum input string length
      );
```

Description

The **bae_readedittext** function activates a text edit dialog with the prompt string being displayed in the dialog title bar. A multiline edit window instead of a single-line text edit box is activated if the first prompt string character is a quotation mark (**!**). The size of the dialog window can be changed if in multiline edit mode. The text edit box contents is initialyzed with the specified default answer string. The edit dialog provides an OK button for accepting the input and an Abort button for cancelling the text input operation. The maximum input/aswer string length is limited through the third function parameter. Abort causes the default answer string instead of the current edit text to be returned to the caller. The multiline text edit dialog provides additional buttons for loading contents from selectable input files into the text edit window (Load) and for saving the current edit text to output files (Save).

Warning

BAE Demo does not provide the option for saving the edit text and also deactivates the Windows context menu function for copying the edit text onto the clipboard.

See also

Function **bae_readtext**.

bae_readtext - BAE text input with popup menu (STD)

Synopsis

```
string bae_readtext(          // Returns answer string
      string;                 // Prompt string
      int;                    // Maximum input string length
      );
```

Description

The **bae_readtext** function asks the user for a string value. The required interaction is indicated by the given prompt string. The user input string value is passed with the function return value. The maximum keyboard input string length is specified with the corresponding function parameter. Mouse click input text is returned if previously enabled with the **bae_setmousetext** function; such mouse click default text input is disabled through the **bae_readtext** call. Popup menu item selections are simultaneously enabled by **bae_readtext** if a popup menu was previously defined with the **bae_popshow** function and mouse click text input is disabled. Possible popup menu selections are defined through **bae_popcolchoice** and/or **bae_poptextchoice**. The function return value is retrieved from the corresponding answer string parameters.

Warning

The **bae_readtext** function deactivates all selection elements previously defined with **bae_popcolchoice**, **bae_poptextchoice** or **bae_setmousetext**.

See also

Functions **bae_popcolchoice**, **bae_popshow**, **bae_poptextchoice**, **bae_readedittext**, **bae_setmousetext**.

bae_redefmainmenu - BAE main menu redefinition start (STD)

Synopsis

```
int bae_redefmainmenu(        // Returns status
      );
```

Description

The **bae_redefmainmenu** function starts the (re)definition of the main menu of the currently active BAE module. The function returns (-1) on error or zero otherwise. After calling **bae_redefmainmenu**, the **bae_defmenu** (to be terminated with **bae_endmenu**) should be applied for defining the main menu entries. Then the **bae_defselmenu** (to be terminated with **bae_endmenu**) can be applied for configuring the submenus. Between **bae_defmenu** or **bae_defselmenu** calls and the call to **bae_endmenu** the **bae_defmenutext** function should be applied for defining the menu entries. The menu definition initiated with **bae_redefmenu** *must* be terminated by a call to the **bae_endmainmenu** function. The **bae_resetmenuprog** function can be used to reset *all* menu assignments, thus restoring the default menu configuration of the currently active BAE module.

See also

Functions **bae_defmenu**, **bae_defmenuprog**, **bae_defmenutext**, **bae_defselmenu**, **bae_endmainmenu**, **bae_endmenu**, **bae_redefmenu**, **bae_resetmenuprog**.

bae_redefmenu - Redefine BAE menu item (STD)

Synopsis

```
int bae_redefmenu(            // Returns status
      int [0,999];            // Menu number
      int [0,99];             // Menu line
      string;                 // Menu text
      int;                    // BAE menu function code (STD4)
      int;                    // Menu entry processing key:
                              //    8000000h = always available
                              //    7FFFFFFh = available for each element type
                              //    else     = (combined) DDB class processing key
      );
```

Description

The **bae_redefmenu** function assigns the specified menu text and the BAE menu function to the given menu entry. The menu number specifies the number of the main menu, whilst the menu line designates the position in the according submenu. The menu entry processing key activates ghost menu configurations. The processing key is a coded integer value as retrieved and/or defined using the **bae_getclassbitfield** and **bae_getmenubitfield** functions (the hex value 80000000h can be used to allow for application in any case). The function returns (-1) on error or zero otherwise. The **bae_resetmenuprog** function can be used to reset *all* menu assignments, thus restoring the default menu configuration of the currently active BAE module.

See also

Functions **bae_defmenuprog**, **bae_getclassbitfield**, **bae_getmenubitfield**, **bae_redefmenu**, **bae_resetmenuprog**.

bae_resetmenuprog - Reset BAE menu definitions (STD)

Synopsis

```
void bae_resetmenuprog(
      );
```

Description

The **bae_resetmenuprog** function resets *all* key and menu assignments, thus restoring the default menu configuration of the currently active BAE module.

See also

Functions **bae_deffuncprog**, **bae_defkeyprog**, **bae_defmenu**, **bae_defmenuprog**, **bae_defmenutext**, **bae_defselmenu**, **bae_redefmainmenu**, **bae_redefmenu**.

bae_sendmsg - Send BAE HighEnd message (STD/HighEnd)

Synopsis

```
int bae_sendmsg(              // Returns status
      string;                 // Message text string
      int [0,1];              // Send to projects members only flag
      );
```

Description

The **bae_sendmsg** function is only available in **BAE HighEnd**. The function return value is nonzero if called outside **BAE HighEnd** or on invalid parameter specifications. **bae_sendmsg** sends the specified message string to the **BAE HighEnd** message system. The second parameter controls whether the message should be send to all other BAE modules started in the same session or only to those BAE program instances which are currently processing elements from the same DDB file. A **BAE HighEnd** session is started with a BAE call and includes any other BAE program instance subsequently started with the New Task function from the BAE main menu or with the Next SCM Window function from the **Schematic Editor**. Each **BAE HighEnd** module receiving a message automatically activates the **User Language** program named **bae_msg**. If **bae_msg** is not available, the system tries to start an interpreter-specific **User Language** program (**scm_msg** in the **Schematic Editor**, **ged_msg** in the **Layout Editor**, **ar_msg** in the **Schematic Editor**, etc.). The **bae_getmsg** function must be used in the ***_msg** **User Language** program to retrieve pending messages. Pending messages are only available during the execution of the ***_msg** **User Language** program, i.e., any message not retrieved by ***_msg** is lost. The message text string can be used to trigger certain actions in the destination program instances.

See also

Function **bae_getmsg**.

bae_setanglelock - Set BAE angle lock flag (STD)

Synopsis

```
int bae_setanglelock(         // Returns status
      int [0,1];              // Required angle lock flag (STD9)
      );
```

Description

The **bae_setanglelock** function sets the current BAE angle lock mode (0=angle unlocked, 1=angle locked). The function returns nonzero if an invalid angle lock flag value has been specified.

See also

Function **bae_getanglelock**.

bae_setbackgrid - Set BAE display grid (STD)

Synopsis

```
int bae_setbackgrid(          // Returns status
      double [0.0,[;          // Required X display grid (STD2)
      double [0.0,[;          // Required Y display grid (STD2)
      );
```

Description

The **bae_setbackgrid** function sets the BAE X/Y display grid values. Zero grid values refer to switched-off grids. The function returns nonzero, if invalid grid values are specified.

See also

Function **bae_getbackgrid**.

bae_setclipboard - Store text string to (Windows) clipboard (STD)

Synopsis

```
int bae_setclipboard(         // Returns status
      string;                 // Text string
      );
```

Description

The **bae_setclipboard** function saves the specified text string to the **Windows** clipboard. The function returns zero if done, or nonzero on error.

bae_setcolor - Set BAE color value (STD)

Synopsis

```
int bae_setcolor(             // Returns status
      int;                    // Display item type (SCM1|LAY9|ICD9)
      int;                    // Color value (STD18)
      );
```

Description

The **bae_setcolor** function sets the color for the given display item type to the specified value. The display item type value must be set according to the currently active **User Language Interpreter** environment. The function returns nonzero on error.

Warning

To avoid redundant screen redraws at the color table redefinition, the **bae_setcolor** system function does not perform a screen redraw. I.e., it is the responsibility of the caller to trigger screen redraws as required at the end of a color table definition sequence.

See also

Function **bae_getcolor**.

bae_setcoorddisp - Set BAE coordinate display mode (STD)

Synopsis

```
int bae_setcoorddisp(         // Returns status
      int [0,1];              // Required coordinate display mode (STD7)
      );
```

Description

The **bae_setcoorddisp** function sets the current BAE coordinate display mode. The display mode is passed with the parameter, where 0 designates mm display units (micrometer in **IC Design**) and 1 designates Inch display units (mil units in **IC Design**). The function returns nonzero for invalid display mode parameters or zero otherwise.

See also

Function **bae_getcoorddisp**.

bae_setdblpar - Set BAE double parameter (STD)

Synopsis

```
int bae_setdblpar(            // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = maximum dialog box width
                              //    1 = maximum dialog box height
                              //    2 = display zoom factor
                              //    3 = Rubberband corner radius (STD2)
                              //    4 = Rubberband X vector coordinate (STD2)
                              //    5 = Rubberband Y vector coordinate (STD2)
                              //    6 = fixed X pick coordinate (STD2)
                              //    7 = fixed Y pick coordinate (STD2)
                              //  [ 8 = System parameter - no write access ]
                              //  [ 9 = System parameter - no write access ]
                              //    10 = Screen pick aperture (STD2)
                              //    11 = Element selection preview area
                              //          relative size [0.05, 0.95]
                              //  [ 12 = System parameter - no write access ]
                              //  [ 13 = System parameter - no write access ]
      double;                 // Parameter value
      );
```

Description

The **bae_setdblpar** function is used to set **Bartels AutoEngineer** `double` system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **bae_getdblpar** function can be used to query parameter values set with **bae_setdblpar**.

See also

Functions **bae_getdblpar**, **bae_getintpar**, **bae_getstrpar**, **bae_setintpar**, **bae_setstrpar**.

bae_setgridlock - Set BAE grid lock flag (STD)

Synopsis

```
int bae_setgridlock(          // Returns status
      int [0,1];              // Required grid lock flag (STD8)
      );
```

Description

The **bae_setgridlock** function sets the current BAE grid lock mode (0=grid unlocked, 1=grid locked). The function returns nonzero if an invalid grid lock flag value was specified.

See also

Function **bae_getgridlock**.

bae_setgridmode - Set BAE grid dependency mode (STD)

Synopsis

```
int bae_setgridmode(          // Returns status
      int [0,255];            // Automatic gid setting mode:
                              //    0x01: input grid = 0.25 × display grid
                              //    0x02: input grid = 0.50 × display grid
                              //    0x04: input grid = 1.00 × display grid
                              //    0x08: input grid = 2.00 × display grid
                              //    0x10: display grid = 0.25 × input grid
                              //    0x20: display grid = 0.50 × input grid
                              //    0x40: display grid = 1.00 × input grid
                              //    0x80: display grid = 2.00 × input grid
      );
```

Description

The **bae_setgridmode** function sets the BAE grid dependency mode. The function returns nonzero if an invalid grid dependency mode was specified.

See also

Function **bae_getgridmode**.

bae_setinpgrid - Set BAE input grid (STD)

Synopsis

```
int bae_setinpgrid(           // Returns status
      double [0.0,[;          // Required X input grid (STD2)
      double [0.0,[;          // Required Y input grid (STD2)
      );
```

Description

The **bae_setinpgrid** function sets the BAE X/Y input grid values. Zero grid values refer to switched-off grids. The function returns nonzero if invalid grid values are specified.

See also

Function **bae_getinpgrid**.

bae_setintpar - Set BAE integer parameter (STD)

Synopsis

```
int bae_setintpar(              // Returns status
      int [0,[;                 // Parameter type/number:
                                //    0 = Input coordinate range check mode:
                                //        0 = range check enabled
                                //        1 = range check disabled
                                //    1 = Module change autosave mode:
                                //        0 = autosave without prompt
                                //        1 = prompt before autosave
                                //    2 = Display disable mode:
                                //        0 = display enabled check
                                //        1 = display disabled
                                //  [ 3 = Menu/mouse mode: ]
                                //  [     system parameter write-protected ]
                                //    4 = Workspace text color mode:
                                //        0 = standard colors
                                //        1 = inverted standard colors
                                //        2 = workspace related colors
                                //    5 = Load display mode:
                                //        0 = display overview after load
                                //        1 = handle load display in bae_load
                                //    6 = File dialog view:
                                //        0 = old BAE style file selection
                                //        1 = Explorer style default view
                                //        2 = Explorer style list view
                                //        3 = Explorer style details
                                //        4 = Explorer style small icons
                                //        5 = Explorer style large icons
                                //        6 = Use default style and size
                                //    7 = Element selection box mode:
                                //        0 = display name only
                                //        1 = display name and date
                                //    8 = Element selection sort mode:
                                //        0|1 = sort by name
                                //        2 = sort numerically
                                //        3 = sort by date
                                //    9 = Inverted placement visibility flag:
                                //        0 = placed elements visible
                                //        1 = unplaced elements visible
                                //  [ 10 = Last file system error: ]
                                //  [     system parameter write-protected ]
                                //    11 = Command history disable flag:
                                //        0 = Command history enabled
                                //        1 = Command history disabled
                                //    12 = Popup menu mouse warp mode:
                                //        0 = No popup menu mouse warp
                                //        1 = First popup menu entry mouse warp
                                //        2 = Preselected menu entry mouse warp
                                //        +4 = Mouse position restore warp
                                //        +8 = Element pick position warp
                                //    13 = Save disable flag:
                                //        0 = save enabled
                                //        1 = save disabled
                                //    14 = Mouse rectangle min. size:
                                //        ]0,[ = min. rectangle size
                                //    15 = Menu selection mouse info display flag:
                                //        0 = no continuous info display
                                //        1 = continuous info display
                                //    16 = Next dialog box id
                                //    17 = Last created tooltip id
                                //    18 = Polygon drop count
                                //    19 = Polygon check disabled flag:
                                //        0 = Enable polygon check
                                //        1 = Disable polygon check
                                //    20 = Cursor key grid mode:
                                //        0 = Input grid
                                //        1 = Pixel grid
                                //    21 = Unsaved plan flag
```

```
//    22 = Element batch load mode:
//         0 = no batch load
//         1 = Batch load
//         2 = batch load, restore zoom window
//    23 = Grid lines display:
//         0 = Dot grid
//         1 = Line grid
//    24 = Display mirroring flag
//    25 = Input grid display flag
//    26 = Mouse function repeat mode:
//         0 = Repeat menu function
//         +1 = Repeat keystroke function
//         +2 = Repeat context menu function
//  [ 27 = Maximum Undo/Redo count: ]
//  [     system parameter write-protected ]
//    28 = Menu tree view mode:
//         0 = No menu tree view window
//         1 = Left attached menu tree view window
//         2 = Right attached menu tree view window
//    29 = Menu tree view pixel width
//    30 = Message history disabled flag
//    31 = Element load message mode:
//         0 = Standard message
//         1 = User message
//         2 = User error message
//    32 = Pick marker display mode:
//         0 = Circle marker
//         1 = Diamond marker
//    33 = Mouse drag status:
//         0 = No mouse drag
//         1 = Request mouse drag
//         2 = Mouse dragged
//         3 = Request mouse drag release
//    34 = Menu function repeat request flag
//    35 = Function aborted flag
//    36 = Plan selection preview flag
//    37 = File error display mode:
//         0 = Status message only
//         1 = Confirm message box
//    38 = Element selection reference display mode:
//         0 = Display project file references
//         1 = Display library file references
//    39 = Mouse double-click mode:
//         0 = Map double-click and
//              select 0 to right mouse button
//         1 = Ignore double-click
//         2 = Map double-click to right mouse button
//    40 = Mouse pick double-click mode:
//         0 = Map double-click to right mouse button
//         1 = Ignore double-click
//  [ 41 = Dialog control support flags: ]
//  [     system parameter write-protected ]
//    42 = Progress box display mode:
//         0 = No progress box window
//         1 = Display progress window
//    43 = Progress box abort request flag:
//    44 = Middle mouse button disable flag:
//  [ 45 = Current undo items count: ]
//  [     system parameter write-protected ]
//  [ 46 = File drag and drop operation flag.: ]
//  [     system parameter write-protected ]
//    47 = Autoraise BAE window flag
//  [ 48 = Menu function active count ]
//  [     system parameter write-protected ]
//  [ 49 = Internal polygon list point count ]
//  [     system parameter write-protected ]
```

```
                              //    50 = Alternate configuration file priority
                              //    51 = Dialog position save mode:
                              //        0 = Store absolute coordinates
                              //        1 = Store main window relative coordinates
                              //        2 = Store main window monitor
                              //             absolute coordinates
                              //    52 = Message box default button index:
                              //        (-1) = No default (Abort or No)
                              //        0-2 = Default button index
      int;                    // Parameter value
      );
```

Description

The **bae_setintpar** function is used to set **Bartels AutoEngineer** integer system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **bae_getintpar** function can be used to query parameter values set with **bae_setintpar**.

See also

Functions **bae_getdblpar**, **bae_getintpar**, **bae_getstrpar**, **bae_setdblpar**, **bae_setstrpar**.

bae_setmoduleid - Set BAE module id (STD)

Synopsis

```
int bae_setmoduleid(          // Status
      string;                 // Module id
      );
```

Description

The **bae_setmoduleid** sets a name/identification for the currently active BAE program module. The function returns zero if the module id was successfully assigned or nonzero on error.

See also

Function **bae_getmoduleid**.

bae_setmousetext - Set BAE mouse click input text (STD)

Synopsis

```
int bae_setmousetext(         // Returns nonzero on error
      string;                 // Answer text string to be returned
      );
```

Description

The **bae_setmousetext** function enables the specified answer string to be the default mouse click return value for the next **bae_readtext** call. The next call to **bae_readtext** resets the mouse click default text.

See also

Function **bae_readtext**.

bae_setplanfname - Set BAE project file name (STD)

Synopsis

```
int bae_setplanfname(         // Returns status
      string;                 // BAE project file name
      );
```

Description

The **bae_setplanfname** function sets the BAE DDB project file name. The **.ddb** file name extension if a BAE project file name without file name extension is specified. The function returns zero if the BAE project file name was successfully assigned, 1 if parameter specifications are missing and/or invalid or 2 if a BAE project file name is already assigned through the currently loaded element.

See also

Function **bae_planfname**.

bae_setpopdash - Set BAE popup/toolbar polygon dash line parameters (STD)

Synopsis

```
void bae_setpopdash(
      double ]0.0,[;          // Dash base length (STD2)
      double ]-0.5,0.5[;      // Dash relative spacing
      );
```

Description

The **bae_setpopdash** function sets the base length and the relative spacing for dash lines to be created in popup menus and/or toolbars with the **bae_popdrawpoly** function.

See also

Function **bae_popdrawpoly**.

bae_setstrpar - Set BAE string parameter (STD)

Synopsis

```
int bae_setstrpar(            // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = Current element comment text
                              //    1 = Current element specification
                              //  [ 2 = System parameter - no write access ]
                              //  [ 3 = System parameter - no write access ]
                              //    4 = Crosshair info text
                              //    5 = Tooltip text
                              //  [ 6 = System parameter - no write access ]
                              //    7 = Menu text of currently active funktion
                              //    8 = Current menu item element text
                              //    9 = Current element load user message
                              //    10 = Clipboard text string
                              //    11 = Next module call file argument
                              //    12 = Next module call element argument
                              //    13 = Next module call command/type argument
                              //    14 = Last output file name
                              //  [ 15 = System parameter - no write access ]
                              //    16 = Toolbar button character/resource item
                              //  [ 17 = System parameter - no write access ]
                              //  [ 18 = System parameter - no write access ]
                              //    19 = Alternate configuration data directory
                              //    20 = Local data column
                              //    21 = Global data column
      string;                 // Parameter value
      );
```

Description

The **bae_setstrpar** function is used to set **Bartels AutoEngineer** string system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **bae_getstrpar** function can be used to query parameter values set with **bae_setstrpar**.

See also

Functions **bae_getdblpar**, **bae_getintpar**, **bae_getstrpar**, **bae_setdblpar**, **bae_setintpar**.

bae_settbsize - Define/display BAE toolbar area (STD)

Synopsis

```
void bae_settbsize(
      double [0.0,[;          // Toolbar size
      int [0,3];              // Toolbar attachment mode:
                              //    0 = horizontally attached to
                              //         lower workspace boundary
                              //    1 = vertically attached to
                              //         right workspace boundary
                              //    2 = horizontally attached to
                              //         upper workspace boundary
                              //    3 = vertically attached to
                              //         left workspace boundary
      );
```

Description

The **bae_settbsize** function defines and/or displays a toolbar. The first parameter specifies the desired toolbar size depending on the attachment mode, i.e., with vertical attachment the toolbar size is interpreted as column count, whilst with horizontal attachment the toolbar size is interpreted as row count. A zero toolbar size specification can be used to fade out a currently visible toolbar. The dimension of the graphic area available for displaying toolbars can be retrieved with the **bae_mtpsize** function. The **bae_tbsize** function can be used to retrieve the current toolbar dimensions. The **bae_popsetarea** function activates either the toolbar or the standard popup menu for subsequent popup operations. The **bae_popclrtool** function can be used to clear and/or deactivate all elements from the current toolbar area.

See also

Functions **bae_mtpsize**, **bae_popclrtool**, **bae_popsetarea**, **bae_tbsize**.

bae_storecmdbuf - Store BAE command to command history (STD)

Synopsis

```
int bae_storecmdbuf(          // Returns status
      int [0,[;               // Command store mode
                              //    0 : append command to history
      string;                 // Command (sequence) string
      string;                 // Command notification text
      );
```

Description

The **bae_storecmdbuf** function adds the specified command to the current BAE command history. The command store mode is used to specify the location for storing the command in the history and/or whether or how the command history should be reorganized. The command (sequence) string and the command notification text (as displayed in the title bar) are specified through the second and third function parameter. The function returns zero if the assignment was successful or non-zero otherwise.

See also

Function **bae_getcmdbuf**.

bae_storedistpoly - Store internal BAE distance query polygon (STD)

Synopsis

```
int bae_storedistpoly(        // Status
      );
```

Description

The **bae_storedistpoly** function stores the internal polygon created with **bae_clearpoints** and **bae_storepoint** as distance query polygon. The **bae_querydist** can then be used to query the distance between a given point and the distance query polygon. The **bae_cleardistpoly** function can be used to delete the distance query polygon. The function returns zero if no error occurred, (-1) if the internal distance query polygon is already defined or (-2) in case of invalid polygon data.

See also

Functions **bae_cleardistpoly**, **bae_clearpoints**, **bae_querydist**, **bae_storepoint**.

bae_storeelem - Store BAE element (STD)

Synopsis

```
int bae_storeelem(            // Returns status
      string;                 // File name
      string;                 // Element name
      );
```

Description

The **bae_storeelem** function stores the currently loaded BAE element to the specified file (i.e., this function is equivalent to Save Element As function from the File menu). The function returns 0 if the element has been successfully stored, 1 on invalid function parameters, 2 if no BAE element is currently loaded or (-1) on file access errors.

bae_storekeyiact - Store BAE key-press interaction to queue (STD)

Synopsis

```
void bae_storekeyiact(
      int [0,3];              // Automatic interaction mode (STD21)
      int;                    // Key character code (ASCII)
      );
```

Description

The **bae_storekeyiact** function stores a key-press interaction to the interaction queue to be passed to a **bae_callmenu**-activated BAE menu function. The given key character code is automatically passed if the interaction flag is set, otherwise an interactive user input is required.

See also

Functions **bae_peekiact**, **bae_storemenuiact**, **bae_storemouseiact**, **bae_storetextiact**.

bae_storemenuiact - Store BAE menu interaction to queue (STD)

Synopsis

```
void bae_storemenuiact(
      int [0,3];              // Automatic interaction mode (STD21)
      int;                    // Menu line number (0..n-1)
      int [1,3];              // Mouse key (STD17)
      );
```

Description

The **bae_storemenuiact** function stores a menu interaction to the interaction queue to be passed to a **bae_callmenu**-activated BAE menu function. The given menu selection with mouse-click is passed automatically passed if the interaction flag is set, otherwise an interactive user input is required.

See also

Functions **bae_peekiact**, **bae_storekeyiact**, **bae_storemouseiact**, **bae_storetextiact**.

bae_storemouseiact - Store BAE mouse interaction to queue (STD)

Synopsis

```
void bae_storemouseiact(
      int [0,3];              // Automatic interaction mode (STD21)
      double;                 // Mouse X coordinate (STD2)
      double;                 // Mouse Y coordinate (STD2)
      int [0,15];             // Mouse coordinate mode:
                              //    0 = use given coordinates
                              //         with snap to input grid
                              //    1 = use old mouse coordinates
                              //    2 = use given coordinates gridless
                              //    +4 = set mouse pointer to specified position
                              //    +8 = activate BAE window
      int [0,3];              // Mouse key code (STD17)
      [];                     // Keyboard input
      );
```

Description

The **bae_storemouseiact** function stores a mouse interaction to the interaction queue to be passed to a **bae_callmenu**-activated BAE menu function. The given positioning with mouse-click is automatically passed if the interaction flag is set, otherwise an interactive user input is required. A character must be passed to the keyboard input parameter if the mouse key code is set to zero.

See also

Functions **bae_callmenu**, **bae_peekiact**, **bae_storekeyiact**, **bae_storemenuiact**, **bae_storetextiact**.

bae_storepoint - Store point to internal BAE polygon (STD)

Synopsis

```
int bae_storepoint(           // Returns status
      double;                 // Point X coordinate (STD2)
      double;                 // Point Y coordinate (STD2)
      int [0,2];              // Point type (STD15)
      );
```

Description

The **bae_storepoint** function stores the specified point to the internal polygon point list. The function returns nonzero if invalid point parameters have been specified. The internal polygon point list is required for generating polygons or traces using the module-specific ***_storepoly** and/or ***_storepath** functions or the **bae_storedistpoly** function. The internal BAE polygon can be deleted/cleared with the **bae_clearpoints** function.

See also

Functions **bae_clearpoints**, **bae_getpolyrange**, **bae_storedistpoly**.

bae_storetextiact - Store BAE text input interaction to queue (STD)

Synopsis

```
void bae_storetextiact(
      int [0,3];              // Automatic interaction mode (STD21)
      string;                 // Input text string
      );
```

Description

The **bae_storetextiact** function stores a text input interaction to the interaction queue to be passed to a **bae_callmenu**-activated BAE menu function. The given text string is automatically passed if the interaction flag is set, otherwise an interactive user input is required.

See also

Functions **bae_peekiact**, **bae_storekeyiact**, **bae_storemenuiact**, **bae_storemouseiact**.

bae_swconfig - Get BAE software configuration (STD)

Synopsis

```
int bae_swconfig(              // Returns software configuration code:
                               //    (-1) = invalid configuration class,
                               // for configuration class 0:
                               //    0 = unknown system
                               //    1 = Bartels ACAD-PCB
                               //    2 = BAE Professional
                               //    3 = BAE HighEnd
                               // for configuration class 1:
                               //    0 = not BAE Demo software
                               //    1 = BAE Demo software
                               //    2 = BAE FabView software
                               // for configuration class 2:
                               //    nonzero = BAE Economy software
                               // for configuration class 3:
                               //    0 = BAE standard menu screen
                               //    1 = BAE Windows standard menu
                               //    2 = BAE Windows pull-down menu
                               //    3 = BAE Motif standard menu
                               //    4 = BAE Motif pull-down menu
                               // for configuration class 4:
                               //    nonzero = BAE Light software
                               // for configuration class 5:
                               //    0 = no BAE Schematics software
                               //    1 = BAE Schematics
                               //    2 = BAE HighEnd Schematics
                               // for configuration class 6:
                               //    BAE version build number
     int;                      // Configuration class:
                               //    0 = BAE software system query
                               //    1 = BAE Demo software check
                               //    2 = BAE Economy software check
                               //    3 = BAE user interface type query
                               //    4 = BAE Light software check
                               //    5 = BAE Schematics software check
                               //    6 = BAE version build number query
     );
```

Description

The **bae_swconfig** function returns the currently active BAE software configuration. This information is required for performing correct menu selections using **bae_defmenuprog**, **bae_callmenu** and **bae_store*iact** calls in different software configurations such as **BAE Professional**, **BAE HighEnd**, **BAE Economy** and/or **BAE Light**, or in different user interfaces such as BAE standard user interface or BAE pull-down user interface under DOS/X11, Windows or Motif.

See also

Function **bae_swversion**.

bae_swversion - Get BAE software version (STD)

Synopsis

```
string bae_swversion(          // Returns software version string
                               //    or operating system setting
      int;                     // Software version query mode:
                               //    0 = BAE version number
                               //    1 = BAE release year (format YY)
                               //    2 = BAE release year (format YYYY)
                               //    3 = Operating system specific
                               //         pattern for arbitrary string match
                               //         (e.g., * under Linux, .* under MS-DOS)
                               //    4 = Operating system specific
                               //         directory name delimiter
                               //         (e.g., / under Linux, \ under MS-DOS)
      );
```

Description

The **bae_swversion** function returns BAE software version information or operating system specific BAE settings according to the specified software version query mode.

See also

Function **bae_swconfig**.

bae_tbsize - Get BAE toolbar dimensions (STD)

Synopsis

```
void bae_tbsize(
      & double;                // Returns toolbar text columns
      & double;                // Returns toolbar text rows
      );
```

Description

The **bae_tbsize** function retrieves the size of the toolbar area currently defined with the **bae_settbsize** function. The toolbar width is returned with the toolbar text columns parameter, whilst the toolbar height is returned with the toolbar text rows parameter. The **bae_charsize** function can be utilized to convert these values to standard length units.

See also

Functions **bae_charsize**, **bae_mtpsize**, **bae_settbsize**.

bae_twsize - Get BAE text screen workspace size (STD)

Synopsis

```
void bae_twsize(
      & int;                   // Returns text column count
      & int;                   // Returns text row count
      );
```

Description

The **bae_twsize** function determines the current size of the BAE text screen workarea. The text screen size is returned through the parameters, with the first parameter returning the text column count, and the second parameter returning the text row count. These functions are useful for dynamically adapting output routines to the BAE graphic environment.

See also

Function **bae_mtpsize**.

bae_wswinlx - Get BAE workspace window left boundary (STD)

Synopsis

```
double bae_wswinlx(           // Left workspace window boundary (STD2)
      );
```

Description

The **bae_wswinlx** function returns the left boundary coordinate of the currently visible workspace window.

bae_wswinly - Get BAE workspace window lower boundary (STD)

Synopsis

```
double bae_wswinly(           // Lower workspace window boundary (STD2)
      );
```

Description

The **bae_wswinly** function returns the lower boundary coordinate of the currently visible workspace window.

bae_wswinux - Get BAE workspace window right boundary (STD)

Synopsis

```
double bae_wswinux(           // Right workspace window boundary (STD2)
      );
```

Description

The **bae_wswinux** function returns the right boundary coordinate of the currently visible workspace window.

bae_wswinuy - Get BAE workspace window upper boundary (STD)

Synopsis

```
double bae_wswinuy(           // Upper workspace window boundary (STD2)
      );
```

Description

The **bae_wswinuy** function returns the upper boundary coordinate of the currently visible workspace window.

bae_wsmouse - Get BAE workspace mouse position (STD)

Synopsis

```
void bae_wsmouse(
      & double;               // Returns mouse X coordinate (STD2)
      & double;               // Returns mouse Y coordinate (STD2)
      & int;                  // Returns mouse state:
                              //    Bit 0 : Beyond left workspace boundary
                              //    Bit 1 : Beyond right workspace boundary
                              //    Bit 2 : Beyond lower workspace boundary
                              //    Bit 3 : Beyond upper workspace boundary
                              //    Bit 4 : Left mouse button pressed
                              //    Bit 5 : Right mouse button pressed
                              //    Bit 6 : Middle mouse button pressed
      );
```

Description

The **bae_wsmouse** function retrieves the current mouse coordinates within the graphic workarea. The mouse state return parameter can be used to designate which mouse buttons are currently pressed or whether the mouse coordinates exceed any of the workspace boundaries.

See also

Function **bae_popmouse**.

catext - Concatenate file name extension (STD)

Synopsis

```
void catext(
      & string;               // File name
      string;                 // File name extension
      );
```

Description

The **catext** function appends the specified file name extension to the file name provided with the first function parameter if that file name doesn't have this extension yet.

See also

Function **catextadv**.

catextadv - Optionally concatenate file name extension (STD)

Synopsis

```
void catextadv(
      & string;               // File name
      string;                 // File name extension
      int;                    // Extension mode:
                              //   0 = Apply extension only if
                              //         no extension yet
                              //   1 = Assume file name contains
                              //         none or specified extension
                              //   2 = Force specified extension
      );
```

Description

The **catextadv** function optionally appends the specified file name extension to the specified file name. The extension mode parameter control whether and/or under which circumstances the extension is appended.

See also

Function **catext**.

ceil - Ceiling function (STD)

Synopsis

```
double ceil(                  // Returns result value
      double;                 // Input value
      );
```

Description

The **ceil** function calculates and returns the smallest "integer" greater than or equal to the specified input double value.

clock - Get elapsed processor time (STD)

Synopsis

```
double clock(                 // Returns elapsed CPU time (seconds)
      );
```

Description

The **clock** function returns (in seconds) the CPU time elapsed since the start of the currently active BAE program module. This value should be used only for comparison purposes.

con_clear - Delete internal logical net list (STD)

Synopsis

```
void con_clear(
      );
```

Description

The **con_clear** function deletes the net list data previously stored with **con_storepart** and/or **con_storepin** from main memory. **con_clear** should always be applied before the first call to **con_storepart** to avoid net list mix with previously stored net data. The **con_clear** function should also be called as soon as the net list data is not needed any more, e.g., after finishing a net list transformation since otherwise you might run short of main memory.

See also

Functions **con_storepart**, **con_storepin**, **con_write**.

con_compileloglib - Compile logical library definition (STD)

Synopsis

```
int con_compileloglib(        // Returns status
      string;                 // DDB destination file name
      string;                 // Logical library definition file name
      string;                 // Logical library definition
      );
```

Description

The **con_compileloglib** function compiles the given logical library definition(s) and stores the compiled loglib definition(s) to the DDB destination file specified with the first function parameter. The logical library definition(s) must be provided according to the **LOGLIB** utility program input format specification. **con_compileloglib** compiles either the logical library definition file specified with the second function parameter or the logical library definition passed through the third function parameter. The **NULL** keyword can be specified to ignore either of these parameters. The function returns zero if the compilation was successful or nonzero otherwise.

See also

Functions **con_compileloglib**, **con_getlogpart**, **lay_deflibname**, **scm_defloglname**; BAE utility program **LOGLIB**.

con_deflogpart - Define a logical library part entry (STD)

Synopsis

```
int con_deflogpart(           // Returns status
      string;                 // DDB destination file name
      string;                 // Logical library part name
      string[];               // Physical library part name
      int [0,1];              // Default assignment flag
      );
```

Description

The **con_deflogpart** function stores the given logical library part definition to the name-specified DDB destination file. The defined assignment of a physical library part to a logical part is requested by the BAE **Packager** for transferring logical to physical net lists. The library entry defined through **con_deflogpart** is an 1-to-1 assignment, i.e., the **Packager** does not perform any pin name transformations. Note that gates, pin/gate swaps, power supply pins or attribute assignments cannot be defined with **con_deflogpart**. A virtual part assignment is defined if an empty string is specified for the physical library part name. Alternate package types are defined if a list of multiple strings is given for the physical library part name. The **$plname** attribute can be utilized for physical part name assignments if the default assignment flag is set. The function returns zero on successful library part definition, (-1) on missing and/or invalid parameters or (-2) on library file access errors.

See also

Functions **con_deflogpart**, **con_getlogpart**; BAE utility program **LOGLIB**.

con_getddbpattrib - Get part/pin attribute from DDB file (STD)

Synopsis

```
int con_getddbpattrib(        // Returns status
      string;                 // DDB file name
      string;                 // Part name
      string;                 // Pin name
      & string;               // Attribut name
                              //   or empty string for plan name
                              //   or ! for first attribute
                              //   or !$attrname for next attribute
      & string;               // Attribute value
      );
```

Description

The **con_getddbpattrib** function gets a part or pin attribute value from a DDB file. The function returns 1 if the attribute value has been found, zero if the attribute does not exist or (-1) on file access errors. Part attributes are searched if an empty pin name is specified; otherwise pin attributes are searched. The selected attribute name is returned with the attribute name function parameter if a placeholder (`!` or `!$attrname`) is specified for the attribute name.

See also

Function **con_setddbpattrib**.

con_getlogpart - Get a logical library part definition (STD)

Synopsis

```
int con_getlogpart(           // Returns status
      string;                 // LOGLIB DDB file name
      string;                 // Logical library part name
      int;                    // Output columns
      & string;               // Returns logical library definition
      );
```

Description

The **con_getlogpart** function extracts ASCII-formatted logical library part definition of the name-specified logical part. The function returns zero on successful part definition extraction, (-1) on missing and/or invalid parameters, (-2) if the DDB file is not available, (-3) if the logical library part is not available or (-4) on logical library part load errors. The LOGLIB DDB file name is supposed to be a job file name or the default **Packager** layout library name which can be determined using either the **scm_defloglname** or the **lay_deflibname** function.

See also

Functions **con_compileloglib**, **con_deflogpart**, **lay_deflibname**, **scm_defloglname**; BAE utility program **LOGLIB**.

con_setddbpattrib - Store part/pin attribute to DDB file (STD)

Synopsis

```
int con_setddbpattrib(        // Returns status
      string;                 // DDB file name
      string;                 // Part name
      string;                 // Pin name
      string;                 // Attribute name
      string;                 // Attribute value
      );
```

Description

The **con_setddbpattrib** function sets a part or pin attribute value in the given DDB file. The function returns nonzero on error. The part name is required; the specified part must be defined in the net list of the given DDB file. The pin name is optional. The attribute value is considered a part attribute value if an empty string is specified for the pin name; otherwise, a pin attribute assignment is assumed. Neither part nor pin names can contain upper case letters. The attribute name must start with the percent sign (**$**) and must not contain upper case letters. The attribute value can be an empty string for resetting previously assigned attribute values. Attribute value strings can be stored with a maximum length of up to 40 characters. For **PA_NILVAL** attribute value specifications, the attribute value assignment for the specified attribute is removed to mimic the behaviour of No Value button in the **Schematic Editor**

See also

Function **con_getddbpattrib**.

con_storepart - Store part to internal logical net list (STD)

Synopsis

```
int con_storepart(            // Returns status
      string;                 // Logical part name
      string;                 // Logical part library name
      );
```

Description

The **con_storepart** function stores the given logical part to the internal logical net list kept in main memory. The function returns zero if the part has been successfully stored, (-1) on missing and/or invalid parameters or (-2) if the part is already defined. The internal logical net list can be written to a DDB file using the **con_write** function.

Warning

Main memory is allocated when using the **con_store*** functions. The **con_clear** function should be called as soon as the net data is not needed any more, or otherwise you might run short of main memory.

See also

Functions **con_clear**, **con_storepin**, **con_write**.

con_storepin - Store pin to internal logical net list (STD)

Synopsis

```
int con_storepin(             // Returns status
      string;                 // Logical part name
      string;                 // Logical part pin name
      string;                 // Net name
      );
```

Description

The **con_storepin** function stores the given logical part pin to the internal logical net list kept in main memory. The corresponding part must be stored already with the **con_storepart** function. The pin is not connected if an empty string is specified for the net name; otherwise the pin is connected to the specified net. The net entry is automatically generated if no net with the given name exists. The function returns zero if the part pin has been successfully stored, (-1) on missing and/or invalid parameters, (-2) if the part is not yet defined, (-3) if the pin is already connected to the net list or (-4) on net list overflow. The internal logical net list can be written to a DDB file using the **con_write** function.

Warning

Main memory is allocated when using the **con_store*** functions. The **con_clear** function should be called as soon as the net data is not needed any more, or otherwise you might run short of main memory.

See also

Functions **con_clear**, **con_storepart**, **con_write**.

con_write - Write internal logical net list to file (STD)

Synopsis

```
int con_write(                // Returns status
      string;                 // DDB file name
      string;                 // Net list element name
      );
```

Description

The **con_write** function stores internal logical net list generated with the **con_store*** function to the specified net list element name in the given DDB file. **con_write** also deletes the internal logical net list from memory. The function returns zero if the net list has been successfully written, (-1) on missing and/or invalid parameters, (-2) if no internal net list data is available or (-3) on net list write errors. The net list format produced by **con_write** is identical to the format generated by the **Schematic Editor** of the **Bartels AutoEngineer**, i.e., the net lists generated with **con_write** can be transformed to the **Bartels AutoEngineer** layout system using the Packager.

Warning

The **con_*** functions provide powerful tools for transforming third party net lists to the **Bartels AutoEngineer**. It is strongly recommended to check destination files before storing net lists with the **con_write** function to avoid conflicts with other net list data already defined in the destination file.

See also

Functions **con_clear**, **con_storepart**, **con_storepin**.

convstring - Convert string (STD)

Synopsis

```
string convstring(            // Returns converted string
      string;                 // Input string
      int;                    // Conversion mode:
                              //    0 = Get file name without extension
                              //    1 = Get file name without directory path
                              //    2 = Get file name without extension
                              //         and directory path
      );
```

Description

The **convstring** function converts the input string according to the specified conversion mode and returns the converted string.

cos - Cosine (STD)

Synopsis

```
double cos(                   // Returns result value
      double;                 // Input angle value (STD3)
      );
```

Description

The **cos** function calculates and returns the cosine value of the given angle value. The input angle value must be in radians.

cosh - Hyperbolic cosine (STD)

Synopsis

```
double cosh(                  // Returns result value
      double;                 // Input angle value (STD3)
      );
```

Description

The **cosh** function calculates and returns the hyperbolic cosine value of the given angle value. The input angle value must be in radians.

cvtangle - Convert an angle value (STD)

Synopsis

```
double cvtangle(              // Returns angle value
      double;                 // Input angle value
      int [0,3];              // Input angle units
      int [0,3];              // Output angle units
      );
```

Description

The **cvtangle** function converts the specified angle value from input units to output units and returns the resulting angle value. Valid input/output angle unit codes are 0 for internal units (radians, STD3), 1 for degree units (180/pi radians), 2 for radians (STD3) or 3 for grad units (200/pi radians).

cvtlength - Convert a length value (STD)

Synopsis

```
double cvtlength(             // Returns length value
      double;                 // Input length value
      int [0,4];              // Input length units
      int [0,4];              // Output length units
      );
```

Description

The **cvtlength** function converts the specified length value from input units to output units and returns the resulting length value. Valid input/output length unit codes are 0 for internal units (meter, STD2), 1 for inch units, 2 for millimeter units (mm), 3 for mil units (1/1000 inch) or 4 for micrometer units (um).

ddbcheck - Check DDB file/element availability (STD)

Synopsis

```
int ddbcheck(                 // Returns query status
      string;                 // DDB file name
      int;                    // DDB element class (STD1)
                              //    or (-1) for valid DDB file check
      string;                 // Element name or empty string for any class
element check
      );
```

Description

The **ddbcheck** function checks whether an element with the specified DDB class and element name exists in the DDB file specified with the DDB file name parameter. If (-1) is specified instead of a valid DDB class code, then **ddbcheck** only checks whether the specified file is a DDB file. If an empty element name string is specified, then **ddbcheck** only checks whether the DDB file contains any element of the specified DDB class. The function returns zero if the DDB object was found and/or accessible or (-1) otherwise.

See also

Functions **ddbclassscan**, **ddbelemrefcount**, **ddbelemrefentry**,

ddbclassid - Get DDB class identifier (STD)

Synopsis

```
string ddbclassid(            // Returns DDB class identifier or empty string
      int ]0,[;               // DDB element class (STD1)
      );
```

Description

The **ddbclassid** function returns the DDB class identifier for the specified DDB class code. An empty string is returned on invalid/unknown DDB class specifications.

ddbclassscan - Scan DDB class elements (STD)

Synopsis

```
int ddbclassscan(             // Returns scan status
      string;                 // DDB file name
      int ]0,[;               // Element class code (STD1)
      * int;                  // Element name callback function
      );
```

Description

The **ddbclassscan** function scans all elements matching the specified DDB class in the given DDB file. A user-defined callback function (see below) is automatically activated for each element scanned, unless the keyword **NULL** is specified for the corresponding parameter. The function returns either the number of scanned elements or (-1) on invalid parameter specifications and/or errors from the callback function.

Element name callback function definition

```
int callbackfunction(         // Continue scan request flag
      string ename            // Element name
      )
{
      // Element name callback function statements
      :
      return(scanstatus);
}
```

The element name callback function return value should be 1 for scan continue request, 0 for scan stop request or (-1) on error.

See also

Functions **ddbcheck**, **ddbelemrefcount**, **ddbelemrefentry**.

ddbcopyelem - Copy DDB file element (STD)

Synopsis

```
int ddbcopyelem(              // Returns status
      string;                 // DDB source file name
      string;                 // DDB destination file name
      int ]0,[;               // DDB database class code (STD1)
      string;                 // DDB element name
      int [0,1];              // Delete allowed (merge source) flag:
                              //    0 = do not overwrite existing
                              //        elements in destination file
                              //    1 = allow overwriting of existing
                              //        elements in destination file
      );
```

Description

The **ddbcopyelem** function copies the named DDB element of the specified DDB database class from the DDB source file to the DDB destination file. The DDB element name is preserved during the copy, i.e., the source and destination files must be different. The copy includes all dependent and/or referenced elements of the selected DDB element. The merge source switch parameter designates whether existing elements in the destination file can be overwritten (merge source; source file is master) or not (merge destination; destination file is master). The function returns zero if the copy operation was successful or (-1) otherwise (invalid/missing parameters or DDB file access failure).

Warnings

The **ddbcopyelem** function is *not* subject to the Undo/Redo facility since it works on DDB file level. It is strongly recommended to use this function with care to prevent from unintentionally overwriting BAE design and/or system data.

See also

Functions **ddbdelelem**, **ddbrenameelem**.

ddbdelelem - Delete DDB file element (STD)

Synopsis

```
int ddbdelelem(               // Returns status
      string;                 // DDB file name
      int ]0,[;               // DDB database class code (STD1)
      string;                 // DDB element name
      );
```

Description

The **ddbdelelem** function deletes the named DDB file element of the specified DDB database class code from the given DDB file. The function returns zero if the delete operation was successful or (-1) otherwise (invalid/missing parameters or DDB file access failure).

Warnings

The **ddbdelelem** function is *not* subject to the Undo/Redo facility since it works on DDB file level. It is strongly recommended to use this function with care to prevent from unintentionally deleting BAE design and/or system data.

See also

Functions **ddbcopyelem**, **ddbrenameelem**.

ddbelemrefcount - Get DDB file element reference count (STD)

Synopsis

```
int ddbelemrefcount(          // Returns reference count or (-1)
      string;                 // DDB file name
      int ]0,[;               // DDB database class code (STD1)
      string;                 // DDB element name
      );
```

Description

The **ddbelemrefcount** function retrieves the number of library elements referenced by the specified DDB element. The DDB element is specified with the corresponding DDB file name, DDB class code and element name function parameters. The function returns (-1) if the specified DDB element does not exist or cannot be accessed. The **ddbelemrefentry** function can be used to retrieve the DDB database class codes and the element names of selected library references of a DDB element.

See also

Functions **ddbelemcheck**, **ddbclassscan**, **ddbelemrefentry**.

ddbelemrefentry - Get DDB file element reference entry (STD)

Synopsis

```
int ddbelemrefentry(          // Returns reference count or (-1)
      string;                 // DDB file name
      int ]0,[;               // DDB database class code (STD1)
      string;                 // DDB element name
      int ]0,[;               // Reference entry list index
      & int ]0,[;             // Reference entry DDB database class code (STD1)
      & string;               // Reference entry name
      );
```

Description

The **ddbelemrefentry** function retrieves the DDB database class code and the element name of the library element referenced by the DDB element specified with the DDB file name, DDB database class code and element name function parameters. The reference entry to be queried is specified with the reference list index parameter. The **ddbelemrefcount** function can be used to retrieve the number of DDB element library reference entries. This value also designates the valid reference list index range to be specified for selecting a specific DDB element reference entry. The function returns zero if the query was successful or (-1) otherwise.

See also

Functions **ddbcheck**, **ddbclassscan**, **ddbelemrefcount**.

ddbgetelemcomment - Get DDB file element comment (STD)

Synopsis

```
int ddbgetelemcomment(        // Returns status
      string;                 // DDB file name
      int ]0,[,               // DDB database class (STD1)
      string;                 // DDB element name
      & string;               // Returns comment string
      );
```

Description

The **ddbgetelemcomment** function is used to query DDB file element comments which have been assigned with the **ddbsetelemcomment** function. The functions returns zero if the query was successful or (-1) otherwise (missing/invalid parameters or DDB access failure).

See also

Function **ddbsetelemcomment**.

ddbgetlaypartpin - Get DDB file layout part pin data (STD)

Synopsis

```
int ddbgetlaypartpin(         // Returns status:
                              //    ( 0) = no error, pin query successful
                              //    (-1) = DDB file access error
                              //    (-2) = Invalid parameters
                              //    (-3) = Layout part symbol not found
                              //    (-4) = Pin not defined
      string;                 // DDB file name
      string;                 // Layout part symbol name
      int [0,[;               // Pin index
      & string;               // Pin name
      & string;               // Pin symbol name
      & double;               // Pin X position (STD2)
      & double;               // Pin Y position (STD2)
      & double;               // Pin rotation angle (STD3)
      & int;                  // Pin mirror mode (STD14)
      );
```

Description

The **ddbgetlaypartpin** function can be used to query layout part symbol pin information on DDB file level. Repetitive function calls with incremented pin index and the pin not defined return value as abort criteria can be applied for iterating all pins of an existing layout part symbol.

See also

Functions **ddbcheck**, **ddbclassscan**, **ddbelemrefcount**, **ddbelemrefentry**.

ddbrenameelem - Rename DDB file element (STD)

Synopsis

```
int ddbrenameelem(            // Returns status
      string;                 // DDB file name
      int ]0,[,               // DDB database class (STD1)
      string;                 // Old DDB element name
      string;                 // New DDB element name
      );
```

Description

The **ddbrenameelem** changes the element name of the specified DDB file element. The function returns zero if the rename operation was successful or (-1) otherwise (invalid/missing parameters or DDB file access failure).

Warnings

The **ddbrenameelem** function is *not* subject to the Undo/Redo facility since it works on DDB file level. It is strongly recommended to use this function with care to prevent from unintentionally renaming BAE design and/or system data.

See also

Functions **ddbcopyelem**, **ddbdelelem**.

ddbsetelemcomment - Set DDB file element comment (STD)

Synopsis

```
int ddbsetelemcomment(        // Returns status
      string;                 // DDB file name
      int ]0,[,               // DDB database class (STD1)
      string;                 // DDB element name
      string;                 // Comment string
      );
```

Description

The **ddbsetelemcomment** function can be used to assign comment texts to DDB file elements for subsequent queries with the **ddbgetelemcomment** function. The functions returns zero if the assignement was successful or (-1) otherwise (missing/invalid parameters or DDB access failure). **ddbsetelemcomment** sets DDB element comments not only in DDB files but also in main memory if the processed DDB element is currently loaded in the **Bartels AutoEngineer**.

See also

Function **ddbgetelemcomment**.

ddbupdtime - Get DDB file element update time (STD)

Synopsis

```
int ddbupdtime(               // Returns status
      string;                 // Input DDB file name
      int [100,[;             // Input DDB element class code (STD1)
      string;                 // Input DDB element name
      & int;                  // Returns update time seconds
      & int;                  // Returns update time minutes
      & int;                  // Returns update time hours
      & int;                  // Returns update time days
      & int;                  // Returns update time months
      & int;                  // Returns update time year
      );
```

Description

The **ddbupdtime** function gets the date and time of the last change performed on the specified DDB file element. The function returns 1 on success, 0 if the DDB file element has not been found or (-1) on file access error or invalid parameters.

dirscan - Scan directory (STD)

Synopsis

```
int dirscan(                    // Returns scan status
      string;                   // Directory path name
      string;                   // File name extension:
                                //    .EXT = denotes extension
                                //    .*   = all files/subdirectories
      * int;                    // File name callback function
      );
```

Description

The **dirscan** function scans the specified directory for all file and/or subdirectory names matching the file name extension. A user-defined callback function (see below) is activated automatically for each file name scanned, unless the keyword **NULL** is specified for the corresponding parameter. The function returns either the number of scanned file names or (-1) on invalid parameter specifications and/or errors from the callback function.

File name callback function

```
int callbackfunction(           // Continue scan request flag
      string fname              // File name
      )
{
      // File name callback function statements
      :
      return(scanstatus);
}
```

The file name callback function return value should be 1 for scan continue request, 0 for scan stop request, or (-1) on error.

existddbelem - Check DDB file element (STD)

Synopsis

```
int existddbelem(               // Returns status
      string;                   // Input DDB file name
      int [100,[;               // Input DDB element class code (STD1)
      string;                   // Input DDB element name
      );
```

Description

The **existddbelem** function checks if the specified DDB file element exists. The function returns 1 if the DDB file element has been found, 0 if the DDB file element has not been found or (-1) on file access error or invalid parameters.

exit - Terminate a program immediately (STD)

Synopsis

```
void exit(
      int;                      // Return status
      );
```

Description

The **exit** function terminates the currently active **User Language** program immediately and passes the specified status code to the program caller.

Warning

The return status is not evaluated by the **Bartels User Language Interpreter**.

See also

Function **ulsystem_exit**.

exp - Exponential function (STD)

Synopsis

```
double exp(                    // Returns result value
      double;                  // Input value
      );
```

Description

The **exp** function calculates and returns the exponential value of the given input value.

fabs - Absolute value of a double (STD)

Synopsis

```
double fabs(                   // Returns result value
      double;                  // Input value
      );
```

Description

The **fabs** function calculates and returns the absolute value of the given double value.

fclose - Close a file (STD)

Synopsis

```
int fclose(                    // Returns status
      int;                     // File handle
      );
```

Description

The **fclose** function closes the file specified with the given file handle. The function returns nonzero on file close errors.

fcloseall - Close all files opened by the program (STD)

Synopsis

```
int fcloseall(                 // Returns status
      );
```

Description

The **fcloseall** function closes all files opened by the currently active **User Language** program. The function returns nonzero on file close errors.

feof - Test for end-of-file (STD)

Synopsis

```
int feof(                      // Returns status
      int;                     // File handle
      );
```

Description

The **feof** function checks if the end of the file specified with the given file handle is reached. The function returns nonzero on end-of-file or zero otherwise.

fgetc - Read next character from file (STD)

Synopsis

```
int fgetc(                    // Returns character code (or -1 if EOF)
      int;                    // File handle
      );
```

Description

The **fgetc** function reads the next character from the file specified with the given file handle. The fucntion returns the read character code on success or (-1) on file read errors or end-of-file (these states should be distinguished with the **feof** function; the file function error handling mode must be set accordingly with the **fseterrmode** function).

fgets - Read next line of text from file (STD)

Synopsis

```
int fgets(                    // Returns status
      & string;               // Returns text string
      int;                    // Maximum text string length
      int;                    // File handle
      );
```

Description

The **fgetc** function reads the next line of text from the file specified with the given file handle. Reading stops after a newline character is encountered or if the number of characters stored to the result string exceeds the maximum text string length. The return value is nonzero on file read errors or end-of-file (these states should be distinguished with the **feof** function; the file functions error handling mode must be set accordingly with the **fseterrmode** function).

filemode - Get file mode (STD)

Synopsis

```
int filemode(                 // Return file name
      string;                 // File name
      );
```

Description

The **filemode** return the access mode of the specified file. The return value is 0 for write access, 1 for read-only access, or (-1) on file access failure.

See also

Functions **filesize**, **filetype**.

filesize - Get file size (STD)

Synopsis

```
int filesize(                 // File size in bytes or (-1) on error
      string;                 // File name
      );
```

Description

The **filesize** function returns the size of the specified file in bytes or (-1) on file access failure.

See also

Functions **filemode**, **filetype**.

filetype - Get file type (STD)

Synopsis

```
int filetype(                 // Returns file type:
                              //    (-1) = file access failure
                              //    ( 0) = directory
                              //    ( 1) = regular file
                              //    ( 2) = character mode file
      string;                 // File name
      );
```

Description

The **filetype** function returns the type of the specified file or (-1) if file access failure.

See also

Functions **filemode**, **filesize**.

floor - Floor function (STD)

Synopsis

```
double floor(                 // Returns result value
      double;                 // Input value
      );
```

Description

The **floor** function calculates and returns the largest "integer" not greater than the specified input double value.

fmod - Floating point remainder (STD)

Synopsis

```
double fmod(                  // Returns result value
      double;                 // Dividend
      double;                 // Divisor
      );
```

Description

The **fmod** function calculates and returns the floating point remainder of the division of the two input values. The result value has the same sign like the dividend. If the division result cannot be represented, the result value is undefined.

fopen - Open a file (STD)

Synopsis

```
int fopen(                    // Returns status
      string;                 // File name
      int [0,14];             // File access mode:
                              //    0 = r read
                              //         (the only valid BAE Demo mode without |8)
                              //    1 = w write
                              //    2 = a append
                              //    3 = rb read binary
                              //    4 = wb write binary
                              //    5 = ab append binary
                              //    |8 = autoclose (for BAE Demo write tests)
      );
```

Description

The **fopen** function opens the file with the specified file name and provides the required file access. The file access mode is 0 for file read access, 1 for file write access or 2 for file append access. On write access, the file is generated if it does not yet exist. The function returns (-1) on file open errors; on success, the return value is the file handle to be used for subsequent file access functions.

Limitation

When used in **BAE Demo** software configurations, **fopen** allows for file read access only.

fprintf - Print to a file using format (STD)

Synopsis

```
int fprintf(                    // Returns status
      int;                      // File handle
      string;                   // Format string
      []                        // Output parameter list
      );
```

Description

The **fprintf** function writes the data contained in the output parameter list to the file specified with the file handle. The format string contains information on how to format the output. The function returns nonzero on file write errors or zero otherwise.

Format string

The format string transforms the subsequent parameter values and performs a formatted output according to the format control information specified with the format string. The format string can contain normal characters and format elements. Normal character sequences are written unchanged whilst a format element causes transformation and formatted output of the next unprocessed output parameter value. Each format element starts with the percent sign **%** and is delimited by a format control character. The following table lists the valid format control characters for output data type specifications:

Character	Output Data Type
d	Decimal integer format
o	Octal integer format
x	Hexadecimal integer format (lowercase)
X	Hexadecimal integer format (uppercase)
u	Unsigned decimal integer format
c	Character format
s	String format
e	Scientific floating point format (lowercase)
E	Scientific floating point format (uppercase)
f	Fixed floating point format
g	**e** or **f**, with shorter form to be used
G	**E** or **f**, with shorter form to be used
%	Print percent sign **%**

The following output format specifications can be used between the percent sign and the format control character (n = numeric value):

Character	Output Format
`-`	left justify (default: right justify)
`+`	sign output with positive numeric values
`SPACE`	blank output with positive numeric values
`#`	octal output with leading `0` or hexadecimal output with leading `0x` or `0X` or output with decimal point for `e`, `E`, `f`, `g`, `G` or zeros after decimal point for `g`, `G`
`n`	fieldwidth; i.e., minimum output length
`.n`	precision; i.e., length of output string (with `s`) or number of digits after decimal point (with `f`, `g`, `G`)
`l`	long int decimal format (with `d`, `o`, `x`, `X`)

With `e`, `E`, `f`, `g` and `G`, six digits after the decimal point are printed on default. If the `*` character appears instead of the fieldwidth or precision specification then the corresponding value is taken from the next unprocessed function parameter. A percent sign can be printed by specifying `%%`. Control characters delimited by a backslash (`\`) are treated like normal characters.

Warning

The number and types of the parameter values passed to **fprintf** must match the number of percent sign / format control character pairs specified with the format stringl; otherwise garbage might be printed.

See also

Functions **printf**, **sprintf**.

fputc - Write a character to a file (STD)

Synopsis

```
int fputc(                    // Returns status
      char;                   // Character
      int;                    // File handle
      );
```

Description

The **fputc** function writes the given character to the file specified with the file handle. The function returns nonzero on file write error or zero otherwise.

fputs - Write a string to a file (STD)

Synopsis

```
int fputs(                    // Returns status
      string;                 // String
      int;                    // File handle
      );
```

Description

The **fputs** function writes the given string to the file specified with the file handle. The function returns nonzero on file write error or zero otherwise.

frexp - Break double into fraction and exponent (STD)

Synopsis

```
double frexp(                  // Returns fraction value
      double;                  // Input value
      & int;                   // Returns exponent value
      );
```

Description

The **frexp** function breaks the input double value fraction and exponent and returns the calculated fraction value; the exponent is returned with the second parameter.

fseterrmode - Set the file functions error handling mode (STD)

Synopsis

```
int fseterrmode(               // Returns status
      int [0,1];               // Error handling mode
      );
```

Description

The **fseterrmode** function sets the file functions error handling mode. The file error mode 1 causes the interpreter environment to handle file errors. The file error mode 0 leaves the **User Language** program with the file error handle task. The initial file error handling mode on **User Language** program calls is 1.

Warning

The file error handling mode should be set to 0, when using the functions **fgetc** and/or **fgets**. Otherwise a **User Language** program might fault with file read errors when reaching end-of-file (see also functions **fgetc**, **fgets**).

get_date - Get the current system date (STD)

Synopsis

```
void get_date(
      & int;                   // Returns day of the month (1..31)
      & int;                   // Returns month of the year (0..11)
      & int;                   // Returns year since 1900
      );
```

Description

The **get_date** function returns with its parameters the current system date.

get_time - Get the current system time (STD)

Synopsis

```
void get_time(
      & int;                   // Returns hours since midnight (0..23)
      & int;                   // Returns minutes after hour (0..59)
      & int;                   // Returns seconds after minute (0..59)
      );
```

Description

The **get_time** function returns with its parameters the current system time.

getchr - Get a character from standard input (STD)

Synopsis

```
char getchr(                   // Returns character
      );
```

Description

The **getchr** function activates a character input request (from keyboard) and returns the input character value.

getcwd - Get current working directory path name (STD)

Synopsis

```
string getcwd(                  // Returns path name
      );
```

Description

The **getcwd** function returns the path name of the current working directory.

getenv - Get environment variable (STD)

Synopsis

```
int getenv(                     // Returns status
      string;                   // Variable name
      & string;                 // Variable value
      );
```

Description

The **getenv** function searches the environment variable list for an entry matching the specified variable name and returns the variable value with the corresponding parameter. The function returns zero if the variable is defined or nonzero otherwise (in which case the variable value parameter is left unchanged).

See also

Function **putenv**.

getextprog - Get file type specific application (STD)

Synopsis

```
int getextprog(                 // Returns status
      string;                   // File name extension
      & string;                 // Command string
      );
```

Description

The **getextprog** gets the application command string for opening file types with the specified file name extension. The function returns zero if the query was successful or (-1) if no application/command was found for the specified file name extension.

getstr - Get a line of text from standard input (STD)

Synopsis

```
int getstr(                     // Returns status
      & string;                 // Returns text string
      int;                      // Maximum text string length
      );
```

Description

The **getstr** function activates a string input request (from keyboard). Input characters are stored to the return text string parameter until the return/enter key ↵ key is pressed or the maximum text string length is reached. The function returns nonzero if invalid parameters have been specified.

isalnum - Test for alphanumeric character (STD)

Synopsis

```
int isalnum(                    // Returns boolean test result
      char;                     // Input character
      );
```

Description

The **isalnum** function returns nonzero if the input character is alphanumeric or zero otherwise.

isalpha - Test for alphabetic character (STD)

Synopsis

```
int isalpha(                  // Returns boolean test result
      char;                   // Input character
      );
```

Description

The **isalpha** function returns nonzero if the input character is alphabetic or zero otherwise.

iscntrl - Test for control character (STD)

Synopsis

```
int iscntrl(                  // Returns boolean test result
      char;                   // Input character
      );
```

Description

The **iscntrl** function returns nonzero if the input character is a control character or zero otherwise.

isdigit - Test for numeric character (STD)

Synopsis

```
int isdigit(                  // Returns boolean test result
      char;                   // Input character
      );
```

Description

The **isdigit** function returns nonzero if the input character is numeric or zero otherwise.

isgraph - Test for visible character (STD)

Synopsis

```
int isgraph(                  // Returns boolean test result
      char;                   // Input character
      );
```

Description

The **isgraph** function returns nonzero if the input character is visible or zero otherwise.

islower - Test for lowercase alphabetic character (STD)

Synopsis

```
int islower(                  // Returns boolean test result
      char;                   // Input character
      );
```

Description

The **islower** function returns nonzero if the input character is lowercase alphabetic or zero otherwise.

isprint - Test for printing character (STD)

Synopsis

```
int isprint(                  // Returns boolean test result
      char;                   // Input character
      );
```

Description

The **isprint** function returns nonzero if the input character is a printing character (including space) or zero otherwise.

ispunct - Test for punctuation character (STD)

Synopsis

```
int ispunct(                    // Returns boolean test result
      char;                     // Input character
      );
```

Description

The **ispunct** function returns nonzero if the input character is a punctuation character (i.e., a printing character that is not a digit, letter or space) or zero otherwise.

isspace - Test for whitespace character (STD)

Synopsis

```
int isspace(                    // Returns boolean test result
      char;                     // Input character
      );
```

Description

The **isspace** function returns nonzero if the input character is a whitespace character (i.e., a space ' ', a form feed **\f**, a horizontal tab **\t**, a newline **\n**, a carriage return **\r** or a vertical tab **\v**) or zero otherwise.

isupper - Test for uppercase alphabetic character (STD)

Synopsis

```
int isupper(                    // Returns boolean test result
      char;                     // Input character
      );
```

Description

The **isupper** function returns nonzero if the input character is uppercase alphabetic or zero otherwise.

isxdigit - Test for hexadecimal numeric character (STD)

Synopsis

```
int isxdigit(                   // Returns boolean test result
      char;                     // Input character
      );
```

Description

The **isxdigit** function returns nonzero if the input character is hexadecimal numeric or zero otherwise.

kbhit - Test if key hit (STD)

Synopsis

```
int kbhit(                      // Keyboard state
      );
```

Description

The **kbhit** function returns the current state of the keyboard input. A zero value is returned if no key was pressed. A nonzero return value signals a key was pressed. The key code remains in the keyboard input buffer and can be read with **getchr** or other keyboard input functions.

See also
Function **kbstate**.

kbstate - Shift/control/alt key state query (STD)

Synopsis

```
int kbstate(                  // Keyboard state (bit values):
                              //    0x***1 = Shift pressed
                              //    0x***2 = Ctrl pressed
                              //    0x**1* = Left Alt key pressed
                              //    0x**2* = Right Alt key pressed
     );
```

Description

The **kbstate** function can be used to check whether `Shift`, `Ctrl` and/or `Alt` keys are currently pressed.

See also
Function **kbhit**.

launch - Pass command to operating system without waiting for completion (STD)

Synopsis

```
int launch(                    // Startup status
      string;                  // Command string
      );
```

Description

The **launch** function activates and/or executes the command specified in the command string parameter. The command is used to start and/or execute an application, and control is regained by BAE immediately after passing the command to the Windows operating system (the called application runs independentely from BAE). The function returns zero if the command was successfully launched or nonzero otherwise.

Limitations

The **launch** function does *not* work in **BAE Demo** software configurations.

Requirements

Executing MS-DOS (child) processes through the DOS Extender requires enough conventional memory to be available for running the executable. Conventional memory must be controlled with the **-MINREAL** and **-MAXREAL** variables of the **Phar Lap 386|DOS Extender**. For running **User Language** programs using the **launch** function, the corresponding **User Language Interpreter** environments must be re-configured by applying Phar Lap's redistributed CFIG386 tool as in

```
>  cfig386 <EXEFILE> -maxreal 0ffffh ↵
```

where **<EXEFILE>** must be set to the appropriate **User Language Interpreter** executable(s) (**scm.exe**, **ged.exe**, **neurrut.exe**, **cam.exe**, **gerview.exe** and/or **ced.exe**).

Warnings

Note that the **launch** function introduces basic multi-processing/multi-tasking features which are not fully supported on PC-based systems or can cause some problems on network-based workstation systems (depending on whichever OS command is to be executed).

It is strongly recommended to redirect DOS command standard output to temporary files (and use some file view **User Language** function for display); otherwise DOS standard output overwrites the BAE graphic user interface.

Erroneous DOS command calls cause error output to the screen, thus overwriting the BAE graphic interface. Due to the fact that DOS lacks from some substantial standard features such as redirect error output, this problem can only be solved by refraining and/or preventing from running erroneous DOS commands, e.g., by pre-checking the consistency of each DOS command to be called.

It is strongly recommended to refrain and/or prevent from calling interactive DOS commands and/or application software with the **launch** function since otherwise the system will "hang up" due to the fact that DOS standard input cannot be redirected from the BAE graphic user interface. It is also strongly recommended to refrain and/or prevent from *directly* calling UNIX commands which expect some user input (such as `more`, `vi`, etc.); this problem can be solved by a command cast to background (&) command shell start where the desired command should be called from (which however might cause a terminal device connection problem under remote login).

See also
Function **system**.

ldexp - Multiply by a power of 2

Synopsis

```
double ldexp(                  // Returns result value
      double;                  // Input value
      & int;                   // Exponent
      );
```

Description

The **ldexp** function calculates and returns the result of (input double value) multiplied with (2 power exponent value).

localtime - Get local processor system date and time (STD)

Synopsis

```
double localtime(              // Returns elapsed CPU time (in seconds)
      & int;                   // Returns seconds after minute (0..59)
      & int;                   // Returns minutes after hour (0..59)
      & int;                   // Returns hours since midnight (0..23)
      & int;                   // Returns day of month (1..31)
      & int;                   // Returns month of year (0..11)
      & int;                   // Returns years since 1900
      & int;                   // Returns days since Sunday (0..6)
      & int;                   // Returns days of year (0..365)
      );
```

Description

The **localtime** functions returns with its parameters the system time and date (including weekday and day of year). The function returns the elapsed CPU time (in seconds).

log - Natural logarithm; base e (STD)

Synopsis

```
double log(                    // Returns result value
      double ]0.0,[;           // Input value
      );
```

Description

The **log** function calculates and returns the natural logarithm (base e) for the given non-negative input double value.

log10 - Common logarithm; base ten (STD)

Synopsis

```
double log10(                  // Returns result value
      double ]0.0,[;           // Input value
      );
```

Description

The **log** function calculates and returns the common logarithm (base ten) for the given non-negative input double value.

mkdir - Create directory (STD)

Synopsis

```
int mkdir(                    // Returns status
      string;                 // Directory path name
      );
```

Description

The **mkdir** creates a directory with the specified directory path name. The functions returns zero if the directory was successfully created or (-1) if the creation of the directory failed due to missing or wrong parameters or directory access error.

modf - Break double into integer and fractional parts (STD)

Synopsis

```
double modf(                  // Returns fractional value
      double;                 // Input value
      & double;               // Returns integer value
      );
```

Description

The **modf** function breaks the given input double value into integer and fractional parts. The resulting "integer" value is passed with the second parameter. The resulting fractional part is passed with the function return value.

namestrcmp - Name string compare (STD)

Synopsis

```
int namestrcmp(               // Returns comparison result
      string;                 // First name
      string;                 // Second name
      );
```

Description

The **namestrcmp** function compares the specified input strings. A case-insensitive character-by-character alphanumeric comparison is applied. The function returns zero if the strings are equal, (-1) if the first string is smaller than the second string, or 1 if the first string is greater than the second string.

See also

Functions **numstrcmp**, **strcmp**.

numstrcmp - Numeric string compare (STD)

Synopsis

```
int numstrcmp(                // Returns comparison result
      string;                 // First input string
      string;                 // Second input string
      );
```

Description

The **numstrcmp** function compares the specified input strings. The function returns zero if the strings are equal, (-1) if the first string is smaller than the second string or 1 otherwise. **numstrcmp** operates like the **strcmp** function unless for numeric parts of the compare strings for which numeric comparison is applied. I.e., this results in sequences such as R1, R2, ..., R10, R11 (instead of R1, R10, R11, ..., R2).

See also

Functions **namestrcmp**, **strcmp**.

perror - Print error message (STD)

Synopsis

```
void perror(
      string;                 // Message string
      );
```

Description

The **perror** function displays the specified message string in the status line of the BAE user interface. The status line display is inverted for a short moment (single blink effect) to attract the user's attention. The status line is cleared if an empty message string is passed to the function.

See also

Function **bae_prtdialog**.

pow - Raise a double to a power (STD)

Synopsis

```
double pow(                   // Returns result value
      double;                 // Base input value
      double;                 // Exponent input value
      );
```

Description

The **pow** function calculates and returns the value of (base input value) power (exponent input value).

printf - Print to standard output using format (STD)

Synopsis

```
void printf(
      string;                 // Format string
      []                      // Parameter list
      );
```

Description

The **fprintf** function writes the data contained in the output parameter list to the BAE text output workarea. The format string contains information on how to format the output (see also description of the **fprintf** function).

See also

Functions **fprintf**, **sprintf**.

programid - Get current program name (STD)

Synopsis

```
string programid(             // Returns program name
      );
```

Description

The **programid** function returns the name of the currently active **User Language** program.

putchr - Write a character to standard output (STD)

Synopsis

```
int putchr(                   // Returns status
      char;                   // Character
      );
```

Description

The **putchr** function writes the specified character to the BAE text output workarea. The function returns nonzero on error.

putenv - Set environment variable (STD)

Synopsis

```
int putenv(                    // Returns status
      string;                  // Variable name
      string;                  // Variable value
      );
```

Description

The **putenv** function assigns a value to the specified operating system environment variable. The function returns zero if the assignment was successfull or (-1) if the variable was not found.

See also

Function **getenv**.

puts - Write a string to standard output (append NL) (STD)

Synopsis

```
int puts(                      // Returns status
      string;                  // String
      );
```

Description

The **puts** function writes the specified string to the BAE text output workarea and appends a newline. The function returns nonzero on error.

putstr - Write a string to standard output (STD)

Synopsis

```
int putstr(                    // Returns status
      string;                  // String
      );
```

Description

The **puts** function writes the specified string to the BAE text output workarea. The function returns nonzero on error.

quicksort - Sort index list (STD)

Synopsis

```
int quicksort(                  // Returns status
      & void;                   // Index list (integer array)
      int;                      // Index count
      * int;                    // Element compare function
      );
```

Description

The **quicksort** function sorts the specified index list. The return value is 0 if the list was successfully sorted, or (-1) on error.

Element compare callback function

```
int sortfuncname(
      int idx1,                 // Index 1
      int idx2,                 // Index 2
      )
{
      // Compare index 1 to index 2
      :
      return(compareresult);
}
```

The return value of the element compare function must be (-1) if the first index is smaller than the second index, 1 if the first index is greater than the second index, or 0 if both index values are equeal.

remove - Delete a file or directory (STD)

Synopsis

```
int remove(                     // Returns status
      string;                   // Path name
      );
```

Description

The **remove** function deletes the file or directory with the specified path name. The function returns nonzero on error.

rename - Change the name of a file (STD)

Synopsis

```
int rename(                     // Returns status
      string;                   // Old file name
      string;                   // New file name
      );
```

Description

The **rename** function changes the name of a file. The function returns nonzero on error.

rewind - Seek to the beginning of a file (STD)

Synopsis

```
void rewind(
      int;                      // File handle
      );
```

Description

The **rewind** function sets the file pointer of the file specified with the given file handle to the beginning of this file.

rulecompile - Compile a rule definition (STD)

Synopsis

```
int rulecompile(              // Returns status
      string;                 // Destination file name
      string;                 // Rule name
      string;                 // Rule code
      );
```

Description

The **rulecompile** function compiles the specified rule code and saves the compiled rule with the given name to the destination file. The function returns zero on success or non-zero on error.

See also

Function **rulesource**.

rulesource - Get rule definition source code (STD)

Synopsis

```
int rulesource(               // Returns status
      string;                 // Rule database file name
      string;                 // Rule name
      & string;                 // Rule source code
      );
```

Description

The **rulesource** function retrieves the source code for the named rule definition from the specified rule database file. The rule definition source code is returned as string through the corresponding function parameter. The functions returns zero if the query was successful, (-1) if parameters are missing or invalid, (-2) if the rule database access failed, (-3) if the rule definition wasn't found or (-5) if the rule definition access failed.

See also

Function **rulecompile**.

scanddbenames - Scan DDB file element names (STD)

Synopsis

```
int scanddbenames(            // Returns scan status
      string;                 // DDB file name
      int [0,[;               // DDB element class (STD1) or zero for cache flush
      & string;               // Input/output element name
      );
```

Description

The **scanddbenames** function scans the DDB file element following the one specified (the name of the scanned element is passed with the element name parameter). The first DDB file element is scanned if the input element name is an empty string. The scan works only in the specified DDB element class. The function returns 1 if an element has been found, 0 if no next element has been found or (-1) on DDB file access errors or invalid parameters.

scandirfnames - Scan directory file names (STD)

Synopsis

```
int scandirfnames(            // Returns scan status
      string;                 // Directory path name
      string;                 // Name extension:
                              //    .EXT = extension .EXT
                              //    .*   = all files/subdirectories
      & string;               // Input/output name
      );
```

Description

The **scandirfnames** function scans the specified directory for the file or directory name entry following the input name specification; the name of the scanned file or directory is passed with the output file name parameter. The first directory name entry is scanned on empty string name input. The scan works only in the specified extension match. The function returns 1 if a name entry has been found, 0 if no next name entry has been found or (-1) on directory access errors or invalid parameters.

setprio - Set BAE process priority (STD)

Synopsis

```
void setprio(
      int;                    // Process priority value
                              // - Unix/Linux:
                              //   nice priority value
                              // - Windows:
                              //   <0 = HIGH_PRIORITY_CLASS
                              //    0 = NORMAL_PRIORITY_CLASS
                              //   >0 = IDLE_PRIORITY_CLASS
                              // - otherwise ignored
      );
```

Description

The **setprio** function sets the priority of the current BAE process according to the specified process priority value.

sin - Sine (STD)

Synopsis

```
double sin(                   // Returns result value
      double;                 // Input angle value (STD3)
      );
```

Description

The **sin** function calculates and returns the sine value of the given angle value. The input angle value must be in radians.

sinh - Hyperbolic Sine (STD)

Synopsis

```
double sinh(                  // Returns result value
      double;                 // Input angle value (STD3)
      );
```

Description

The **sinh** function calculates and returns the hyperbolic sine value of the given angle value. The input angle value must be in radians.

sprintf - Print to string using format (STD)

Synopsis

```
int sprintf(                  // Returns decoded character count
      & string;               // Output string
      string;                 // Format string
      string;                 // Format string
      []                      // Parameter list
      );
```

Description

The **sprintf** function writes the data contained in the output parameter list to the output string, i.e., **sprintf** performs a format conversion in memory. The format string contains information on how to format the output (see also the description of the **fprintf** function). The function returns the number of successfully decoded characters, i.e., the length of the resulting output string.

See also

Functions **fprintf**, **printf**.

sqlcmd - SQL command execution (STD)

Synopsis

```
int sqlcmd(                    // Returns status
      string;                  // Database file name
      string;                  // SQL command
      * int;                   // Data return function
      );
```

Description

The **sqlcmd** function is used to control the data transfer from and to a relational database system initialized with **sqlinit**. This is done via a SQL like query language. The first parameter of the function **sqlcmd** specifies the name of the database file to be opened for usage. The second parameter specifies the SQL command to be executed for database access. The selected data fields are returned to the caller via the data callback function specified with the third parameter. If no data callback function is used the third function parameter should be set to **NULL**. The function returns nonzero on missing or invalid parameters or on SQL database errors. In case of a database error, the exact error reason can be determined by the **sqlerr** function.

SQL Commands

The database query language interpreted by **sqlcmd** is a restricted subset of the Structured Query Language (SQL) for relational databases. The following basic commands are supported:

create table
drop table
insert into table values
quickinsert into table values
index table
select from table
delete from table
help

The following data types are supported for the representation of data:

Keyword	Data Type
integer	Integer values in range [-2147483648, 2147483647]
float	Float values in range [-10^308, 10^308] with a precision of about 15 leading digits
string	Strings (enclosed with apostrophes)
boolean	Logical value (**FALSE** or **TRUE**)
date	Date; on input **dd/mm/yyyy**, on output **yyyymmdd**

Command `create table`

The **create** command generates a database table structure. The command requires a table name and a list of field names and corresponding data types. The field name sequence defined with this list is used for data field output unless an explicit output field order is specified. The syntax of the **create** command is:

```
create table tablename ( name1 type1, ..., namen typen ) ;
```

An index is generated for each database field. Indices are automatically during database queries. The user has not to care about the optimal use of indices. The index for string variables is restricted to the first 39 characters.

Command drop table

The **drop** command deletes a table structure with table all entries from the database file. The syntax of the **drop** command is:

```
drop table tablename ;
```

Command insert into

The **insert** command stores a data record to a table. The specified values must match the table definition in number and data type sequence. The syntax of the **insert** command is:

```
insert into tablename values ( val1, ..., valn ) ;
```

Command quickinsert insert into

The **quickinsert** is similar to the **insert** command. However, unlike **insert**, **quickinsert** does not update any field indices after storing data records into a database table. The **index table** command (see below) can be used for field index updates after (multiple) **quickinsert** applications. Performing a single **index table** call is much quicker than updating indices through repeated **insert** calls. I.e., **quickinsert** and **index table** are useful for quickly inserting large numbers of records into a database. Please note however that data entered with **quickinsert** is not included in any query results until a **index table** takes place.

Command index table

The **index table** command is used to create the field indices for data records which have been inserted into a database table with the **quickinsert** command (see above). The syntax of the **index** command is:

```
index table tablename ;
```

Command `select from`

The **`select`** command provides a query to the database contents. The set of output records can be restricted to records matching some condition using the optional **`where`** clause. All records are returned if no **`where`** clause is specified. The syntax of the **`select`** command is:

```
select [ field1, ..., fieldn ] from table1, ..., tablen
       [ where ... ] ;
```

The specification of output fields is optional. All database fields are be returned in the order of their definition if no output fields are given. If an output field is defined in more than one table, then this ambiguity must be resolved by the specification of the field name in the form **`table.field`**. The same specification must be given, when referring to that field in the **`where`** clause. The **`where`** clause consists of comparisons which can be combined with the logical operators **`AND`** (logical and), **`OR`** (logical or) and **`NOT`** (logical negation). The comparison operators **`=`** (equal), **`<>`** (not equal), **`>`** (greater than), **`>=`** (greater than or equal), **`<`** (less than) and **`<=`** (less than or equal) are valid for all data types. The **`PREVTO`** (select previous similar to pattern string) and **`NEXTTO`** (select next similar to pattern string) operators and the string pattern match operator **`LIKE`** are additionally provided for the string data type. The right side of these operators represents a pattern string which can contain the wildcard characters **`%`** (specifying arbitrary substring) and **`?`** (specifying arbitrary character). The operators of a comparison expression must be type compatible, i.e., of the same data type. The only exception to this rule is the combination of **`integer`** and **`float`** data types. Terms can consist of operators combining data field references and constants. The operators are **`+`** (addition), **`-`** (subtraction), **`*`** (multiplication), **`/`** (division), **`%`** (division modulo), **`UPPER`** (string conversion to upper case) and **`LOWER`** (string conversion to lower case). The following table lists the valid combinations of operators and data types of referenced data fields or constants:

Operator	Data Type
`+`	`integer`, `float`, `string`
`-`	`integer`, `float`
`*`	`integer`, `float`
`/`	`integer`, `float`
`%`	`integer`, `float`
`UPPER()`	`string`
`LOWER()`	`string`

The data types **`date`** and **`boolean`** can be combined by none of the defined operators.

Command `delete from`

The **`delete`** command deletes data records from a database table. The syntax of the **`delete`** command is:

```
delete from tablename [ where ... ] ;
```

The optional **`where`** clause syntax accords to the **`where`** clause of the **`select`** command. The **`delete`** command deletes the data records from a table matching the condition specified with the **`where`** clause. All data records of the specified table are deleted if no **`where`** clause is specified.

Command cache

The **cache** command can be used for optionally keeping SQL databases open for multiple read/write SQL operations. These features can be used to avoid time-consuming database open and close operations between multiple SQL read/write operations. Facilitating the SQL cache can increase the performance of certain applications dramatically, especially if SQL databases are accessed over a network. The syntax of **cache** supports the following commands:

```
cache read on ;
cache write on ;
cache off ;
```

Command help

The **help** command can be used for retrieving information about the database table structures. The syntax of the **help** command is:

```
help [ tablename ] ;
```

The data retrieved with the **help** command is returned to the caller by the data callback function. The table name specification is optional. If no table name is given, then the **help** command returns the names of the defined database tables via the table name parameter of the data callback function, i.e., the data callback function is called once per table. If a table name is given, then the **help** command returns the table field names and data types via the corresponding parameters of the data callback function, i.e. the data callback function is called once for each table field.

Data return function

```
int datafunc(
      string dstr,            // String or date
      int  dint,              // Integer or logical value
      double ddbl,            // Float value
      int dval,               // Data valid flag:
                              //   0 = invalid data
                              //   1 = valid data
      int dtype,              // Data field type:
                              //   2 = integer
                              //   3 = float
                              //   4 = string
                              //   5 = date (format "yyyymmdd")
                              //   6 = logical value (0=FALSE,1=TRUE)
      string dtable,          // Table name
      string dfield,          // Data field name
      int didx                // Data output field index
      )
{
      // Data return function statements
      :
      return(errstat);
}
```

The data callback function is used to return selected data fields to the caller of **sqlcmd**. The function is automatically called once for each data field. Hence this function is called fifty times if 10 data records with 5 data fields each are selected. The index of the data output field specifies the position of the output field in the current record. It ranges from 1 to the number of defined data output fields. The return value of the data callback function should be zero if no (semantic) error occurred. In case of an error, a nonzero value must be returned to abort the database query.

Warning

The **sqlcmd** function operates directly on database file level and is therefore not attached to the Undo/Redo mechanism.

See also

Functions **sqlerr**, **sqlinit**.

Example

Definition of a table named **partdata** with the **symname** (String), **val** (String) and **partno** (String) data fields in the **partdata.dat** database file:

```
if (sqlinit("partdata.dat",1)!=0)
{
    perror("SQL Init error!");
    exit(0);
}
if (sqlcmd("partdata.dat",
 "create table partdata (symname string,val string,partno string);",
 NULL)!=0)
{
    perror("SQL Query error!");
    exit(0);
}
```

Data insertion:

```
if (sqlcmd("partdata.dat",
 "insert into partdata values ('r','470','STK100470');",NULL)!=0)
{
    perror("SQL Data input error!");
    exit(0);
}
```

Data query:

```
if (sqlcmd("partdata.dat",
 "select partno from partdata where symname='r' AND val='470';",
 datafunc)!=0)
{
    perror("SQL Query error!");
    exit(0);
}
:
int datafunc(dstr,dint,ddbl,dval,dtype,dtable,dfield,didx)
string dstr;
int dint;
double ddbl;
int dval,dtype;
string dtable,dfield;
int didx;
{
        printf("Part Number : %s\n");
}
```

Data delete:

```
if (sqlcmd("partdata.dat",
 "delete from partdata where symname='r';",NULL)!=0)
{
    perror("SQL Delete error!");
    exit(0);
}
```

sqlerr - SQL error status query (STD)

Synopsis

```
void sqlerr(
      & int;                  // Error code
      & string;               // Error item string
      );
```

Description

The **sqlerr** function is used to determine the error reason after an unsuccessful call of the **sqlcmd** function.

Diagnosis

The error reason can be determined by the parameters returned by the **sqlerr** function. The error item string identifies the element which caused the error. The following table lists the possible error codes:

Error Code	Meaning
0	SQL command executed without errors
1	SQL command read error (internal)
2	SQL command too complex (more than 2000 terms)
3	Invalid numeric expression
4	SQL command file not found (internal)
5	SQL command item too long (more than 200 characters)
6	SQL command syntax error at **<c>**
7	General SQL command parser error
8	Error creating database
9	File access error
10	Too many open files
11	File **<d>** is of wrong type/not a database
12	Database file structure is damaged
13	Database file structure is invalid
14	Key **<k>** not found
15	Key **<k>** already defined
16	File **<d>** not found
17	Table element **<t>.<f>** has invalid data type
18	Too many table elements
19	Data entry length exceeds database limit
20	Multiple table delete not allowed
21	Command contains illegal type combination
22	Output field **<f>** undefined/not in any table
23	Output field **<f>** defined in different tables
24	Output table **<t>** undefined in **from** table list
25	Table **<t>** already defined
26	Database class limit exceeded
27	Table **<t>** not found
28	Error from data callback function

29	No **delete** record found
30	Unknown/new database format
31	Query field not in table(s)
32	Query field in multiple tables
33	File read access denied
34	File write access denied
35	General database error

Depending on the error condition the error item string can describe a command element **<c>**, a database file **<d>**, a key **<k>**, a table **<t>** or a data field **<f>**.

See also

Functions **sqlcmd**, **sqlinit**.

sqlinit - SQL database initialization (STD)

Synopsis

```
int sqlinit(                  // Returns status
      string;                 // Database file name
      int;                    // Database init mode
      );
```

Description

The **sqlinit** function initializes and/or creates a database system for further access by the **sql*** functions. The database init mode can be zero for initializing an existing database file (**.ddb**, **.dat**) or 1 to create and initialize a new database file. The function return value is zero on successful initialization, 1 if the database file is already initialized or another value if an error occurred on initializing the database file. Databases processed with the **sql*** functions are stored in **Bartels AutoEngineer** DDB format. This introduces powerful features for including relational databases with **AutoEngineer** library or design data. PPS data can be stored together with layout design data or part attributes can be set with data retrieved from a separate database. The relational database system uses the DDB classes ranged from 4096 to 8191. DDB class 4096 contains the structure of the defined tables. DDB class 4097 is used to manage free DDB classes; this class contains just one entry named **info**, which is created by the **sqlinit** function. The existence of this entry can be used to check whether a database is initialized or not. The DDB classes 4352 to 8191 are assigned dynamically when storing user-defined table records and indices. The number of DDB classes used by each table corresponds to the number of table fields plus 1.

See also

Functions **sqlcmd**, **sqlerr**.

sqrt - Square root (STD)

Synopsis

```
double sqrt(                  // Returns result value
      double [0.0,[;          // Input value
      );
```

Description

The **sqrt** function calculates and returns the square root of the given double input value.

strcmp - String compare (STD)

Synopsis

```
int strcmp (                    // Returns comparison result
      string;                   // First string
      string;                   // Second string
      );
```

Description

The **strcmp** function compares the two input strings. A character-by character alphanumeric comparison is applied. The function returns zero if the strings are equal, (-1) if the first string is smaller than the second string or 1 otherwise.

See also

Functions **namestrcmp**, **numstrcmp**.

strcspn - String prefix length not matching characters (STD)

Synopsis

```
int strcspn(                    // Returns match position
      string;                   // Test string
      string;                   // Suffix match pattern
      );
```

Description

The **strcspn** function returns the number of test string characters not matching any character contained in the suffix match pattern string. The test string is searched from start.

strdelchr - Delete characters from string (STD)

Synopsis

```
void strdelchr(
      & string;                 // Input/output string
      string;                   // Delete character set
      int;                      // Delete start position
      int;                      // Delete end position
      );
```

Description

The **strdelchr** function deletes from the input string all characters contained in the delete character set. The string is scanned from delete start to delete end position.

strextract - Extract sub-string from another string (STD)

Synopsis

```
string strextract(              // Returns extracted string
      string;                   // Input string
      int;                      // Extract start position (0 - strlen-1)
      int;                      // Extract end position (0 - strlen-1)
      );
```

Description

The **strextract** function extracts and returns the substring of the input string, which starts at the specified extract start position (counting from zero) and ends at the extract end position. The extract start and end positions are automatically adjusted to the input string boundaries. A reverse string extraction is applied, if the end position is smaller than the start position.

strextractfilepath - Extract directory name from a file path name string (STD)

Synopsis

```
string strextractfilepath(    // Returns directory name
      string;                 // Path name
      );
```

Description

The **strextractfilepath** function extracts and returns the directory name from the specified path name.

See also

Function **strgetpurefilename**.

strgetconffilename - Get environment variable expanded configuration file name (STD)

Synopsis

```
string strgetconffilename(    // Returns configuration file path name
      string;                 // Environment variable name
      string;                 // File base name
      int;                    // Directory preference:
                              //    0 : Prefer program directory
                              //    1 : Prefer all users directory
                              //    2 : Prefer user directory
      );
```

Description

The **strgetconffilename** returns the configuration file path name defined through the specified environment variable. The configuration file search is carried out in different configuration file directories according to the specified directory preference.

See also

Function **strgetvarfilename**.

strgetvarfilename - Get environment variable expanded file name string (STD)

Synopsis

```
string strgetvarfilename(     // Returns file name
      string;                 // Environment variable name
      );
```

Description

The **strgetvarfilename** function returns the file and/or path name defined through the specified environment variable.

See also

Function **strgetconffilename**.

strgetpurefilename - Extract file name from file path name string (STD)

Synopsis

```
string strgetpurefilename(    // Returns file name
      string;                 // Path name
      );
```

Description

The **strgetpurefilename** function extracts and returns the file name from the specified path name.

See also

Function **strextractfilepath**.

strlen - String length (STD)

Synopsis

```
int strlen(                    // Returns string length
      string;                  // Test string
      );
```

Description

The **strlen** function determines and returns the length of the specified test string (the **NUL** delimiter character is ignored).

strlistitemadd - Add string to string list (STD)

Synopsis

```
void strlistitemadd(
      & string;                // Comma-separated string list
      string;                  // String
      );
```

Description

The **strlistitemadd** adds the specified string in the comma-separated string list.

See also

Function **strlistitemchk**.

strlistitemchk - Search string in string list (STD)

Synopsis

```
int strlistitemchk(            // Search result
      string;                  // Comma-separated string list
      string;                  // Search string
      );
```

Description

The **strlistitemchk** searches the specified search string in the comma-separated string list. The functions returns zero if the search string is not in the string list, 1 if the search string is in the string list, or 2 if the search string matches the full length string list.

See also

Function **strlistitemadd**.

strlower - Convert string to lower case (STD)

Synopsis

```
void strlower(
      & string;                // Input/output string
      );
```

Description

The **strlower** function transforms the uppercase characters of the input string to lowercase.

strmatch - Test for string pattern match (STD)

Synopsis

```
int strmatch(                 // Returns string match flag
      string;                 // Test string
      string;                 // Pattern string (can contain wildcards)
      );
```

Description

The **strmatch** function checks, whether the test string matches the pattern string. The pattern string can contain wildcards. The ***** character refers to any character sequence and **?** refers to any single character. The function returns nonzero if the test string matches the pattern string or zero otherwise.

strnset - Fill part or all of string with any character (STD)

Synopsis

```
void strnset(
      & string;               // Input/output string
      char;                   // Fill character
      int;                    // Fill count
      );
```

Description

The **strnset** function replaces the leftmost characters of the input string with the specified fill character. The number of characters to be changed is specified with the fill count.

strreverse - Reverse string (STD)

Synopsis

```
void strreverse(
      & string;               // Input/output string
      );
```

Description

The **strreverse** function reverses the character sequence of the given input string.

strscannext - Forward find characters in string (STD)

Synopsis

```
int strscannext(              // Returns match position
      string;                 // Test string
      string;                 // Match character set
      int;                    // Scan start position (1 - strlen)
      int;                    // Stop-on-match flag:
                              //    0 = continue on match
                              //    1 = stop on match
      );
```

Description

The **strscannext** function scans the test string for characters contained in the match character set. The scan starts from the specified scan start position (counting from 0) and proceeds towards the end of the test string. If the stop-on-match flag is set (1), the scan stops at the first character matching the match character set. If the stop-on-match flag is not set (0), the scan stops at the first character not matching the match character set. The function returns the match character position (or the test string length plus 1 on mismatch). The scan start position is automatically adjusted to the input string boundaries.

strscanprior - Backward find characters in string (STD)

Synopsis

```
int strscanprior(             // Returns match position
      string;                 // Test string
      string;                 // Match character set
      int;                    // Scan start position
      int;                    // Stop-on-match flag:
                              //    0 = continue on match
                              //    1 = stop on match
      );
```

Description

The **strscanprior** function scans the test string for characters contained in the match character set. The scan starts from the specified scan start position and proceeds towards the beginning of the test string. If the stop-on-match flag is set (1), the scan stops at the first character matching the match character set. If the stop-on-match flag is not set (0), the scan stops at the first character not matching the match character set. The function returns the match character position (or zero on mismatch). The scan start position is automatically adjusted to the input string boundaries.

strset - Fill string with any character (STD)

Synopsis

```
void strset(
      & string;               // Input/output string
      char;                   // Fill character
      );
```

Description

The **strset** function replaces all characters of the input string with the specified fill character.

strspn - String prefix length matching characters (STD)

Synopsis

```
int strspn(                   // Returns mismatch position
      string;                 // Test string
      string;                 // Prefix match pattern
      );
```

Description

The **strspn** returns the number of test string characters matching characters contained in the prefix match pattern string. The test string is searched from start.

strupper - Convert string to uppercase (STD)

Synopsis

```
void strupper(
      & string;               // Input/output string
      );
```

Description

The **strupper** transforms the lowercase characters of the input string to uppercase.

syngetintpar - Get BNF/scanner integer parameter (STD)

Synopsis

```
int syngetintpar(             // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = String-Kontrollzeichenauswertungsmodus
                              //    1 = Activate comment text callback function
                              //    2 = Any identifier character flag
      & int;                  // Returns parameter value
      );
```

Description

The **syngetintpar** function is used to query **User Language** BNF/syntax scanner integer parameters previously set with **synsetintpar**. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **synparsefile**, **synparseincfile**, **synparsestring**, **synsetintpar**.

synparsefile - BNF/parser input file scan (STD)

Synopsis

```
synparsefile(                 // Returns scan status
      string;                 // Input file name
      [];                     // Optional comment callback function
      );
```

Description

The **synparsefile** function activates a parser for scanning the name-specified input file. The input file is processed according to the data format described with the BNF definition of the corresponding **User Language** program. The BNF-defined parser action functions automatically called. The **synscanline** and **synscanstring** functions can be utilized in parser action functions to query the current input scan line number and the current input scan string. On request, the input scan string can be subject to semantic tests. The **synparsefile** function is terminated if the end of the input file is reached or if a syntax error (or a semantic error encountered by a parser action function) has occurred.

Comment Callback Function

The second function parameter allows for the specification of a comment text callback function. This function is activated if the corresponding scanner/parser parameter has been set with the **synsetintpar** function. The callback function definition is as follows:

```
int commentfuncname(
      string commentstring,   // Comment without comment delimiters
      )
{
      // Function statements
      :
      return(stat);
}
```

The parser is stopped if the comment callback function returns a value other than zero. Otherwise the parser continues.

Diagnosis

The **synparsefile** function return value denotes a certain scan status according to the following table:

Return Value	Meaning
0	No error
1	No BNF definition available
2	Parser (**synparsefile**) is already active
3	File open error
4	Too many open files
5	Fatal read/write error
6	Scan item too long
7	Syntax error
8	Unexpected end of file
9	Stack overflow (BNF too complex)
10	Stack underflow (BNF erroneous)
11	Error from parser action function

See also

Functions **syngetintpar**, **synparseincfile**, **synparsestring**, **synscaneoln**, **synscanigncase**, **synscanline**, **synscanstring**, **synsetintpar**, and chapter 2.6.4 of this manual.

synparseincfile - BNF/parser include file scan (STD)

Synopsis

```
int synparseincfile(          // Returns status
      string;                 // Include file name
      );
```

Description

The **synparseincfile** function can be utilized for processing include files when parsing input files with the **synparsefile** function. The parser starts reading at the beginning of the name-specified include file when calling the **synparseincfile** function. The **EOFINC** terminal symbol is returned if the end of the include file is reached, and reading continues where it was interrupted in the previously processed file. The function returns zero if no error occurred, (-1) on include file open errors or (-2) if the parser (i.e., the **synparsefile** function) is not currently active.

Warning

The **EOFINC** terminal symbol is required in the BNF definition whenever the **synparseincfile** function is used; otherwise the parser issues an unexpected symbol syntax error when reaching the end of an include file. The **EOFINC** terminal symbol is obsolete if the **synparseincfile** function is not applied.

See also

Functions **syngetintpar**, **synparsefile**, **synparsestring**, **synscaneoln**, **synscanigncase**, **synscanline**, **synscanstring**, **synsetintpar**, and chapter 2.6.4 of this manual.

synparsestring - BNF/parser input string scan (STD)

Synopsis

```
synparsestring(               // Returns scan status
      string;                 // Input string
      [];                     // Optional comment callback function
      );
```

Description

The **synparsestring** function activates a parser for scanning the given input string. The input string is processed according to the data format described with the BNF definition of the corresponding **User Language** program. The BNF-defined parser action functions are automatically called. In these parser action functions, the **synscanline** and **synscanstring** functions can be utilized for getting the current input scan line number and the current input scan string. On request, the input scan string can be subject to semantic tests. The **synparsestring** function is terminated if the end of the input string is reached or if a syntax error (or a semantic error encountered by a parser action function) has occurred.

Comment Callback Function

The second function parameter allows for the specification of a comment text callback function. This function is activated if the corresponding scanner/parser parameter has been set with the **synsetintpar** function. The callback function definition is as follows:

```
int commentfuncname(
      string commentstring,   // Comment without comment delimiters
      )
{
      // Function statements
      :
      return(stat);
}
```

The parser is stopped if the comment callback function returns a value other than zero. Otherwise the parser continues.

Diagnosis

The **synparsestring** function return value denotes a certain scan status according to the following table:

Return Value	Meaning
0	No error
1	No BNF definition available
2	Parser (**synparsestring**) is already active
3	File open error
4	Too many open files
5	Fatal read/write error
6	Scan item too long
7	Syntax error
8	Unexpected end of string
9	Stack overflow (BNF too complex)
10	Stack underflow (BNF erroneous)
11	Error from parser action function

See also

Functions **syngetintpar**, **synparsefile**, **synparseincfile**, **synscaneoln**, **synscanigncase**, **synscanline**, **synscanstring**, **synsetintpar**, and chapter 2.6.4 of this manual.

synscaneoln - BNF/scanner end-of-line recognition (STD)

Synopsis

```
int synscaneoln(              // Return status
      int [0,1];              // Scanner end-of-line recognition mode:
                              //    0 = deactivate EOLN recognition
                              //    1 = activate EOLN recognition
      );
```

Description

The **synscaneoln** function is used to enable and/or disable the end-of-line recognition for the BNF parser activated with the **synparsefile** function. The end-of-line recognition is disabled on default. The function returns nonzero on error.

Warning

The usage of the **EOLN** terminal symbol in a BNF definition is only valid if the end-of-line recognition is activated; otherwise end-of-line characters cause parser syntax errors.

See also

Functions **synparsefile**, **synparseincfile**, **synparsestring**, **synscanigncase**, **synscanline**, **synscanstring**, and chapter 2.6.4 of this manual.

synscanigncase - BNF/scanner keyword case-sensitivity mode setting (STD)

Synopsis

```
int synscanigncase(           // Return status
      int [0,1];              // Keyword case-sensitivity mode:
                              //    0 = match case
                              //    1 = ignore case
      );
```

Description

The **synscanigncase** function is used to disable and/or enable keyword case-sensitivity for the BNF parser activated with the **synparsefile** function. On default, keyword case-sensitivity is activated. The function returns nonzero on error.

See also

Functions **synparsefile**, **synparseincfile**, **synparsestring**, **synscaneoln**, **synscanline**, **synscanstring**, and chapter 2.6.4 of this manual.

synscanline - BNF/scanner input line number (STD)

Synopsis

```
int synscanline(              // Returns current scan line number
      );
```

Description

The **synscanline** function returns the input file line number currently processed by the **synparsefile** function. The **synscanline** function can be utilized in referenced parser action user functions to trace the scan process.

See also

Functions **synparsefile**, **synparseincfile**, **synparsestring**, **synscaneoln**, **synscanigncase**, **synscanstring**, and chapter 2.6.4 of this manual.

synscanstring - BNF/scanner input string (STD)

Synopsis

```
string synscanstring(         // Returns current scan string
      );
```

Description

The **synscanstring** function returns the string currently scanned by the **synparsefile** function. The **synscanstring** function can be utilized in referenced parser action user functions to check and store certain scanner input data.

See also

Functions **synparsefile**, **synparseincfile**, **synparsestring**, **synscaneoln**, **synscanigncase**, **synscanline**, and chapter 2.6.4 of this manual.

synsetintpar - Set BNF/scanner integer parameter (STD)

Synopsis

```
int synsetintpar(             // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = String control character interpretation
mode
                              //    1 = Activate comment text callback function
                              //    2 = Any identifier character flag
      int;                    // Parameter value
      );
```

Description

The **synsetintpar** function is used to set **User Language** BNF/syntax scanner integer system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **syngetintpar** function can be used to query parameter values set with **synsetintpar**.

See also
Functions **syngetintpar**, **synparsefile**, **synparseincfile**, **synparsestring**.

system - Pass command to operating system and wait for completion (STD)

Synopsis

```
int system(                    // Completion status
      string;                  // Command string
      );
```

Description

The **system** function activates and/or executes the command specified in the command string parameter. The command string is passed to the operating system command shell, and BAE waits until command execution is completed. The function returns the status code returned by the operating system, the command interpreter or the executed program (whichever is last to execute before passing control back to BAE). A zero return value usually denotes successful execution, whilst nonzero return values notify errors and/or warnings.

Limitations

The **system** function does *not* work in **BAE Demo** software configurations.

Requirements

Executing MS-DOS (child) processes through DOS Extender requires enough conventional memory to be available for running the executable. Conventional memory must be controlled with the **-MINREAL** and **-MAXREAL** variables of the **Phar Lap 386|DOS Extender**. For running **User Language** programs using the **system** function, the corresponding **User Language Interpreter** environments must be re-configured by applying Phar Lap's redistributed CFIG386 tool as in

```
>  cfig386 <EXEFILE> -maxreal 0ffffh ↵
```

where **<EXEFILE>** must be set to the appropriate **User Language Interpreter** executable(s) (**scm.exe**, **ged.exe**, **neurrut.exe**, **cam.exe**, **gerview.exe** and/or **ced.exe**).

Warnings

Note that the **system** function introduces basic multi-processing/multi-tasking features which are not fully supported on PC-based systems or can cause some problems on network-based workstation systems (depending on whichever OS command is to be executed).

It is strongly recommended to redirect DOS command standard output to temporary files (and use some file view **User Language** function for display); otherwise, DOS standard output overwrites the BAE graphic user interface.

Erroneous DOS command calls cause error output to the screen, thus overwriting the BAE graphic interface. Due to the fact that DOS lacks from some substantial standard features such as redirect error output, this problem can only be solved by refraining and/or preventing from running erroneous DOS commands, e.g., by pre-checking the consistency of each DOS command to be called.

It is strongly recommended to refrain and/or prevent from calling interactive DOS commands and/or application software with the **system** function since otherwise the system will "hang up" due to the fact that DOS standard input cannot be redirected from the BAE graphic user interface. It is also strongly recommended to refrain and/or prevent from *directly* calling UNIX commands which expect some user input (such as `more`, `vi`, etc.); this problem can be solved by a command cast to background (&) command shell start where the desired command should be called from (which however might cause a terminal device connection problem under remote login).

See also
Function **launch**.

tan - Tangent (STD)

Synopsis

```
double tan(                    // Returns result value
      double;                  // Input angle value (STD3)
      );
```

Description

The **tan** function calculates and returns the tangent value of the given angle value. The input angle value must be in radians.

tanh - Hyperbolic tangent (STD)

Synopsis

```
double tanh(                   // Returns result value
      double;                  // Input angle value (STD3)
      );
```

Description

The **tanh** function calculates and returns the hyperbolic tangent value of the given angle value. The input angle value must be in radians.

tolower - Convert uppercase to lowercase character (STD)

Synopsis

```
char tolower(                  // Returns lowercase character
      char;                    // Input character
      );
```

Description

The **tolower** function converts and returns the specified uppercase alphabetic input character to lowercase (or returns the input character unchanged if not uppercase alphabetic).

toupper - Convert lowercase to uppercase character (STD)

Synopsis

```
char toupper(                  // Returns uppercase character
      char;                    // Input character
      );
```

Description

The **toupper** function converts and returns the specified lowercase alphabetic input character to uppercase (or returns the input character unchanged if not lowercase alphabetic).

uliptype - Get User Language interpreter environment (STD)

Synopsis

```
int uliptype(                  // Interpreter type:
                               //    0x0000 = invalid/unknown
                               //    0x0080 = SCM - Schematic Editor
                               //    0x0040 = GED - Layout Editor
                               //    0x0010 = AR - Autorouter
                               //    0x0008 = CAM - CAM Processor
                               //    0x0004 = CED - Chip Editor
                               //    0x1000 = CV - CAM View
      );
```

Description

The **uliptype** function returns the type of the currently active **User Language Interpreter** environment.

ulipversion - Get User Language interpreter version (STD)

Synopsis

```
int ulipversion(               // Returns interpreter version
      );
```

Description

The **ulipversion** function returns the internal version number of the currently active **User Language Interpreter** environment.

ulproginfo - Get User Language program info (STD)

Synopsis

```
int ulproginfo(                // Returns status
      string;                  // Program name
      & int;                   // Program version
      & int;                   // Program caller type
      );
```

Description

The **ulproginfo** function gets the program version and the program caller type of a name-specified **User Language** program. The program version is the internal version number of the **User Language Compiler** used for compiling the program. The program caller type is a bit-mask value designating the **User Language Interpreter** environments compatible for executing the **User Language** program. The **ulproginfo** function can be utilized together with the **ulip*** functions to check whether a certain **User Language** program can be executed in the current **User Language Interpreter** environment.

See also
Functions **uliptype**, **ulipversion**.

ulsystem - Run another User Language program (STD)

Synopsis

```
int ulsystem(                  // Returns status
      string;                  // Program name
      & int;                   // Program counter
      );
```

Description

The **ulsystem** function executes the **User Language** program with the specified program name. The function returns nonzero if an error occurred whilst loading or running the program (see below for diagnosis). On error, the program counter parameter returns the address of the machine program instruction, which caused the error (to be compared with the listing file generated by the **User Language Compiler**).

Diagnosis

The **ulsystem** function can return the following return values:

Return Value	Meaning
0	Program successfully executed
1	DDB/database access error
2	Program already loaded
3	Program not found
4	Incompatible User Language Program Version
5	Incompatible index/function references
6	Stack underflow
7	Stack overflow
8	Division by zero
9	Error calling system function
10	System function not available
11	System function not implemented
12	User function not found
13	Invalid data type for user function
14	Invalid parameter list for user function
15	Error accessing array variable
16	Invalid array variable index
17	General file access error
18	General file read error
19	General file write error

See also
Function **ulsystem_exit**.

ulsystem_exit - Run a User Language program after exiting current User Language program (STD)

Synopsis

```
void ulsystem_exit(
      string;                 // Program name
      );
```

Description

The **ulsystem** function executes the **User Language** program with the specified program name after exiting the current **User Language** program.

See also

Functions **exit**, **ulsystem**.

vardelete - Delete global User Language variable (STD)

Synopsis

```
int vardelete(                // Returns status
      string;                 // Variable name
      );
```

Description

The **vardelete** function is used for deleting a global **User Language** variable previously defined with **varset**. The function returns zero if the variable has been successfully deleted or (-1) if no variable with the specified name is defined.

See also

Functions **varget**, **varset**.

varget - Get global User Language variable value (STD)

Synopsis

```
int varget(                   // Returns status
      string;                 // Variable name
      & void;                 // Variable value
      );
```

Description

The **varget** function is used for retrieving the value of global **User Language** variable previously defined with **varset**. The function return value is zero if the query was successful, (-1) if no variable with the specified name is defined or (-2) if the data type of the variable does not match the data type of the variable value return parameter.

See also

Functions **vardelete**, **varset**.

varset - Set global User Language variable value (STD)

Synopsis

```
int varset(                    // Returns status
      string;                  // Variable name
      void;                    // Variable value
      );
```

Description

The **varset** functions defines a global **User Language** variable with the specified variablen name and assigns the provided variable value. The variable value must match a basic data type (**int**, **double**, **char** or **string**), i.e., complex and/or combined data types such as arrays, structures or **index** types are not allowed. The function returns zero on successful variable definition, (-1) for invalid variable name specifications or (-2) if the variable value does not match a basic data type. Global **User Language** variable definitions are not deleted when the **User Language** program exits, i.e., they stay resident until they are explicitly deleted with the **vardelete** function or until the currently active BAE program module is exited. The **varget** function can be used to retrieve the value of a global **User Language** variable previously defined with **varset**. Global **User Language** variables can be used to exchange data between **User Language** programs which run at different times in the same BAE module.

See also

Functions **vardelete**, **varget**.

C.3 SCM System Functions

This section describes (in alphabetical order) the SCM system functions of the **Bartels User Language**. See Appendix C.1 for function description notations.

C.3.1 Schematic Data Access Functions

The following **User Language** system functions are assigned to caller type CAP; i.e., they can be called from the **Schematic Editor** interpreter environment of the **Bartels AutoEngineer**:

cap_blockname - Get SCM sheet block name (CAP)

Synopsis

```
string cap_blockname(         // Returns block name
      );
```

Description

The **cap_blockname** function returns the hierarchical block name of the currently loaded SCM sheet element. An empty string is returned if no SCM sheet is loaded or if the currently loaded plan is not a hierarchically defined module block.

cap_blocktopflag - Get SCM sheet block hierarchy level (CAP)

Synopsis

```
int cap_blocktopflag(         // Returns block top level flag
      );
```

Description

The **cap_blocktopflag** is used to get the hierarchical circuit design mode of the currently loaded SCM sheet. The function returns zero if the currently loaded SCM sheet is not on top hierarchy level, 1 if the SCM sheet is on top hierarchy level (normal SCM sheet) or 2 if the SCM sheet is a single reference sub block.

cap_figboxtest - Check SCM element rectangle cross (CAP)

Synopsis

```
int cap_figboxtest(           // Returns status
      & index C_FIGURE;       // Element
      double;                 // Rectangle left border (STD2)
      double;                 // Rectangle lower border (STD2)
      double;                 // Rectangle right border (STD2)
      double;                 // Rectangle upper border (STD2)
      );
```

Description

The **cap_figboxtest** function checks if the given figure list element crosses the given rectangle. The fucntion returns nonzero if the element boundaries cross the given rectangle.

cap_findblockname - Find SCM block circuit sheet with given block name (CAP)

Synopsis

```
string cap_findblockname(     // Returns block plan name
      string;                 // DDB file name
      string;                 // Block name
      );
```

Description

The **cap_findblockname** function searches the DDB file with the given name for an hierarchical SCM plan with the specified block name and returns the name of that SCM plan element or an empty string if the DDB file is not available and/or if no SCM sheet with the given block name is found in the DDB file.

cap_findlayconpart - Get layout connection list part index (CAP)

Synopsis

```
int cap_findlayconpart(       // Returns status
      string;                 // Part name
      & index CL_CPART;       // Returns layout connection list part index
      );
```

Description

The **cap_findlayconpart** function searches the currently loaded layout net list for the specified part name and returns the corresponding part index. The function returns zero if the part was found or non-zero if the part was not found. The **cap_layconload** function is used to load layout net lists to the SCM system.

See also

Functions **cap_findlayconpartpin**, **cap_findlaycontree**, **cap_getlaytreeidx**, **cap_layconload**.

cap_findlayconpartpin - Get layout connection list pin index (CAP)

Synopsis

```
int cap_findlayconpartpin(    // Returns status
      string;                 // Pin name
      index CL_CPART;         // Net list part index
      & index CL_CPIN;        // Returns net list part pin index
      );
```

Description

The **cap_findlayconpartpin** function searches the currently loaded layout net list for the specified part pin name and returns the corresponding pin index. The function returns zero if the part pin was found or nonzero otherwise. The **cap_layconload**. function is used to load layout net lists to the SCM system.

See also

Functions **cap_findlayconpart**, **cap_findlaycontree**, **cap_getlaytreeidx**, **cap_layconload**.

cap_findlaycontree - Get layout connection list net name net index (CAP)

Synopsis

```
int cap_findlaycontree(       // Returns status
      string;                 // Net name
      & index CL_CNET;        // Returns layout connection list net index
      );
```

Description

The **cap_findlaycontree** function searches the currently loaded layout net list for the specified net name and returns the corresponding net index. The function returns zero if the tree was found or non-zero if the tree was not found. The **cap_layconload**. function is used to load layout net lists to the SCM system.

See also

Functions **cap_findlayconpartpin**, **cap_findlayconpart**, **cap_getlaytreeidx**, **cap_layconload**.

cap_getglobnetref - Get global net name reference (CAP)

Synopsis

```
string cap_getglobnetref(     // Returns net name reference
                              //  (or empty string if not referenced)
      string;                 // Net/tree name
      );
```

Description

The **cap_getglobnetref** function retrieves the global net name reference for the soecified net.

cap_getlaytreeidx - Get layout connection list net number net index (CAP)

Synopsis

```
int cap_getlaytreeidx(        // Returns status
      int;                    // Tree number
      & index CL_CNET;        // Returns layout connection list net index
      );
```

Description

The **cap_getlaytreeidx** function searches the currently loaded layout net list for the specified tree number and returns the corresponding net index. The function returns zero if the tree number was found or non-zero if the tree number was not found. The **cap_layconload**. function is used to load layout net lists to the SCM system.

See also

Functions **cap_findlayconpartpin**, **cap_findlayconpart**, **cap_findlaycontree**, **cap_layconload**.

cap_getpartattrib - Get SCM part attribute value (CAP)

Synopsis

```
int cap_getpartattrib(        // Returns status
      string;                 // Part name
      string;                 // Attribute name
      & string;               // Attribute value
      );
```

Description

The **cap_getpartattrib** function retrieves an attribute value of a name-specified part of the currently loaded SCM sheet and returns it with the attribute value parameter. The function returns zero if the required attribute value has been successfully retrieved, (-1) if no SCM sheet is loaded, (-2) on missing and/or invalid parameters, (-3) if the specified part is not placed on the current sheet or (-4) if no attribute with the specified name is defined on the part or has not been set.

cap_getrulecnt - Get rule count for specific object (CAP)

Synopsis

```
int cap_getrulecnt(           // Returns rule count or (-1) on error
      int;                    // Object class code
      int;                    // Object ident code (int or index type)
      );
```

Description

The **cap_getrulecnt** function is used for determining the number of rules attached to a specific object. The object can be the currently loaded element (object class code 0 with **int** value 0 passed for the object ident code), a figure list element of the currently loaded element (object class code 1 with valid **C_FIGURE** index type value passed for the object ident code), or a pool list element (object class code 2 with valid **C_POOL** index type value passed for the object ident code). The function returns a (non-negative) rule count or (-1) on error. The rule count determines the valid range for rule list indices to be passed to the **cap_getrulename** function for getting object-specific rule names. The **cap_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **cap_getrulecnt** function.

See also

Functions **cap_getrulename**, **cap_ruleerr**, **cap_ruleconatt**, **cap_rulecondet**, **cap_rulefigatt**, **cap_rulefigdet**, **cap_ruleplanatt**, **cap_ruleplandet**, **cap_rulequery**; **Neural Rule System** and Rule System Compiler.

cap_getrulename - Get rule name from specific object (CAP)

Synopsis

```
int cap_getrulename(          // Returns nonzero on error
      int;                    // Object class code
      int;                    // Object ident code (int or index type)
      int [0,[;               // Rule name list index
      & string;               // Rule name result
      );
```

Description

The **cap_getrulename** function is used to get the name of an index-specified rule assigned to the specified object. The object can be the currently loaded element (object class code 0 with **int** value 0 passed for the object ident code), a figure list element of the currently loaded element (object class code 1 with valid **C_FIGURE** index type value passed for the object ident code), or a pool list element (object class code 2 with valid **C_POOL** index type value passed for the object ident code). The rule name list index to be specified can be determined using the **cap_getrulecnt** function. The rule name is returned with the last function parameter. The function return value is zero on success or nonzero on error. The **cap_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **cap_getrulename** function.

See also

Functions **cap_getrulecnt**, **cap_ruleerr**, **cap_ruleconatt**, **cap_rulecondet**, **cap_rulefigatt**, **cap_rulefigdet**, **cap_ruleplanatt**, **cap_ruleplandet**, **cap_rulequery**; **Neural Rule System** and Rule System Compiler.

cap_getscbustapidx - Get currently scanned SCM bus tap (CAP)

Synopsis

```
index C_BUSTAP cap_getscbustapidx( // Bus tap index or (-1) if no bus tap scanned
      );
```

Description

The **cap_getscrefpidx** function returns the currently scanned bus tap index. **cap_getscrefpidx** is intended for use in the callback functions of **cap_scanall**, **cap_scanfelem** or **cap_scanpool** only. The function returns (-1) if no scan function is active or if no bus tap is currently scanned.

See also

Functions **cap_scanall**, **cap_scanfelem**, **cap_scanpool**.

cap_getscclass - Get currently scanned SCM class (CAP)

Synopsis

```
int cap getscclass(           // Returns SCM element class:
                              //    0 = Schematic
                              //    1 = Symbol
                              //    2 = Marker
                              //    3 = Label
                              //    (-1) otherwise
      );
```

Description

The **cap_getscclass** function returns the currently scanned SCM element class. **cap_getscclass** is intended for use in the callback functions of **cap_scanall**, **cap_scanfelem** or **cap_scanpool** only. The function returns (-1) if no scan function is active or if no SCM element is currently scanned.

See also

Functions **cap_scanall**, **cap_scanfelem**, **cap_scanpool**.

cap_getscrefpidx - Get currently scanned SCM library element (CAP)

Synopsis

```
index C_POOL cap_getscrefpidx // Returns pool index or (-1) if outside macro
      );
```

Description

The **cap_getscrefpidx** function returns the currently scanned macro reference pool index. This allows for designating the SCM library element to which the currently scanned polygon, text, etc. belongs to. **cap_getscrefpidx** is intended for use in the callback functions of **cap_scanall**, **cap_scanfelem** or **cap_scanpool** only. The function returns (-1) if no scan function is active or if no macro is currently scanned.

See also

Functions **cap_scanall**, **cap_scanfelem**, **cap_scanpool**.

cap_getscstkcnt - Get SCM scan function stack depth (CAP)

Synopsis

```
int cap_getscstkcnt(          // Returns scan stack depth
      );
```

Description

The **cap_getscstkcnt** function returns the current SCM scan function stack depth. I.e., **cap_getscstkcnt** can be used for control purposes in the callback functions of **cap_scanall**, **cap_scanfelem** or **cap_scanpool**.

See also

Functions **cap_scanall**, **cap_scanfelem**, **cap_scanpool**.

cap_gettagdata - Get SCM tag symbol destination data (CAP)

Synopsis

```
int cap_gettagdata(           // Returns status
      index C_FIGURE;         // Tag figure list element
      int [0, [;              // Tag index
      & int;                  // Tag pin type (CAP6)
      & string;               // Tag pin name
      & string;               // Tag reference name 1
      & string;               // Tag reference name 2
      );
```

Description

The **cap_gettagdata** can be used to retrieve the destination data (pin type, pin name reference names) for the specified SCM tag symbol. The function return value is zero if the query was successful or nonzero otherwise.

See also

Function **scm_settagdata**.

cap_lastconseg - Get last modified SCM connection segment (CAP)

Synopsis

```
int cap_lastfigelem(          // Returns status
      & index C_CONSEG;       // Returns connection segment index
      );
```

Description

The **cap_lastconseg** function gets the last created and/or modified SCM connection segment and returns the corresponding connection segment index with the return parameter. The function returns zero if such a connection segment exists or nonzero else.

See also

Function **cap_lastfigelem**.

cap_lastfigelem - Get last modified SCM figure list element (CAP)

Synopsis

```
int cap_lastfigelem(          // Returns status
      & index C_FIGURE;       // Returns figure list index
      );
```

Description

The **cap_lastfigelem** function gets the last created and/or modified SCM figure list element and returns the corresponding figure list index with the return parameter. The function returns zero if such an element exists or nonzero else.

See also

Function **cap_lastconseg**.

cap_layconload - Load layout net list (CAP)

Synopsis

```
int cap_layconload(           // Returns status
      string;                 // DDB file name ("?" for name query)
      string;                 // Layout net list name ("?" for name query)
      );
```

Description

The **cap_layconload** function loads the layout net list with the given name from the specified DDB file name. The function returns zero if the layout net list was successfully loaded, (-1) on file access error or (-2) on missing and/or invalid parameters.

See also

Functions **cap_findlayconpartpin**, **cap_findlayconpart**, **cap_findlaycontree**, **cap_getlaytreeidx**.

cap_maccoords - Get SCM (scanned) macro coordinates (CAP)

Synopsis

```
void cap_maccoords(
      & double;               // Macro X coordinate (STD2)
      & double;               // Macro Y coordinate (STD2)
      & double;               // Macro rotation angle (STD3)
      & int;                  // Macro mirror mode (STD14)
      );
```

Description

The **cap_maccoords** function returns with its parameters the placement data of the currently scanned macro. This function is intended for use in the macro callback function of **cap_scanall**, **cap_scanfelem** or **cap_scanpool** only (otherwise zero/default values are returned).

See also

Functions **cap_scanall**, **cap_scanfelem**, **cap_scanpool**.

cap_macload - Load SCM macro element to memory (CAP)

Synopsis

```
int cap_macload(              // Returns status
      & index C_POOL;         // Macro pool element index
      string;                 // DDB file name
      string;                 // Element name
      int [100,[;             // Element DDB class (STD1)
      );
```

Description

The **cap_macload** function loads the specified SCM library symbol to memory and returns the macro pool element index with the corresponding parameter. The function returns zero if the element was successfully loaded, (-1) on file access errors, (-2) on missing and/or invalid parameters or 1 if referenced macros (library elements) are missing. **cap_macload** is intended to be applied by features such as SCM symbol browsers for examining library file contents. The **cap_macrelease** function can be used to unload and/or release macro elements from memory.

See also

Function **cap_macrelease**.

cap_macrelease - Unload/release SCM macro element from memory (CAP)

Synopsis

```
void cap_macrelease(          // Returns status
      index C_POOL;           // Macro pool element index
      );
```

Description

The **cap_macrelease** function unloads and/or releases the SCM library symbol specified with the macro pool index parameter from memory. **cap_macrelease** is intended to be used together with the **cap_macload** function.

See also

Function **cap_macload**.

cap_mactaglink - Get SCM (scanned) macro tag link data (CAP)

Synopsis

```
int cap_mactaglink(           // Returns status
      & int;                  // Tag pin type (CAP6)
      & double;               // Start point X coordinate (STD2)
      & double;               // Start point Y coordinate (STD2)
      & double;               // End point X coordinate (STD2)
      & double;               // End point Y coordinate (STD2)
      );
```

Description

The **cap_mactaglink** retrieves the tag pin type and the start and end point coordinates of the tag link defined with the currently scanned macro element. This function is intended for use in the macro callback function of **cap_scanall**, **cap_scanfelem** or **cap_scanpool** only. The function returns 1 if a tag link has been found, 0 if no tag link is defined on the currently scanned macro element or (-1) on invalid and/or missing parameter specifications.

See also

Functions **cap_scanall**, **cap_scanfelem**, **cap_scanpool**.

cap_nrefsearch - Search named SCM reference (CAP)

Synopsis

```
int cap_nrefsearch(           // Returns status
      string;                 // Reference name or empty string for newest named
reference
      & index C_FIGURE;       // Returns figure list index
      );
```

Description

The **cap_nrefsearch** function searches for the specified named reference on the currently loaded SCM element. The figure list index is set accordingly, if the named reference is found. The function returns zero if the named reference has been found or nonzero otherwise.

cap_partplan - Get SCM part sheet name (CAP)

Synopsis

```
string cap_partplan(          // Part sheet name
      string;                 // DDB file name
      string;                 // Part name
      );
```

Description

The **cap_partplan** function returns the name of the SCM sheet where the name-specified part is placed in the given DDB file. An empty string is returned if the part has not been found.

cap_pointpoolidx - Get SCM junction point pool element (CAP)

Synopsis

```
index C_POOL cap_pointpoolidx(// Returns pool element
      );
```

Description

The **cap_pointpoolidx** function returns the pool element index, which references the library data of the junction point marker currently in use for connecting segments on the active SCM sheet. This function is useful for plotting SCM plans; the complete junction point marker data can be scanned with the **cap_scanpool** function.

cap_ruleconatt - Attach rule(s) to SCM connection segment (CAP)

Synopsis

```
int cap_ruleconatt(           // Returns nonzero on error
      index C_CONSEG;         // Connection segment element index
      void;                   // Rule name string or rule name list array
      );
```

Description

The **cap_ruleconatt** function is used to attach a *new* set of name-specified rules to the SCM connection segment element specified with the first function parameter. Either a single rule name (i.e., a value of type **string**) or a set of rule names (i.e., an array of type **string**) can be specified with the second function parameter. Note that any rules previously attached to the figure list element are detached before attaching the new rule set. The function returns zero on success or nonzero on error. The **cap_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **cap_ruleconatt** function.

See also

Functions **cap_getrulecnt**, **cap_getrulename**, **cap_ruleerr**, **cap_rulecondet**, **cap_rulefigatt**, **cap_rulefigdet**, **cap_ruleplanatt**, **cap_ruleplandet**, **cap_rulequery**; **Neural Rule System** and Rule System Compiler.

cap_rulecondet - Detach rules from SCM connection segment (CAP)

Synopsis

```
int cap_rulecondet(           // Returns nonzero on error
      index C_CONSEG;         // Connection segment element index
      );
```

Description

The **cap_rulecondet** function is used to detach *all* currently attached rules from the SCM connection segment element specified with the function parameter. The function returns zero on success or nonzero on error. The **cap_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **cap_rulecondet** function.

See also

Functions **cap_getrulecnt**, **cap_getrulename**, **cap_ruleerr**, **cap_ruleconatt**, **cap_rulefigatt**, **cap_rulefigdet**, **cap_ruleplanatt**. **cap_ruleplandet**, **cap_rulequery**; **Neural Rule System** and Rule System Compiler.

cap_ruleerr - Rule System error status query (CAP)

Synopsis

```
void cap_ruleerr(
      & int;              // Error item code
      & string;           // Error item string
      );
```

Description

The **cap_ruleerr** function provides information on the current Rule System error state, and thus can be used to determine the error reason after an unsuccessful call to one of the **Rule System** management functions.

Diagnosis

The Rule System error state can be determined by evaluating the parameters returned with the **cap_ruleerr** function. The returned error item string identifies the error-causing element if needed. The possible error code values correspond with Rule System error conditions according to the following table:

Error Code	Meaning
0	Rule System operation completed without errors
1	Rule System out of memory
2	Rule System internal error **<e>**
3	Rule System function parameter invalid
128	Rule System DB file create error
129	Rule System DB file read/write error
130	Rule System DB file wrong type
131	Rule System DB file structure bad
132	Rule System DB file not found
133	Rule System DB file other error (internal error)
134	Rule System rule **<r>** not found in rule database
135	Rule System rule bad DB format (internal error **<e>**)
136	Rule System object not found
137	Rule System object double defined (internal error)
138	Rule System incompatible variable **<v>** definition
139	Rule System Rule **<r>** compiled with incompatible RULECOMP version

Depending on the error condition the error item string can describe a rule **<r>**, a variable **<v>** or an (internal) error status **<e>**. DB file errors refer to problems accessing the Rule System database file **brules.vdb** in the BAE programs directory. Internal errors usually refer to Rule System implementation gaps and should be reported to Bartels.

See also

Functions **cap_getrulecnt**, **cap_getrulename**, **cap_ruleconatt**, **cap_rulecondet**, **cap_rulefigatt**, **cap_rulefigdet**, **cap_ruleplanatt**, **cap_ruleplandet**, **cap_rulequery**; **Neural Rule System** and Rule System Compiler.

cap_rulefigatt - Attach rule(s) to figure list element (CAP)

Synopsis

```
int cap_rulefigatt(           // Returns nonzero on error
      index C_FIGURE;         // Figure list element index
      void;                   // Rule name string or rule name list array
      );
```

Description

The **cap_rulefigatt** function is used to attach a *new* set of name-specified rules to the figure list element specified with the first function parameter. Either a single rule name (i.e., a value of type **string**) or a set of rule names (i.e., an array of type **string**) can be specified with the second function parameter. Note that any rules previously attached to the figure list element is detached before attaching the new rule set. The function returns zero on success or nonzero on error. The **cap_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **cap_rulefigatt** function.

See also

Functions **cap_getrulecnt**, **cap_getrulename**, **cap_ruleerr**, **cap_ruleconatt**, **cap_rulecondet**, **cap_rulefigdet**, **cap_ruleplanatt**, **cap_ruleplandet**, **cap_rulequery**; **Neural Rule System** and Rule System Compiler.

cap_rulefigdet - Detach rules from figure list element (CAP)

Synopsis

```
int cap_rulefigdet(           // Returns nonzero on error
      index C_FIGURE;         // Figure list element index
      );
```

Description

The **cap_rulefigdet** function is used to detach *all* currently attached rules from the figure list element specified with the function parameter. The function returns zero on success or nonzero on error. The **cap_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **cap_rulefigdet** function.

See also

Functions **cap_getrulecnt**, **cap_getrulename**, **cap_ruleerr**, **cap_ruleconatt**, **cap_rulecondet**, **cap_rulefigatt**, **cap_ruleplanatt**. **cap_ruleplandet**, **cap_rulequery**; **Neural Rule System** and Rule System Compiler.

cap_ruleplanatt - Attach rule(s) to currently loaded element (CAP)

Synopsis

```
int cap_ruleplanatt(          // Returns nonzero on error
      void;                   // Rule name string or rule name list array
      int [0,1];              // Flag - SCM global rule
      );
```

Description

The **cap_ruleplanatt** function is used to attach a *new* set of name-specified rules to the currently loaded element. Either a single rule name (i.e., a value of type **string**) or a set of rule names (i.e., an array of type **string**) can be specified with the function parameter. Note that any rules previously attached to the current element is detached before attaching the new rule set. The SCM global rule parameter allows for the attachment of rules to *all* SCM sheets of the currently SCM sheet element (i.e., this parameter is only evaluated if an SCM sheet element is loaded). The function returns zero on success or nonzero on error. The **cap_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **cap_ruleplanatt** function.

See also

Functions **cap_getrulecnt**, **cap_getrulename**, **cap_ruleerr**, **cap_ruleconatt**, **cap_rulecondet**, **cap_rulefigatt**, **cap_rulefigdet**, **cap_ruleplandet**, **cap_rulequery**; **Neural Rule System** and Rule System Compiler.

cap_ruleplandet - Detach rules from currently loaded element (CAP)

Synopsis

```
int cap_ruleplandet(          // Returns nonzero on error
      int [0,1];              // Flag - SCM global rule
      );
```

Description

The **cap_ruleplandet** function to detach *all* currently attached rules from the currently loaded element. The SCM global rule parameter allows for the detachment of rules from *all* SCM sheets of the currently SCM sheet element (i.e., this parameter is only evaluated if an SCM sheet element is loaded). The function returns zero on success or nonzero on error. The **cap_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **cap_ruleplandet** function.

See also

Functions **cap_getrulecnt**, **cap_getrulename**, **cap_ruleerr**, **cap_ruleconatt**, **cap_rulecondet**, **cap_rulefigatt**, **cap_rulefigdet**, **cap_ruleplanatt**, **cap_rulequery**; **Neural Rule System** and Rule System Compiler.

cap_rulequery - Perform rule query on specific object (CAP)

Synopsis

```
int cap_rulequery(            // Returns hit count or (-1) on error
      int;                    // Object class code
      int;                    // Object ident code (int or index type)
      string;                 // Subject name
      string;                 // Predicate name
      string;                 // Query command string
      & void;                 // Query result
      []                      // Optional query parameters of requested type
      );
```

Description

The **cap_rulequery** function is used to perform a rule query on a specific object. The object can be the currently loaded element (object class code 0 with `int` value 0 passed for the object ident code), a figure list element of the currently loaded element (object class code 1 with valid **C_FIGURE** index type value passed for the object ident code) or a pool list element (object class code 2 with valid **C_POOL** index type value passed for the object ident code). The rule query function requires a rule subject, a rule predicate and a query command string to be specified with the corresponding function parameters. The query command string can contain one query operator and a series of value definition operators. The following query operators are implemented:

`?d`	for querying `int` values
`?f`	for querying `double` values
`?s`	for querying `string` values

The query operator can optionally be preceded with one of the following selection operators:

`+`	for selecting the maximum of all matching values
`-`	for selecting the minimum of all matching values

The `+` operator is used on default (e.g., when omitting the selection operator). The rule query resulting value is passed back to the caller with the query result parameter. This means that the query result parameter data type must comply with the query operator (`int` for `?d`, `double` for `?f`, `string` for `?s`). The query command string can also contain a series of value definition operators such as:

`%d`	for specifying `int` values
`%f`	for specifying `double` values
`%s`	for specifying `string` values

Each value definition parameter is considered a placeholder for specific data to be passed with optional parameters. Note that these optional parameters must comply with the query command in terms of specified sequence and data types. The **cap_rulequery** function returns a (non-negative) hit count denoting the number of value set entries matched by the query. The function returns (-1) on error. The **cap_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **cap_rulequery** function.

Examples

With the rule

```
rule somerule
{
    subject subj
    {
        pred := ("A", 2);
        pred := ("A", 4);
        pred := ("B", 1);
        pred := ("C", 3);
        pred := ("B", 6);
        pred := ("D", 5);
        pred := ("D", 6);
        pred := ("A", 3);
    }
}
```

defined and attached to the currently loaded element, the **cap_rulequery** call

```
    hitcount = cap_rulequery(0,0,"subj","pred","%s ?d",intresult,"A") ;
```

sets the **int** variable **hitcount** to 3 and the **int** variable **intresult** to 4. whilst a call such as

```
    hitcount = cap_rulequery(0,0,"subj","pred","-?s %d",strresult,6) ;
```

sets **hitcount** to 2 and **string** variable **strresult** to **B**.

See also

Functions **cap_getrulecnt**, **cap_getrulename**, **cap_ruleerr**, **cap_ruleconatt**, **cap_rulecondet**, **cap_rulefigatt**, **cap_rulefigdet**, **cap_ruleplanatt**, **cap_ruleplandet**; **Neural Rule System** and Rule System Compiler.

cap_scanall - Scan all SCM figure list elements (CAP)

Synopsis

```
int cap_scanall(              // Returns scan status
      double;                 // Scan X offset (STD2)
      double;                 // Scan Y offset (STD2)
      double;                 // Scan rotation angle (STD3)
      int [0,1];              // Element in workspace flag (STD10)
      * int;                  // Macro callback function
      * int;                  // Connection callback function
      * int;                  // Polygon callback function
      * int;                  // Text callback function
      );
```

Description

The **cap_scanall** function scans all figure list elements placed on the currently loaded SCM element with all hierarchy levels. User-defined scan functions automatically activated according to the currently scanned element type. If a certain callback function should not be referenced, then the corresponding parameter must be set to the keyword **NULL**. The function returns nonzero on invalid parameter specifications, or if one of the referenced user functions has returned a scan error status.

Macro callback function

```
int macrofuncname(
      index C_MACRO macro,    // Macro index
      index C_POOL pool,      // Internal pool index
      int macinws,            // Macro in workspace flag (STD10)
      string refname,         // Macro reference name
      index C_LEVEL level,    // Macro signal level
      index C_LEVEL buslevel  // Macro bus signal level
      )
{
      // Macro callback function statements
      :
      return(contscan);
}
```

Connected elements share the same non-negative signal level value. The return value of the macro callback function must be 1 for continue scan, 0 for stop scan or (-1) on error.

Connection callback function

```
int confuncname(
      index C_CONBASE cbase,  // Connection base index
      int segidx,             // Segment index
      int ctyp,               // Connection type:
                              //    0 = normal connection
                              //    1 = junction point
      double lx,              // Lower X coordinate (STD2)
      double ly,              // Lower Y coordinate (STD2)
      double ux,              // Upper X coordinate (STD2)
      double uy,              // Upper Y coordinate (STD2)
      int busflag,            // Bus connection flag:
                              //    0 = normal connection
                              //    1 = bus connection
      int cinws,              // Connection in workspace flag (STD10)
      index C_LEVEL level     // Connection signal level
      )
{
      // Connection callback function statements
      :
      return(errstat);
}
```

The connection type value 0 denotes a normal (or bus) connection. Connection type value 1 denotes a connection junction point with identical upper and lower coordinates. Connected elements share the same non-negative signal level value. The return value of the connection callback function must be zero for scan ok or nonzero on error.

Polygon callback function

```
int polyfuncname(
      index C_POLY poly,      // Polygon index
      int polyinws,           // Polygon in workspace flag (STD10)
      index C_LEVEL level,    // Polygon level
      int macclass,           // Polygon macro class (STD1)
      int bustapidx           // Polygon bustap index
      )
{
      // Polygon callback function statements
      :
      return(errstat);
}
```

The macro class refers to the macro where the polygon is placed onto. The bustap index is non-negative if the polygon is placed on a bustap. Connected elements share the same non-negative signal level value. The return value of the polygon callback function must be zero for scan ok or nonzero on error.

Text callback function

```
int textfuncname(
      index C_TEXT text,      // Text index
      double x,               // Text X coordinate (STD2)
      double y,               // Text Y coordinate (STD2)
      double angle,           // Text rotation angle (STD3)
      int mirr,               // Text mirror mode (STD14)
      double size,            // Text size (STD2)
      string textstr,         // Text string
      int textinws,           // Text in workspace flag (STD10)
      int macclass,           // Text macro class
      int variant             // Text variant attribute flag
      )
{
      // Text callback function statements
      :
      return(errstat);
}
```

The macro class refers to the macro where the text is placed onto. The return value of the text callback function must be zero for scan ok or nonzero on error.

See also
Functions **cap_maccoords**, **cap_scanfelem**, **cap_scanpool**.

cap_scanfelem - Scan SCM figure list element (CAP)

Synopsis

```
int cap_scanfelem(            // Returns scan Status
      index C_FIGURE;         // Figure list element index
      double;                 // Scan X offset (STD2)
      double;                 // Scan Y offset (STD2)
      double;                 // Scan rotation angle (STD3)
      int [0,1];              // Element in workspace flag (STD10)
      * int;                  // Macro callback function
      * int;                  // Connection callback function
      * int;                  // Polygon callback function
      * int;                  // Text callback function
      );
```

Description

The **cap_scanfelem** function scans the specified SCM figure list element with all hierarchy levels. User-defined callback functions for the currently scanned element type are automatically activated. If a certain callback function should not be referenced, then the corresponding parameter must be set to the keyword **NULL**. The return value of **cap_scanfelem** is nonzero on invalid parameter specifications or if one of the referenced user functions has returned a scan error status. See **cap_scanall** for the callback function definitions.

See also

Functions **cap_maccoords**, **cap_scanall**, **cap_scanpool**.

cap_scanpool - Scan SCM pool element (CAP)

Synopsis

```
int cap_scanpool(             // Returns scan Status
      void;                   // Pool element index
      double;                 // Scan X offset (STD2)
      double;                 // Scan Y offset (STD2)
      double;                 // Scan rotation angle (STD3)
      int [0,1];              // Element in workspace flag (STD10)
      * int;                  // Macro callback function
      * int;                  // Connection callback function
      * int;                  // Polygon callback function
      * int;                  // Text callback function
      );
```

Description

The **cap_scanpool** function scans the specified SCM pool element with all hierarchy levels. User-defined callback functions for the currently scanned element type are automatically activated. If a certain callback function should not be referenced, then the corresponding parameter must be set to the keyword **NULL**. The return value of **cap_scanpool** is nonzero on invalid parameter specifications or if one of the referenced user functions has returned a scan error status. See **cap_scanall** for the callback function definitions.

See also

Functions **cap_maccoords**, **cap_scanall**, **cap_scanfelem**.

cap_vecttext - Vectorize SCM text (CAP)

Synopsis

```
int cap_vecttext(             // Returns status
      double;                 // Text X coordinate (STD2)
      double;                 // Text Y coordinate (STD2)
      double;                 // Text rotation angle (STD3)
      int [0,1];              // Text mirror mode (STD14)
      double ]0.0,[;          // Text size (STD2)
      int [0,[;               // Text style (CAP7)
      string;                 // Text string
      * int;                  // Text vectorize function
      );
```

Description

The **cap_vecttext** function vectorizes the specified text using the currently loaded text font. The text vectorize user function is automatically called for each text segment. The function returns nonzero if invalid parameters have been specified or if the referenced user function returns nonzero.

Text vectorize function

```
int vecfuncname(
      double x1,              // Start point X coordinate (STD2)
      double y1,              // Start point Y coordinate (STD2)
      double x2,              // End point X coordinate (STD2)
      double y2               // End point Y coordinate (STD2)
      )
{
      // Text vectorize function statements
      :
      return(errstat);
}
```

The return value of the text vectorize function must be zero if scan ok or nonzero on error.

C.3.2 Schematic Editor Functions

The following **User Language** system functions are assigned to caller type SCM; i.e., they can be called from the **Schematic Editor** interpreter environment of the **Bartels AutoEngineer**:

scm_askrefname - SCM reference name selection (SCM)

Synopsis

```
int scm_askrefname(           // Returns status
      & string;               // Returns reference name
      );
```

Description

The **scm_askrefname** function activates a dialog for selecting a named reference, i.e., a symbol on SCM plan level or a pin on SCM symbol level. The functions returns zero if a named reference was successfully selected or (-1) on error.

scm_asktreename - SCM net name selection (SCM)

Synopsis

```
int scm_asktreename(          // Returns status
      & string;               // Returns tree/net name
      );
```

Description

The **scm_asktreename** function activates a dialog for selecting a net. The functions returns zero if a net was successfully selected or (-1) on error.

scm_attachtextpos - Attach text position to SCM element (SCM)

Synopsis

```
int scm_attachtextpos(        // Returns status
      index C_FIGURE;         // SCM figure list element
      string;                 // Text string
      double;                 // Text X coordinate (STD2)
      double;                 // Text Y coordinate (STD2)
      double;                 // Text rotation angle (STD3)
      double ]0.0,[;          // Text size (STD2)
      int [-1,1];             // Text mirror mode (STD14) or (-1) for reset
      );
```

Description

The **scm_attachtextpos** funtion assigns a text position modifier with the specified properties for position, rotation, size and mirroring to the text string of the specified SCM figure list element. The function returns zero if the assignment was successful, (-1) for invalid parameters or (-2) if the SCM element provides no text position modifier for the specified text string.

See also

Function **scm_storetext**.

scm_checkbustapplot - Get SCM bus tap plot status (SCM)

Synopsis

```
int scm_checkbustapplot(      // Plot status
      index C_FIGURE;         // Bus connection figure list index
      index C_BUSTAP;         // Bus tap index
      );
```

Description

The **scm_checkbustapplot** function returns 1 if plot output is disabled for the specified bus tap, or zero otherwise.

scm_checkjunctplot - Get SCM junction point plot status (SCM)

Synopsis

```
int scm_checkjunctplot(       // Plot status
      double;                 // Junction point X position (STD2)
      double;                 // Junction point Y position (STD2)
      );
```

Description

The **scm_checkjunctplot** check whether connection junction point markers at the given coordinates are to be plotted. The function returns 1 if plot output is disabled or 0 if plot output is enabled.

scm_chkattrname - SCM attribute name validation (SCM)

Synopsis

```
int scm_chkattrname(          // Returns non-zero if invalid attribute name
      string;                 // Attribute name
      );
```

Description

The **scm_chkattrname** function checks if the specified attribute name is a valid attribute name which allows for attribute value assignment. The functions returns zero if the attribute name is valid or non-zero otherwise.

See also

Function **scm_setpartattrib**.

scm_conseggrpchg - Change SCM connection segment group flag (SCM)

Synopsis

```
int scm_conseggrpchg(         // Returns status
      index C_CONSEG;         // Connection segment
      int [0,6];              // New group selection status (STD13)
                              //    |4 - Display group status message
      );
```

Description

The **scm_conseggrpchg** function changes the group flag of the given connection segment. The function returns zero if the connection segment group flag has been successfully changed or (-1) if the given element is invalid.

scm_deflibname - SCM setup default library name (SCM)

Synopsis

```
string scm_deflibname(        // Returns default library name
      );
```

Description

The **scm_deflibname** function returns the default SCM library name defined in the BAE setup file.

scm_defloglname - SCM setup default packager library name (SCM)

Synopsis

```
string scm_defloglname(       // Returns default packager library name
      );
```

Description

The **scm_defloglname** function returns the default packager layout library name defined in the BAE setup file.

See also

Function **con_getlogpart**; utility program **BSETUP**.

scm_defsegbus - SCM connection segment bus definition (SCM)

Synopsis

```
int scm_defsegbus(            // Returns status
      & index C_CONSEG;       // Connection segment
      );
```

Description

The **scm_defsegbus** performs a bus definition on the given connection segment and all adjacent segments. The function returns nonzero on bus definition errors.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **C_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

scm_delconseg - Delete SCM connection segment (SCM)

Synopsis

```
int scm_delconseg(            // Returns status
      & index C_CONSEG;       // Connection segment
      );
```

Description

The **scm_delconseg** function deletes the given connection segment from the figure list. The function returns zero if the connection segment was successfully deleted or nonzero on error.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **C_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

scm_delelem - Delete SCM figure list element (SCM)

Synopsis

```
int scm_delelem(              // Returns status
      & index C_FIGURE;       // Element
      );
```

Description

The **scm_delelem** function deletes the given figure list element from the figure list. The function returns zero if the element was successfully deleted or nonzero on error.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **C_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

See also
Function **scm_drawelem**.

scm_drawelem - Redraw SCM figure list element (SCM)

Synopsis

```
void scm_drawelem(
      index C_FIGURE;          // Element
      int [0, 4];              // Drawing mode (STD19)
      );
```

Description

The **scm_drawelem** function updates the display of the given figure list element using the specified drawing mode.

See also

Function **scm_delelem**.

scm_elemangchg - Change SCM figure list element rotation angle (SCM)

Synopsis

```
int scm_elemangchg(            // Returns status
      & index C_FIGURE;        // Element
      double;                  // New rotation angle (STD3)
      );
```

Description

The **scm_elemangchg** function changes the rotation angle of the given figure list element. The rotation angle must be in radians. The function returns zero if the element has been successfully rotated, (-1) if the figure list element is invalid or (-2) if the figure list element cannot be rotated.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **C_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

scm_elemgrpchg - Change SCM figure list element group flag (SCM)

Synopsis

```
int scm_elemgrpchg(            // Returns status
      index C_FIGURE;          // Element
      int [0,6];               // New group selection status (STD13)
                               //    |4 - Display group status message
      );
```

Description

The **scm_elemgrpchg** function changes the group flag of the given figure list element. A group flag value of 0 deselects the element, a group flag value of 1 selects the element. The function returns zero if the element group flag has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element cannot be selected to a group.

scm_elemmirrchg - Change SCM figure list element mirror mode (SCM)

Synopsis

```
int scm_elemmirrchg(          // Returns status
      & index C_FIGURE;       // Element
      int [0,1];              // New mirror mode (STD14)
      );
```

Description

The **scm_elemmirrchg** function changes the mirror mode of the given figure list element. The mirror mode can be set for texts and references. The function returns zero if the element mirror mode has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element cannot be mirrored.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **C_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

scm_elemposchg - Change SCM figure list element position (SCM)

Synopsis

```
int scm_elemposchg(           // Returns status
      & index C_FIGURE;       // Element
      double;                 // New X coordinate (STD2)
      double;                 // New Y coordinate (STD2)
      );
```

Description

The **scm_elemposchg** function changes the position of the given figure list element. The function returns zero if the element has been successfully repositioned, (-1) if the figure list element is invalid or (-2) if the figure list element position cannot be set.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **C_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

scm_elemsizechg - Change SCM figure list element size (SCM)

Synopsis

```
int scm_elemsizechg(          // Returns status
      & index C_FIGURE;       // Element
      double;                 // New size (STD2)
      );
```

Description

The **scm_elemsizechg** function changes the size of the given figure list element. The size can be changed for texts only. The function returns zero if the element size has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element size cannot be set.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **C_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

scm_findpartplc - Layout part placement status query (BAE HighEnd) (SCM)

Synopsis

```
int scm_findpartplc(          // Placement status
      string;                 // Part name
      );
```

Description

The **scm_findprtplc** function can be used in the **BAE HighEnd Schematic Editor** to query the placement status of layout parts. The function returns 1 if a part with the specified part name is known to be placed on the project's layout. Otherwise the return value is zero.

scm_getdblpar - Get SCM double parameter (SCM)

Synopsis

```
int scm_getdblpar(            // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = Plot scale factor
                              //    1 = Plotter HPGL speed
                              //    2 = Plotter pen width (STD2)
                              //    3 = Last group placement x coordinate (STD2)
                              //    4 = Last group placement y coordinate (STD2)
                              //    5 = Default symbol placement angle (STD3)
                              //    6 = Default text size (STD2)
                              //    7 = Default text placement angle (STD3)
      & double;               // Returns parameter value
      );
```

Description

The **scm_getdblpar** function is used to query **Schematic Editor** double parameters previously set with **scm_setdblpar**. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **scm_getintpar**, **scm_getstrpar**, **scm_setdblpar**, **scm_setintpar**, **scm_setstrpar**.

scm_getgroupdata - SCM group placement data query (SCM)

Synopsis

```
int scm_getgroupdata(         // Status
      & double;               // Group base X coordinate (STD2)
      & double;               // Group base Y coordinate (STD2)
      & double;               // Group rotation angle (STD3)
      & double;               // Group scale factor
      & int;                  // Group mirror mode
      & double;               // Group quadrant X coordinate (STD2)
      & double;               // Group quadrant Y coordinate (STD2)
      & int;                  // Group quadrant mode
      & int;                  // Group area mode
      );
```

Description

scm_getgroupdata function can be used to retrieve the current **Schematic Editor** group placement interaction input data. The function returns nonzero if no group placement interaction is activated.

See also
Function **scm_getinputdata**.

scm_gethighlnet - Get SCM net highlight mode (SCM)

Synopsis

```
int scm_gethighlnet(          // Returns status
      int [-1,[;              // Net tree number or -1 for highlight focus modus
query
      & int;                  // Highlight mode
      );
```

Description

The **scm_gethighlnet** function can be used to get the highlight mode for the specified net. The highlight mode parameter is set to nonzero if the net highlight is activated, or zero if the net highlight is deactivated. The function returns nonzero if the query was successful, or zero on error (net not found, invalid parameters).

See also

Function **scm_highlnet**.

scm_gethpglparam - SCM HP-GL plot parameter query (SCM)

Synopsis

```
void scm_gethpglparam(
      & string;               // HP-GL plot file name
      & double;               // HP-GL plot scaling factor
      & double;               // HP-GL plotter speed (-1.0=full speed)
      & double;               // HP-GL plotter pen width (STD2)
      & int;                  // HP-GL plot area fill mode:
                              //    0 = fill off
                              //    1 = fill on
      & int;                  // HP-GL plot rotation mode:
                              //       0 = no rotation
                              //       1 = 90 degree rotation
                              //    else = automatic rotation
      );
```

Description

The **scm_gethpglparam** function returns the **Schematic Editor** HP-GL plot parameters.

scm_getinputdata - SCM input data query (SCM)

Synopsis

```
int scm_getinputdata(         // Status
      & double;               // Initial X coordinate (STD2)
      & double;               // Initial Y coordinate (STD2)
      & double;               // Initial width (STD2)
      & int;                  // Initial display element type (SCM1)
      & double;               // Current X coordinate (STD2)
      & double;               // Current Y coordinate (STD2)
      & double;               // Current width (STD2)
      & int;                  // Current display element type (SCM1)
      & void;                 // Input mode/element
      & double;               // Input first segment start X coordinate (STD2) */
      & double;               // Input first segment start Y coordinate (STD2) */
      & double;               // Input first arc center X coordinate (STD2) */
      & double;               // Input first arc center Y coordinate (STD2) */
      & int;                  // Input first arc center type (STD15) */
      & double;               // Input last segment start X coordinate (STD2) */
      & double;               // Input last segment start Y coordinate (STD2) */
      & double;               // Input last arc center X coordinate (STD2) */
      & double;               // Input last arc center Y coordinate (STD2) */
      & int;                  // Input last arc center type (STD15) */
      );
```

Description

The **scm_getinputdata** function can be used to retrieve the current **Schematic Editor** placement interaction input data. The placement data has to be interpreted according to the input interaction type and/or placement element function parameter. The function returns nonzero if no placement interaction is activated.

See also
Function **scm_getgroupdata**.

scm_getintpar - Get SCM integer parameter (SCM)

Synopsis

```
int scm_getintpar(              // Returns status
      int [0,[;                 // Parameter type/number:
                                //    0 = Pick point display mode:
                                //        0 = No pick point display
                                //        1 = Pick point display
                                //    1 = Symbol/group reroute mode:
                                //        Bit 0/1: Router mode
                                //          0 = Router off
                                //          1 = Symbol & group route
                                //          2 = Symbol route only
                                //          3 = Group route only
                                //        Bit 2: quick shot route flag
                                //    2 = Symbol name move preservation mode:
                                //        0 = Reset symbol texts on symbol move
                                //        1 = Keep symbol text offsets
                                //    3 = Last placed reference type:
                                //        (-1) = No reference placed yet
                                //        0 = Symbol
                                //        1 = Label
                                //        2 = Module Port
                                //    4 = Warning mode for connecting named nets:
                                //        0 = Warning message display deactivated
                                //        1 = Warning message display activated
                                //    5 = Element pick mode:
                                //        0 = Best pick
                                //        1 = Pick element selection
                                //    6 = Generic printer color mode:
                                //        0 = B/W
                                //        1 = Color
                                //    7 = Warning output mode:
                                //        Bit 0: Supress $ rename warnings
                                //        Bit 1: Supress net join warnings
                                //        Bit 2: Supress module port warnings
                                //        Bit 4: Supress group symbol rename
                                //                warnings
                                //        Bit 5: Supress variant mismatch warnings
                                //    8 = Name prompt mode:
                                //        0 = Autopattern symbol name
                                //        1 = Prompt for symbol name
                                //    9 = Info display flag:
                                //        0 = no automatic info display
                                //        1 = automatic info display
                                //    10 = Info display mode:
                                //        0 = no info display
                                //        1 = complete info display
                                //        2 = net related info only display
                                //    11 = Label rerouting mode:
                                //        0 = no label rerouting
                                //        1 = label rerotuing
                                //    12 = Sub symbol number offset
                                //    13 = Generic plot scale mode:
                                //        0 = fix scale factor
                                //        1 = autosize to page
                                //    14 = Generic color mode:
                                //        0 = black/white
                                //        1 = use current color settings
                                //    15 = HPGL fill mode:
                                //        0 = outline draw
                                //        1 = filled draw
                                //        2 = filled draw, draw wide lines/texts
                                //    16 = Area polygon edit mode:
                                //        0 = don't close polylines
                                //        1 = always close polylines
                                //        2 = polyline close prompt
```

```
                                //    17 = Group connection rerouting mode:
                                //         0 = no outside antenna deletion
                                //         1 = delete 1st outside antenna segment
                                //         2 = delete complete outside antenna
                                //    18 = Default user units code:
                                //         0 = metric
                                //         1 = imperial
                                //    19 = Plot preview mode:
                                //         0 = none
                                //         1 = plotter pen width
                                //    20 = Autosave interval
                                //    21 = Automatic connection corners
                                //    22 = Angle lock toggle flag
                                //    23 = Default symbol mirroring
                                //    24 = Group move display mode:
                                //         0 = Moving Picture On
                                //         1 = Moving Picture All
                                //         2 = Instant Moving Picture
                                //    25 = Clipboard text placement request flag
                                //    26 = Signal router routing range
                                //    27 = Automatic bustap connection
                                //    28 = Signal router marker scan flag
                                //    29 = Segment move mode:
                                //         0 = Move without neighbours
                                //         1 = Move with neighbours
                                //         2 = Adjust neighbours
                                //        |4 = End point follows segment
                                //    30 = Group angle lock mode:
                                //         0 = Keep group angle lock
                                //         1 = Automatically release group angle lock
                                //    31 = Default text mirror mode (STD14)
                                //    32 = Default text mode (CAP1|CAP7)
                                //    33 = Symbol $noplc plot visibility mode:
                                //         Bit 0 = Set $noplc if plot
                                //                 visibility changes
                                //         Bit 1 = Set plot visibility
                                //                 if $noplc changes
                                //    34 = Net plan list maximum length [ 3,200]
                                //    35 = Single corner edit flag
                                //    36 = Unroute line creation flag
                                //    37 = Polygon edit autocomplete flag
                                //    38 = Error highlight mode:
                                //         0 = Error highlight
                                //         1 = Error pattern/dash
                                //    39 = Flag - Automirror horizontal bus taps
                                //    40 = Symbol/label tag mode:
                                //         0 = Standard symbol
                                //         1 = Virtual tag
                                //         2 = Netlist tag
                                //    41 = Connection split mode:
                                //         0 = No connection split
                                //         1 = Only at 2 pin symbols
                                //         2 = Connection split
                                //    42 = Symbol connection split mode:
                                //         0 = No connection split
                                //         1 = Only at 2 pin symbols
                                //         2 = Connection split
                                //    43 = Junction marker group segment count
                                //           set by scm_checkjunctplot
                                //    44 = Current project plan count
                                //    45 = Display class bits connection (SCM2)
                                //    46 = Display class bits bus (SCM2)
                                //    47 = Display class bits text (SCM2)
                                //    48 = Display class bits comment text (SCM2)
                                //    49 = Display class bits graphic area (SCM2)
                                //    50 = Display class bits graphic line (SCM2)
                                //    51 = Display class bits dotted line (SCM2)
                                //    52 = Display class bits connection area (SCM2)
                                //    53 = Display class bits net area (SCM2)
```

```
                              //    54 = Display class bits macro outline (SCM2)
                              //    55 = Display class bits tag (SCM2)
      & int;                  // Returns parameter value
      );
```

Description

The **scm_getintpar** function is used to query **Schematic Editor** integer parameters previously set with **scm_setintpar**. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **scm_getdblpar**, **scm_getstrpar**, **scm_setdblpar**, **scm_setintpar**, **scm_setstrpar**.

scm_getstrpar - Get SCM string parameter (SCM)

Synopsis

```
int scm_getstrpar(            // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = Last placed named reference macro name
                              //    1 = Last placed named reference name
                              //    2 = Last placed net name
                              //    3 = Last placed bus tap name
                              //    4 = Last placed text string
                              //    5 = Symbol name pattern
                              //    6 = Next placed text string
                              //    7 = Last placed macro library
                              //    8 = Error bustap name
                              //    9 = Error bus name
                              //    10 = Next free name
                              //    11 = Current hierachical block reference name
                              //    12 = Last picked attribute name
                              //    13 = Last picked attribute value
                              //    14 = Autosave path name
      & string;               // Returns parameter value
      );
```

Description

The **scm_getstrpar** function is used to query **Schematic Editor** string parameter settings. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **scm_getdblpar**, **scm_getintpar**, **scm_setdblpar**, **scm_setintpar**, **scm_setstrpar**.

scm_highlnet - Set SCM net highlight mode (SCM)

Synopsis

```
int scm_highlnet(             // Returns status
      int [-1,[;              // Net tree number
      int [0,1];              // Highlight mode
      );
```

Description

The **scm_highlnet** function sets the highlight mode of the net specified by the given net tree number. The highlight mode parameter designates whether the net should be highlighted (value 1) or not (value 0). The function returns nonzero if the highlight mode was successfully set, or zero if an invalid net tree number and/or highlight mode value has been specified.

See also

Function **scm_gethighlnet**.

scm_pickanyelem - Pick any SCM figure list element (SCM)

Synopsis

```
int scm_pickanyelem(          // Returns status
      & index C_FIGURE;       // Returns picked element
      & index C_CONSEG;       // Returns picked connection segment
      & index C_BUSTAP;       // Returns picked bus tap
      & int;                  // Returns picked element type:
                              //    0 = figure list element
                              //    1 = connection segment
                              //    2 = bus tap
      int;                    // Pick element type set ((CAP3 except 7)<<1 or'ed)
      );
```

Description

The **scm_pickanyelem** function activates a mouse interaction for selecting a figure list element from the specified pick element type set. The picked element index is returned with either of the first three parameters. The returned parameter for the picked element type can be used to determine which of the picked element index variables is valid. The function returns zero if an element has been picked or nonzero if no element was found at the pick position.

See also

Functions **scm_pickbustap**, **scm_pickconseg**, **scm_pickelem**, **scm_setpickconseg**.

scm_pickbustap - Pick SCM bus tap (SCM)

Synopsis

```
int scm_pickbustap(           // Returns status
      & index C_BUSTAP;       // Returns selected bus tap element
      );
```

Description

The **scm_pickbustap** function activates an interactive bus tap pick request (with mouse). The picked bus tap element is returned with the function parameter. The function returns zero if a bus tap was picked or (-1) if no bus tap was found at the pick position.

See also

Functions **scm_pickanyelem**, **scm_pickconseg**, **scm_pickelem**, **scm_setpickconseg**.

scm_pickconseg - Pick SCM connection segment (SCM)

Synopsis

```
int scm_pickconseg(           // Returns status
      & index C_CONSEG;       // Returns picked connection segment
      );
```

Description

The **scm_pickconseg** function activates an interactive connection segment pick request (with mouse). The picked connection segment index is returned with the function parameter. The function returns zero if a connection segment has been picked or (-1) if no connection segment has been found at the pick position.

See also

Functions **scm_pickanyelem**, **scm_pickbustap**, **scm_pickelem**, **scm_setpickconseg**.

scm_pickelem - Pick SCM figure list element (SCM)

Synopsis

```
int scm_pickelem(             // Returns status
      & index C_FIGURE;       // Returns picked element
      int [1,11];             // Pick element type (CAP3 except 2 and 7)
      );
```

Description

The **scm_pickelem** function activates an interactive figure list element pick request (with mouse). The required pick element type is specified with the second parameter. The picked figure list element index is returned with the first parameter. The function returns zero if an element has been picked or (-1) if no element of the required type has been found at the pick position.

See also

Functions **scm_pickanyelem**, **scm_pickbustap**, **scm_pickconseg**, **scm_setpickconseg**.

scm_setdblpar - Set SCM double parameter (SCM)

Synopsis

```
int scm_setdblpar(            // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = Plot scale factor
                              //    1 = Plotter HPGL speed
                              //    2 = Plotter pen width (STD2)
                              //    3 = Last group placement x coordinate (STD2)
                              //    4 = Last group placement y coordinate (STD2)
                              //    5 = Default symbol placement angle (STD3)
                              //    6 = Default text size (STD2)
                              //    7 = Default text placement angle (STD3)
      double;                 // Parameter value
      );
```

Description

The **scm_setdblpar** function is used to set **Schematic Editor** double system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **scm_getdblpar** function can be used to query parameter values set with **scm_setdblpar**.

See also

Functions **scm_getdblpar**, **scm_getintpar**, **scm_getstrpar**, **scm_setintpar**, **scm_setstrpar**.

scm_setintpar - Set SCM integer parameter (SCM)

Synopsis

```
int scm_setintpar(              // Returns status
      int [0,[;                 // Parameter type/number:
                                //    0 = Pick point display mode:
                                //        0 = No pick point display
                                //        1 = Pick point display
                                //    1 = Symbol/group reroute mode:
                                //        Bit 0/1: Router mode
                                //          0 = Router off
                                //          1 = Symbol & group route
                                //          2 = Symbol route only
                                //          3 = Group route only
                                //        Bit 2: quick shot route flag
                                //    2 = Symbol name move preservation mode:
                                //        0 = Reset symbol texts on symbol move
                                //        1 = Keep symbol text offsets
                                //    3 = Last placed reference type:
                                //        Read-only parameter!
                                //    4 = Warning mode for connecting named nets:
                                //        0 = Deactivate warning message display
                                //        1 = Activate warning message display
                                //    5 = Elementpick modus:
                                //        0 = Best pick
                                //        1 = Pick element selection
                                //    6 = Generic printer color mode:
                                //        0 = B/W
                                //        1 = Color
                                //    7 = Warning output mode:
                                //        Bit 0: Supress $ rename warnings
                                //        Bit 1: Supress net join warnings
                                //        Bit 2: Supress module port warnings
                                //        Bit 4: Supress group symbol rename
                                //                warnings
                                //        Bit 5: Suppress variant mismatch warnings
                                //    8 = Name prompt mode:
                                //        0 = Autopattern symbol name
                                //        1 = Prompt for symbol name
                                //    9 = Info display flag:
                                //        0 = no automatic info display
                                //        1 = automatic info display
                                //    10 = Info display mode:
                                //        0 = no info display
                                //        1 = complete info display
                                //        2 = net related info only display
                                //    11 = Label rerouting mode:
                                //        0 = no label rerouting
                                //        1 = label rerotuing
                                //    12 = Sub symbol number offset
                                //    13 = Generic plot scale mode:
                                //        0 = fix scale factor
                                //        1 = autosize to page
                                //    14 = Generic color mode:
                                //        0 = black/white
                                //        1 = use current color settings
                                //    15 = HPGL fill mode:
                                //        0 = outline draw
                                //        1 = filled draw
                                //        2 = filled draw, draw wide lines/texts
                                //    16 = Area polygon edit mode:
                                //        0 = don't close polylines
                                //        1 = always close polylines
                                //        2 = polyline close prompt
                                //    17 = Group connection rerouting mode:
                                //        0 = no outside antenna deletion
                                //        1 = delete 1st outside antenna segment
                                //        2 = delete complete outside antenna
```

```
                              //    18 = Default user unit code:
                              //        0 = metric
                              //        1 = imperial
                              //    19 = Plot preview mode:
                              //        0 = none
                              //        1 = plotter pen width
                              //    20 = Autosave interval
                              //    21 = Automatic connection corners
                              //    22 = Angle lock toggle flag
                              //    23 = Default symbol mirroring
                              //    24 = Group move display mode:
                              //        0 = Moving Picture On
                              //        1 = Moving Picture All
                              //        2 = Instant Moving Picture
                              //    25 = Clipboard text placement request flag
                              //    26 = Signal router routing range
                              //    27 = Automatic bustap connection
                              //    28 = Signal router marker scan flag
                              //    29 = Segment move mode:
                              //        0 = Move without neighbours
                              //        1 = Move with neighbours
                              //        2 = Adjust neighbours
                              //        |4 = End point follows segment
                              //    30 = Group angle lock mode:
                              //        0 = Keep group angle lock
                              //        1 = Automatically release group angle lock
                              //    31 = Default text mirror mode (STD14)
                              //    32 = Default text mode (CAP1|CAP7)
                              //    33 = Symbol $noplc plot visibility mode:
                              //        Bit 0 = Set $noplc if plot
                              //                visibility changes
                              //        Bit 1 = Set plot visibility
                              //                if $noplc changes
                              //    34 = Net plan list maximum length [ 3,200]
                              //    35 = Single corner edit flag
                              //    36 = Unroute line creation flag
                              //    37 = Polygon edit autocomplete flag
                              //    38 = Error highlight mode:
                              //        0 = Error highlight
                              //        1 = Error pattern/dash
                              //    39 = Flag - Automirror horizontal bus taps
                              //  [ 40 = System parameter - no write access ]
                              //    41 = Connection split mode:
                              //        0 = No connection split
                              //        1 = Only at 2 pin symbols
                              //        2 = Connection split
                              //    42 = Symbol connection split mode:
                              //        0 = No connection split
                              //        1 = Only at 2 pin symbols
                              //        2 = Connection split
                              //  [ 43 = System parameter - no write access ]
                              //  [ 44 = System parameter - no write access ]
                              //    45 = Display class bits connection (SCM2)
                              //    46 = Display class bits bus (SCM2)
                              //    47 = Display class bits text (SCM2)
                              //    48 = Display class bits comment text (SCM2)
                              //    49 = Display class bits graphic area (SCM2)
                              //    50 = Display class bits graphic line (SCM2)
                              //    51 = Display class bits dotted line (SCM2)
                              //    52 = Display class bits connection area (SCM2)
                              //    53 = Display class bits net area (SCM2)
                              //    54 = Display class bits macro outline (SCM2)
                              //    55 = Display class bits tag (SCM2)
    int;                      // Parameter value
    );
```

Description

The **scm_setintpar** function is used to set **Schematic Editor** integer system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **scm_getintpar** function can be used to query parameter values set with **scm_setintpar**.

See also

Functions **scm_getdblpar**, **scm_getintpar**, **scm_getstrpar**, **scm_setdblpar**, **scm_setstrpar**.

scm_setpartattrib - Set SCM part attribute value (SCM)

Synopsis

```
int scm_setpartattrib(        // Returns status
      string;                 // Part name
      string;                 // Attribute name
      string;                 // Attribute value
      int;                    // Part processing flags:
                              //  Bit 0: remove from screen
                              //  Bit 1: screen redraw
                              //  Bit 2: force assignment, disable variant
      );
```

Description

The **scm_setpartattrib** function assigns a value to the given attribute of the name-specified part. Attribute values with a maximum length of up to 40 characters can be stored. The function returns zero on successful attribute value assignment, (-1) if no valid element is loaded, (-2) on missing and/or invalid parameters, (-3) if the part has not been found or (-4) if the attribute with the given name is not defined on the specified part.

See also

Function **scm_chkattrname**.

scm_setpickconseg - Set SCM default connection pick element (SCM)

Synopsis

```
int scm_pickconseg(           // Returns status
      index C_CONSEG;         // Connection segment
      );
```

Description

The **scm_setpickconseg** function selects the specified connection segment as default element for subsequent connection segment pick operations. The function returns zero if a connection segment has been selected or (-1) on error.

See also

Functions **scm_pickanyelem**, **scm_pickconseg**, **scm_pickbustap**, **scm_pickelem**.

scm_setpickelem - SCM Defaultpickelement setzen (SCM)

Synopsis

```
int scm_setpickelem(          // Returns status
      index C_FIGURE;         // Default pick element
      );
```

Description

The **scm_setpickelem** function sets a default element for subsequent **Schematic Editor** pick operations. The function returns zero if done or nonzero on error.

See also

Function **scm_pickelem**.

scm_setstrpar - Set SCM string parameter (SCM)

Synopsis

```
int scm_setstrpar(            // Returns status
      int [0,[;               // Parameter type/number:
                              //  [ 0 = System parameter write-protected ]
                              //  [ 1 = System parameter write-protected ]
                              //  [ 2 = System parameter write-protected ]
                              //  [ 3 = System parameter write-protected ]
                              //    4 = Last placed text string
                              //    5 = Symbol name pattern
                              //  [ 6 = System parameter write-protected ]
                              //  [ 7 = System parameter write-protected ]
                              //  [ 8 = System parameter write-protected ]
                              //  [ 9 = System parameter write-protected ]
                              //  [ 10 = System parameter write-protected ]
                              //    11 = Current hierachical block reference name
                              //  [ 12 = System parameter write-protected ]
                              //  [ 13 = System parameter write-protected ]
                              //    14 = Autosave path name
      string;                 // Parameter value
      );
```

Description

The **scm_setstrpar** function is used to set **Schematic Editor** string system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **scm_getstrpar** function can be used to query parameter values set with **scm_setstrpar**.

See also

Functions Functions **scm_getdblpar**, **scm_getintpar**, **scm_getstrpar**, **scm_setdblpar**, **scm_setintpar**.

scm_settagdata - Set SCM tag symbol pin destination (SCM)

Synopsis

```
int scm_settagdata(           // Returns status
      index C_FIGURE;         // Tag element
      string;                 // Tag pin name
      string;                 // Tag reference name 1
      string;                 // Tag reference name 2
      );
```

Description

The **scm_settagdata** function assigns the specified tag destination data (tag pin name and tag reference names) to the specified SCM tag element. The function returns zero if the assignment was carried out successfully or nonzero otherwise.

See also

Function **cap_gettagdata**.

scm_storecon - Place SCM connection (SCM)

Synopsis

```
int scm_storecon(             // Returns status
      );
```

Description

The **scm_storepath** function generates a connection on the currently loaded SCM sheet. The connection polygon points are taken from the internal polygon point list previously stored with **bae_storepoint**. The function returns zero if the connection has been successfully generated, (-1) on invalid environment, (-2) if no connection junction point marker is defined or (-3) on invalid connection polygon data (i.e., the internal polygon contains non-orthogonal segments and/or arcs).

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **C_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

scm_storelabel - Place SCM label (SCM)

Synopsis

```
int scm_storelabel(           // Returns status
      string;                 // Label net name
      int [0,2];              // Label type:
                              //    0 = Standard Label
                              //    1 = Module port
                              //    2 = Bus tap
      double;                 // Label X coordinate (STD2)
      double;                 // Label Y coordinate (STD2)
      double;                 // Label rotation angle (STD3)
      int [0,1];              // Label mirror mode (STD14)
      );
```

Description

The **scm_storelabel** function stores a label with the given placement parameters to the currently loaded SCM sheet element. The first label name character must not be a question mark since this character is reserved for module port recognition. The rotation angle is ignored when placing bus taps; bus taps must always be connected to bus connection segments, i.e., their placement coordinates must match a valid bus connection segment. The label library symbol name is set to the specified net name, unless no label symbol with the given net name is available in which case the label symbol named **standard** is used). The **port** and/or **bustap** label symbols are used on default when placing module ports and/or bus taps. The function returns zero if the label has been successfully placed, (-1) on wrong environment, (-2) on missing and/or invalid parameters, (-3) if the label cannot be loaded, (-4) if the label data could not be copied to the current job file or (-5) if the placement data of a bus tap does not match a valid bus connection segment.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **C_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

scm_storepart - Place SCM part or pin (SCM)

Synopsis

```
int scm_storepart(            // Returns status
      & string;               // Reference name
      string;                 // Library symbol name
      double;                 // X coordinate (STD2)
      double;                 // Y coordinate (STD2)
      double;                 // Rotation angle (STD3)
      int [0,7];              // Mirror mode and tag pin type
                              //  (STD14|(CAP6<<1))
      );
```

Description

The **scm_storepart** function stores a part (or pin) with the given placement parameters to the currently loaded SCM sheet (or symbol/label) element. If an empty string is specified for the part name, then the part name is automatically generated and passed back to the caller via the corresponding parameter. The function returns zero if the part has been successfully placed, (-1) if the environment is invalid, (-2) if parameters are missing or invalid, (-3) if the parts cannot be loaded, (-4) if the part data could not be copied to the current job file, (-5) if the part is placed already or (-6) if automatic part name generation failed.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **C_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

scm_storepoly - Place SCM internal polygon (SCM)

Synopsis

```
int scm_storepoly(            // Returns status
      int [0,5];              // Polygon type (CAP2)
      );
```

Description

The **scm_storepoly** function generates a polygon of the specified type on the currently loaded SCM element using the specified placement parameters. The polygon points are taken from the internal polygon point list previously stored with **bae_storepoint**. The function returns zero if the polygon has been successfully generated, (-1) on invalid environment, (-2) on missing and/or invalid parameters or (-3) if the point list is not valid for the specified polygon type.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **C_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

scm_storetext - Place SCM text (SCM)

Synopsis

```
int scm_storetext(            // Returns status
      string;                 // Text string
      double;                 // Text X coordinate (STD2)
      double;                 // Text Y coordinate (STD2)
      double;                 // Text rotation angle (STD3)
      double ]0.0,[;          // Text size (STD2)
      int [0,1];              // Text mirror mode (STD14)
      int [0,[;               // Text mode/style (CAP1|CAP7)
      );
```

Description

The **scm_storetext** function generates a text on the currently loaded SCM element using the specified placement parameters. The function returns nonzero on wrong environment or missing/invalid parameters.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **C_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops. The input text string can be stored to a maximum of up to 40 characters; longer strings cause the function to return with an invalid parameter error code.

See also

Function **scm_attachtextpos**.

C.4 PCB Design System Functions

This section describes (in alphabetical order) the PCB design system functions of the **Bartels User Language**. See Appendix C.1 for function description notations.

C.4.1 Layout Data Access Functions

The following **User Language** system functions are assigned to caller type LAY; i.e., they can be called from the **Layout Editor**, the **Autorouter** and the **CAM Processor** interpreter environment of the **Bartels AutoEngineer**:

lay_defelemname - Layout setup default element name (LAY)

Synopsis

```
string lay_defelemname(       // Returns default layout element name
      );
```

Description

The **lay_defelemname** function returns the default layout element name defined in the BAE setup file.

lay_deflibname - Layout setup default library name (LAY)

Synopsis

```
string lay_deflibname(        // Returns default library name
      );
```

Description

The **lay_deflibname** function returns the default layout library name defined in the BAE setup file.

lay_defusrunit - Layout setup default user units (LAY)

Synopsis

```
int lay_defusrunit(           // Returns default user units (STD7)
      );
```

Description

The **lay_defusrunit** function returns the default user units mode defined in the BAE setup file (1=Inch, 0=mm).

lay_doclayindex - Layout documentary layer display index (LAY)

Synopsis

```
int lay_doclayindex(          // Documentary layer display index
      int [0,99];             // Documentary layer number
      );
```

Description

The **lay_doclayindex** function returns the documentary layer display index for the specified documentary layer, or (-1) if an invalid documentary layer number was specified.

See also
Functions **lay_doclayname**, **lay_doclayside**, **lay_doclaytext**.

lay_doclayname - Layout setup documentary layer name (LAY)

Synopsis

```
string lay_doclayname(        // Returns documentary layer name
      int [0,99];             // Documentary layer number
      );
```

Description

The **lay_doclayname** function returns the documentary layer name defined in the BAE setup file for the specified documentary layer number.

See also

Functions **lay_doclayindex**, **lay_doclayside**, **lay_doclaytext**.

lay_doclayside - Layout setup documentary layer side mode (LAY)

Synopsis

```
int lay_doclayside(           // Returns documentary layer side mode:
                              //    (-2) = invalid layer number
                              //    (-1) = none
                              //    ( 0) = side 1
                              //    ( 1) = side 2
                              //    ( 2) = both sides
      int [0,99];             // Documentary layer number
      );
```

Description

The **lay_doclayside** function returns the documentary layer side mode defined in the BAE setup file for the specified documentary layer number. Documentary layer side mode (-1) refers to None (i.e., Side 1, Side 2 and Both Sides are selectable), mode 0 refers to Side 1, mode 1 refers to Side 2, mode 3 refers to Both Sides. The function returns (-2) if an invalid layer number has been specified.

See also

Functions **lay_doclayindex**, **lay_doclayname**, **lay_doclaytext**.

lay_doclaytext - Layout setup documentary layer text mode (LAY)

Synopsis

```
int lay_doclaytext(           // Returns doc. layer text mode (LAY2)
      int [0,99];             // Documentary layer number
      );
```

Description

The **lay_doclaytext** function returns the documentary layer text mode (LAY2) defined in the BAE setup file for the specified documentary layer. The function returns (-1) if an invalid layer number has been specified.

See also

Functions **lay_doclayindex**, **lay_doclayname**, **lay_doclayside**.

lay_figboxtest - Check layout element rectangle cross (LAY)

Synopsis

```
int lay_figboxtest(           // Returns status
      & index L_FIGURE;       // Element
      double;                 // Rectangle left border (STD2)
      double;                 // Rectangle lower border (STD2)
      double;                 // Rectangle right border (STD2)
      double;                 // Rectangle upper border (STD2)
      );
```

Description

The **lay_figboxtest** function checks if the given figure list element crosses the given rectangle. The function return nonzero if the element boundaries cross the given rectangle.

lay_findconpart - Find layout part index of a named part (LAY)

Synopsis

```
int lay_findconpart(          // Returns status
      string;                 // Part name
      & index L_CPART;        // Returns part index
      );
```

Description

The **lay_findconpart** function searches the layout connection list part index with the specified part name. The function returns zero if the part has been found or nonzero otherwise.

See also

Functions **lay_findconpartpin**, **lay_findcontree**.

lay_findconpartpin - Find layout part pin index of a named part pin (LAY)

Synopsis

```
int lay_findconpartpin(       // Returns status
      string;                 // Pin name
      index L_CPART;          // Net list part index
      & index L_CPIN;         // Returns net list part pin index
      );
```

Description

The **lay_findconpartpin** function searches a layout connection list part for the part pin index with the specified pin name. The function returns zero if the part pin has been found or nonzero otherwise.

See also

Functions **lay_findconpart**, **lay_findcontree**.

lay_findcontree - Find layout net index of a named net (LAY)

Synopsis

```
int lay_findcontree(          // Returns status
      string;                 // Net name
      & index L_CNET;         // Returns net index
      );
```

Description

The **lay_findcontree** function searches the layout connection list net index with the specified net name. The function returns zero if the net has been found or nonzero otherwise.

See also

Functions **lay_findconpart**, **lay_findconpartpin**.

lay_getplanchkparam - Get layout DRC parameters (LAY)

Synopsis

```
void lay_getplanchkparam(
      & double;               // Returns distance trace - trace (STD2)
      & double;               // Returns distance trace - copper (STD2)
      & double;               // Returns distance copper - copper (STD2)
      & double;               // Trace default width (STD2)
      & string;               // Block name
      int [-6,99];            // Signal layer code (LAY1)
                              //  (layer!=(-1) only allowed for BAE HighEnd)
      int [0,0[;              // DRC block number
      );
```

Description

The **lay_getplanchkparam** function is used to retrieve minimum clearance parameter settings for the design rule check (DRC). The DRC parameter values are returned with the corresponding function parameters. **BAE Professional**, **BAE Economy** and **BAE Light** allow only for the query of global parameters for layer code -1 (All Layers) and DRC parameter block 0. **BAE HighEnd** also allows for querying DRC parameter blocks with layer-specific clearance distance values being assigned to any signal layer (layer codes 0 to 99), the top signal layer (layer code -5) and inside signal layers (layer code -5). The DRC block 0 is always defined and contains the global DRC parameters, and there is always a DRC parameter preference defined for layer code -1 (All Layers).

See also

Function **lay_setplanchkparam**.

lay_getpowplanetree - Get layout power plane tree number (LAY)

Synopsis

```
int lay_getpowplanetree(      // Returns net tree number or (-1)
      double;                 // X coordinate (STD2)
      double;                 // Y coordinate (STD2)
      int;                    // Power layer number (LAY1)
      double;                 // Drill diameter (STD2)
      );
```

Description

The **lay_getpowplanetree** function returns the net tree number of the signal connected via the specified power layer coordinate. **lay_getpowplanetree** checks the given power layer. The specified drill diameter is taken under consideration for distance calculations. This function can be utilized for designating power layer net connections when using split power planes. The function returns (-1) if no signal is connected at the given power layer coordinate or if an invalid power layer number was specified.

Warning

The coordinate values passed to **lay_getpowplanetree** are interpreted as absolute coordinates on the currently loaded element. This means that scan offsets must be considered on coordinate checks when calling **lay_getpowplanetree** indirectly via **lay_scan*** functions.

lay_getpowpolystat - Layout power layer polygon status query (LAY)

Synopsis

```
int lay_getpowpolystat(       // Returns status
      index L_FIGURE;         // Figure list element - power layer polygon
      & int;                  // Returns flag:
                              //    polgyon crosses/touches board outline
      & int;                  // Returns flag:
                              //    polygon crosses other power layer polygon(s)
      );
```

Description

The **lay_getpowpolystat** checks the status of the specified power layer polygon (split power plane). The return flags indicate whether the power layer polygon touches or crosses the board outline and/or any other power layer polygon. The function returns zero if the query was successful, (-1) for invalid parameters/environment or (-2) if the specified figure list element is not a power layer polygon.

lay_getrulecnt - Get rule count for specific object (LAY)

Synopsis

```
int lay_getrulecnt(           // Returns rule count or (-1) on error
      int;                    // Object class code
      int;                    // Object ident code (int or index type)
      );
```

Description

The **lay_getrulecnt** function is used for determining the number of rules attached to a specific object. The object can be the currently loaded element (object class code 0 with `int` value 0 passed for the object ident code), a figure list element of the currently loaded element (object class code 1 with valid **L_FIGURE** index type value passed for the object ident code), or a pool list element (object class code 2 with valid **L_POOL** index type value passed for the object ident code). The function returns a (non-negative) rule count or (-1) on error. The rule count determines the valid range for rule list indices to be passed to the **lay_getrulename** function for getting object-specific rule names. The **lay_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **lay_getrulecnt** function.

See also

Functions **lay_getrulename**, **lay_ruleerr**, **lay_rulefigatt**, **lay_rulefigdet**, **lay_rulelaysatt**, **lay_rulelaysdet**, **lay_ruleplanatt**, **lay_ruleplandet**, **lay_rulequery**; **Neural Rule System** and Rule System Compiler.

lay_getrulename - Get rule name from specific layout object (LAY)

Synopsis

```
int lay_getrulename(          // Returns nonzero on error
      int;                    // Object class code
      int;                    // Object ident code (int or index type)
      int [0,[;               // Rule name list index
      & string;               // Rule name result
      );
```

Description

The **lay_getrulename** function is used to get the name of an index-specified rule assigned to the specified object. The object can be the currently loaded element (object class code 0 with `int` value 0 passed for the object ident code), a figure list element of the currently loaded element (object class code 1 with valid **L_FIGURE** index type value passed for the object ident code), or a pool list element (object class code 2 with valid **L_POOL** index type value passed for the object ident code). The rule name list index to be specified can be determined using the **lay_getrulecnt** function. The rule name is returned with the last function parameter. The function returns zero on success or nonzero on error. The **lay_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **lay_getrulename** function.

See also

Functions **lay_getrulecnt**, **lay_ruleerr**, **lay_rulefigatt**, **lay_rulefigdet**, **lay_rulelaysatt**, **lay_rulelaysdet**, **lay_ruleplanatt**, **lay_ruleplandet**, **lay_rulequery**; **Neural Rule System** and Rule System Compiler.

lay_getscclass - Get currently scanned layout class (LAY)

Synopsis

```
int lay_getscclass(           // Returns layout element class:
                              //    0 = Layout
                              //    1 = Part
                              //    2 = Padstack
                              //    3 = Pad
                              //    (-1) otherwise
      );
```

Description

The **lay_getscclass** function returns the currently scanned layout element class. **lay_getscclass** is intended for use in the callback functions of **lay_scanall**, **lay_scanfelem** or **lay_scanpool** only. The function returns (-1) if no scan function is active or if no layout element is currently scanned.

See also

Functions **lay_scanall**, **lay_scanfelem**, **lay_scanpool**.

lay_getscpartrpidx - Get currently scanned layout part (LAY)

Synopsis

```
index L_NREF lay_getscpartrpidx   // Returns layout part index or (-1)
      );
```

Description

The **lay_getscpartrpidx** function returns the named reference index of the currently scanned layout part. This allows for designating the layout part to which the currently scanned trace, polygon, text, pin, etc. belongs to. **lay_getscpartrpidx** is intended for use in the callback functions of **lay_scanall**, **lay_scanfelem** or **lay_scanpool** only. The function returns (-1) if no scan function is active or if no part is currently scanned.

See also

Functions **lay_scanall**, **lay_scanfelem**, **lay_scanpool**.

lay_getscrefpidx - Get currently scanned layout library element (LAY)

Synopsis

```
index L_POOL lay_getscrefpidx     // Returns pool index or (-1) if outside macro
      );
```

Description

The **lay_getscrefpidx** function returns the currently scanned macro reference pool index. This allows for designating the layout library element to which the currently scanned trace, polygon, text, etc. belongs to. **lay_getscrefpidx** is intended for use in the callback functions of **lay_scanall**, **lay_scanfelem** or **lay_scanpool** only. The function returns (-1) if no scan function is active or if no macro is currently scanned.

See also

Functions **lay_scanall**, **lay_scanfelem**, **lay_scanpool**.

lay_getscstkcnt - Get layout scan function stack depth (LAY)

Synopsis

```
int lay_getscstkcnt(          // Returns scan stack depth
      );
```

Description

The **lay_getscstkcnt** function returns the current layout scan function stack depth. I.e., **lay_getscstkcnt** can be used for control purposes in the callback functions of **lay_scanall**, **lay_scanfelem** or **lay_scanpool**.

See also

Functions **lay_scanall**, **lay_scanfelem**, **lay_scanpool**.

lay_getsctextdest - Get scanned layout text line destination (LAY)

Synopsis

```
int lay_getsctextdest(        // Returns status
      & double;               // Returns text line destination X coordinate (STD2)
      & double;               // Returns text line destination Y coordinate (STD2)
      );
```

Description

The **lay_getsctextdest** retrieves the text base line destination/end point coordinates of the currently scanned text. **lay_getsctextdest** is intended for use in the callback functions of **lay_scanall**, **lay_scanfelem** or **lay_scanpool**. The function returns 1 for successful queries or zero otherwise.

See also

Functions **lay_scanall**, **lay_scanfelem**, **lay_scanpool**.

lay_gettreeidx - Find layout net index of a tree (LAY)

Synopsis

```
int lay_gettreeidx(           // Returns status
      int;                    // Net tree number
      & index L_CNET;         // Returns net index
      );
```

Description

The **lay_gettreeidx** function searches the layout connection list net index with the specified net tree number. The function returns zero if the net has been found or nonzero otherwise.

lay_grpdisplay - Layout setup group display layer (LAY)

Synopsis

```
int lay_grpdisplay(           // Returns documentary layer number
      );
```

Description

The **lay_grpdisplay** function returns the group display documentary layer number defined in the BAE setup file.

lay_lastfigelem - Get last modified layout figure list element (LAY)

Synopsis

```
int lay_lastfigelem(          // Returns status
      & index L_FIGURE;       // Returns figure list index
      );
```

Description

The **lay_lastfigelem** function gets the last created and/or modified layout figure list element and returns the corresponding figure list index with the return parameter. The function returns zero if such an element exists or nonzero else.

lay_maccoords - Get layout (scanned) macro coordinates (LAY)

Synopsis

```
void lay_maccoords(
      & double;               // Macro X coordinate (STD2)
      & double;               // Macro Y coordinate (STD2)
      & double;               // Macro rotation angle (STD3)
      & int;                  // Macro mirror mode (STD14)
      & int;                  // Macro layer (LAY1 for Pad on Padstack)
      );
```

Description

The **lay_maccoords** function returns with its parameters the placement data of the currently scanned macro. This function is intended for use in the macro callback function of **lay_scanall**, **lay_scanfelem** or **lay_scanpool** only (otherwise zero/default values are returned).

See also

Functions **lay_scanall**, **lay_scanfelem**, **lay_scanpool**.

lay_macload - Load layout macro element to memory (STD)

Synopsis

```
int lay_macload(              // Returns status
      & index L_POOL;         // Macro pool element index
      string;                 // DDB file name
      string;                 // Element name
      int [100,[;             // Element DDB class (STD1)
      );
```

Description

The **lay_macload** function loads the specified layout library symbol to memory and returns the macro pool element index with the corresponding parameter. The function returns zero if the element was successfully loaded, (-1) on file access errors, (-2) on missing and/or invalid parameters or 1 if referenced macros (library elements) are missing. **lay_macload** is intended to be applied by features such as layout symbol browsers for examining library file contents. The **lay_macrelease** function can be used to unload and/or release macro elements from memory.

See also

Function **lay_macrelease**.

lay_macrelease - Unload/release layout macro element from memory (STD)

Synopsis

```
void lay_macrelease(          // Returns status
      index L_POOL;           // Macro pool element index
      );
```

Description

The **lay_macrelease** function unloads and/or releases the layout library symbol specified with the macro pool index parameter from memory. **lay_macrelease** is intended to be used together with the **lay_macload** function.

See also
Function **lay_macload**.

lay_menulaylinecnt - Get the layer menu lines count (LAY)

Synopsis

```
int lay_menulaylinecnt(       // Returns number of menu layer lines
      );
```

Description

The **lay_menulaylinecnt** function returns the number of menu lines currently defined with the layout signal layer menu. The layout signal layer menu can be customized using the **BSETUP** utility program which allows for the definition of up to 12 signal layer entries with layer number and layer name.

See also

Functions **lay_menulaylinelay**, **lay_menulaylinename**.

lay_menulaylinelay - Get layer number of specified layer menu line (LAY)

Synopsis

```
int lay_menulaylinelay(       // Returns menu layer line layer number
      int [0,11];             // Menu line number
      );
```

Description

The **lay_menulaylinelay** function returns the layer number assigned to the specified menu line of the layout signal layer menu. The layout signal layer menu can be customized using the **BSETUP** utility program which allows for the definition of up to 12 signal layer entries with layer number and layer name.

See also

Functions **lay_menulaylinecnt**, **lay_menulaylinename**.

lay_menulaylinename - Get name of specified layer menu line (LAY)

Synopsis

```
string lay_menulaylinename(   // Returns menu layer line name
      int [0,11];             // Menu line number
      );
```

Description

The **lay_menulaylinename** function returns the layer name assigned to the specified menu line of the layout signal layer menu. The layout signal layer menu can be customized using the **BSETUP** utility program which allows for the definition of up to 12 signal layer entries with layer number and layer name.

See also

Functions **lay_menulaylinecnt**, **lay_menulaylinelay**.

lay_nrefsearch - Search named layout reference (LAY)

Synopsis

```
int lay_nrefsearch(           // Returns status
      string;                 // Reference name
      & index L_FIGURE;       // Returns figure list index
      );
```

Description

The **lay_nrefsearch** function searches for the specified named reference on the currently loaded layout element. The figure list index is set accordingly if the named reference is found. The function returns zero if the named reference has been found or nonzero otherwise.

lay_planmidlaycnt - Get layout inside layer count (LAY)

Synopsis

```
int lay_planmidlaycnt(        // Returns layout inside layer count
```

```
      );
```

Description

The **lay_planmidlaycnt** function returns the inside layer count of the currently loaded layout.

See also

Function **lay_plantoplay**.

lay_plantoplay - Get layout top layer (LAY)

Synopsis

```
int lay_plantoplay(           // Returns layout top layer (LAY1)
      );
```

Description

The **lay_plantoplay** function returns the top layer setting of the currently loaded layout element or signal layer 2 if no layout element is currently loaded.

See also

Function **lay_planmidlaycnt**.

lay_pltmarklay - Layout setup plot marker layer (LAY)

Synopsis

```
int lay_pltmarklay(           // Returns documentary layer number
      );
```

Description

The **lay_pltmarklay** function returns the plot marker documentary layer number defined in the BAE setup file.

lay_ruleerr - Layout rule system error status query (LAY)

Synopsis

```
void lay_ruleerr(
      & int;              // Error item code
      & string;           // Error item string
      );
```

Description

The **lay_ruleerr** function provides information on the current Rule System error state, and thus can be used to determine the error reason after an unsuccessful call to one of the Rule System management functions.

Diagnosis

The Rule System error state can be determined by evaluating the parameters returned with the **lay_ruleerr** function. The returned error item string identifies the error-causing element if needed. The possible error code values correspond with Rule System error conditions according to the following table:

Error Code	Meaning
0	Rule System operation completed without errors
1	Rule System out of memory
2	Rule System internal error **<e>**
3	Rule System function parameter invalid
128	Rule System DB file create error
129	Rule System DB file read/write error
130	Rule System DB file wrong type
131	Rule System DB file structure bad
132	Rule System DB file not found
133	Rule System DB file other error (internal error)
134	Rule System rule **<r>** not found in rule database
135	Rule System rule bad DB format (internal error **<e>**)
136	Rule System object not found
137	Rule System object double defined (internal error)
138	Rule System incompatible variable **<v>** definition
139	Rule System Rule **<r>** compiled with incompatible RULECOMP version

Depending on the error condition the error item string can describe a rule **<r>**, a variable **<v>** or an (internal) error status **<e>**. DB file errors refer to problems accessing the Rule System database file **brules.vdb** in the BAE programs directory. Internal errors usually refer to Rule System implementation gaps and should be reported to Bartels.

See also

Functions **lay_getrulecnt**, **lay_getrulename**, **lay_rulefigatt**, **lay_rulefigdet**, **lay_rulelaysatt**, **lay_rulelaysdet**, **lay_ruleplanatt**, **lay_ruleplandet**, **lay_rulequery**; **Neural Rule System** and Rule System Compiler.

lay_rulefigatt - Attach rule(s) to layout figure list element (LAY)

Synopsis

```
int lay_rulefigatt(           // Returns nonzero on error
      index L_FIGURE;         // Figure list element index
      void;                   // Rule name string or rule name list array
      );
```

Description

The **lay_rulefigatt** function is used to attach a *new* set of name-specified rules to the figure list element specified with the first function parameter. Either a single rule name (i.e., a value of type **string**) or a set of rule names (i.e., an array of type **string**) can be specified with the second function parameter. Note that any rules previously attached to the figure list element are detached before attaching the new rule set. The function returns zero on success or nonzero on error. The **lay_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **lay_rulefigatt** function.

See also

Functions **lay_getrulecnt**, **lay_getrulename**, **lay_ruleerr**, **lay_rulefigdet**, **lay_rulelaysatt**, **lay_rulelaysdet**, **lay_ruleplanatt**, **lay_ruleplandet**, **lay_rulequery**; **Neural Rule System** and Rule System Compiler.

lay_rulefigdet - Detach rules from layout figure list element (LAY)

Synopsis

```
int lay_rulefigdet(           // Returns nonzero on error
      index L_FIGURE;         // Figure list element index
      );
```

Description

The **lay_rulefigdet** function detaches *all* currently attached rules from the figure list element specified with the function parameter. The function returns zero on success or nonzero on error. The **lay_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **lay_rulefigdet** function.

See also

Functions **lay_getrulecnt**, **lay_getrulename**, **lay_ruleerr**, **lay_rulefigatt**, **lay_rulelaysatt**, **lay_rulelaysdet**, **lay_ruleplanatt**. **lay_ruleplandet**, **lay_rulequery**; **Neural Rule System** and Rule System Compiler.

lay_rulelaysatt - Attach rule(s) to layout layer stackup (LAY)

Synopsis

```
int lay_rulelaysatt(          // Returns nonzero on error
      int [0,111];            // Layer stackup index
      void;                   // Rule name string or rule name list array
      );
```

Description

The **lay_rulelaysatt** function is used to attach a *new* set of name-specified rules to the layer stackup specified by the layer stackup index. Either a single rule name (i.e., a value of type **string**) or a set of rule names (i.e., an array of type **string**) can be specified with the rule name function parameter. Note that any rules previously attached to the layer stackup will be detached before attaching the new rule set. The function returns zero on success or nonzero on error. The **lay_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **lay_rulelaysatt** function.

See also

Functions **lay_getrulecnt**, **lay_getrulename**, **lay_ruleerr**, **lay_rulefigatt**, **lay_rulefigdet**, **lay_rulelaysdet**, **lay_ruleplanatt**, **lay_ruleplandet**, **lay_rulequery**; **Neural Rule System** and Rule System Compiler.

lay_rulelaysdet - Detach rules from layout layer stackup (LAY)

Synopsis

```
int lay_rulelaysdet(          // Returns nonzero on error
      int [0,111];            // Layer stackup index
      );
```

Description

The **lay_rulelaysdet** function detaches *all* attached rules from the layer stackup specified by the layer stackup index. The function returns zero on success or nonzero on error. The **lay_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **lay_rulelaysdet** function.

See also

Functions **lay_getrulecnt**, **lay_getrulename**, **lay_ruleerr**, **lay_rulefigatt**, **lay_rulefigdet**, **lay_rulelaysatt**, **lay_ruleplanatt**, **lay_ruleplandet**, **lay_rulequery**; **Neural Rule System** and Rule System Compiler.

lay_ruleplanatt - Attach rule(s) to currently loaded layout element (LAY)

Synopsis

```
int lay_ruleplanatt(          // Returns nonzero on error
      void;                   // Rule name string or rule name list array
      );
```

Description

The **lay_ruleplanatt** function is used to attach a *new* set of name-specified rules to the currently loaded element. Either a single rule name (i.e., a value of type **string**) or a set of rule names (i.e., an array of type **string**) can be specified with the function parameter. Note that any rules previously attached to the current element will be detached before attaching the new rule set. The function returns zero on success or nonzero on error. The **lay_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **lay_ruleplanatt** function.

See also

Functions **lay_getrulecnt**, **lay_getrulename**, **lay_ruleerr**, **lay_rulefigatt**, **lay_rulefigdet**, **lay_rulelaysatt**, **lay_rulelaysdet**; **lay_ruleplandet**, **lay_rulequery**; **Neural Rule System** and Rule System Compiler.

lay_ruleplandet - Detach rules from currently loaded layout element (LAY)

Synopsis

```
int lay_ruleplandet(          // Returns nonzero on error
      );
```

Description

The **lay_ruleplandet** function detaches *all* currently attached rules from the currently loaded element. The function returns zero on success or nonzero on error. The **lay_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **lay_ruleplandet** function.

See also

Functions **lay_getrulecnt**, **lay_getrulename**, **lay_ruleerr**, **lay_rulefigatt**, **lay_rulefigdet**, **lay_rulelaysatt**, **lay_rulelaysdet**; **lay_ruleplanatt**, **lay_rulequery**; **Neural Rule System** and Rule System Compiler.

lay_rulequery - Perform rule query on specific layout object (LAY)

Synopsis

```
int lay_rulequery(            // Returns hit count or (-1) on error
      int;                    // Object class code
      int;                    // Object ident code (int or index type)
      string;                 // Subject name
      string;                 // Predicate name
      string;                 // Query command string
      & void;                 // Query result
      []                      // Optional query parameters of requested type
      );
```

Description

The **lay_rulequery** function is used to perform a rule query on a specific object. The object can be the currently loaded element (object class code 0 with **int** value 0 passed for the object ident code), a figure list element of the currently loaded element (object class code 1 with valid **L_FIGURE** index type value passed for the object ident code), or a pool list element (object class code 2 with valid **L_POOL** index type value passed for the object ident code). The rule query function requires a rule subject, a rule predicate and a query command string to be specified with the corresponding function parameters. The query command string can contain one query operator and a series of value definition operators. The following query operators are implemented:

?d	for querying **int** values
?f	for querying **double** values
?s	for querying **string** values

The query operator can optionally be preceded with one of the following selection operators:

+	for selecting the maximum of all matching values
-	for selecting the minimum of all matching values

The **+** operator is used on default (e.g., when omitting the selection operator). The rule query resulting value is passed back to the caller with the query result parameter. This means that the query result parameter data type must comply with the query operator (**int** for **?d**, **double** for **?f**, **string** for **?s**). The query command string can also contain a series of value definition operators such as:

%d	for specifying **int** values
%f	for specifying **double** values
%s	for specifying **string** values

Each value definition parameter is considered a placeholder for specific data to be passed with optional parameters. Note that these optional parameters must comply with the query command in terms of specified sequence and data types. The **lay_rulequery** function returns a (non-negative) hit count denoting the number of value set entries matched by the query. The function return value is (-1) on error. The **lay_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **lay_rulequery** function.

Examples

With the rule

```
rule somerule
{
    subject subj
    {
        pred := ("A", 2);
        pred := ("A", 4);
        pred := ("B", 1);
        pred := ("C", 3);
        pred := ("B", 6);
        pred := ("D", 5);
        pred := ("D", 6);
        pred := ("A", 3);
    }
}
```

defined and attached to the currently loaded element, the **lay_rulequery** call

```
    hitcount = lay_rulequery(0,0,"subj","pred","%s ?d",intresult,"A") ;
```

sets the **int** variable **hitcount** to 3, and the **int** variable **intresult** to 4, whilst a call such as

```
    hitcount = lay_rulequery(0,0,"subj","pred","-?s %d",strresult,6) ;
```

sets **hitcount** to 2 and the **string** variable **strresult** to **B**.

See also

Functions **lay_getrulecnt**, **lay_getrulename**, **lay_ruleerr**, **lay_rulefigatt**, **lay_rulefigdet**, **lay_rulelaysatt**, **lay_rulelaysdet**, **lay_ruleplanatt**, **lay_ruleplandet**; **Neural Rule System** and Rule System Compiler.

lay_scanall - Scan all layout figure list elements (LAY)

Synopsis

```
int lay_scanall(              // Returns scan status
      double;                 // Scan X offset (STD2)
      double;                 // Scan Y offset (STD2)
      double;                 // Scan rotation angle (STD3)
      int [0,1];              // Element in workspace flag (STD10)
      int [0,1];              // Connectivity scan allowed flag:
                              //    0 = no scan allowed
                              //    1 = scan allowed
      * int;                  // Macro callback function
      * int;                  // Polygon callback function
      * int;                  // Path callback function
      * int;                  // Text callback function
      * int;                  // Drill callback function
      * int;                  // Layer check function
      * int;                  // Level check function
      );
```

Description

The **lay_scanall** function scans all figure list elements placed on the currently loaded layout element with all hierarchy levels. User-defined scan functions are automatically activated depending on the currently scanned element type. If a certain callback function should not be referenced, then the corresponding parameter must be set to the keyword **NULL**. The function returns nonzero on invalid parameter specifications or if one of the referenced user functions has returned a scan error status.

Macro callback function

```
int macrofuncname(
      index L_MACRO macro,    // Macro index
      index L_POOL pool,      // Pool element index
      int macinws,            // Macro in workspace flag (STD10)
      string refname,         // Macro Reference name
      index L_LEVEL level     // Macro signal level
      )
{
      // Macro callback function statements
      :
      return(contscan);
}
```

The **lay_maccoords** function can be used for determining the macro placement coordinates. The return value of the macro callback function must be 1 for continue scan, 0 for stop scan or (-1) on error.

Polygon callback function

```
int polyfuncname(
      index L_POLY poly,      // Polygon index
      int layer,              // Polygon layer (LAY1)
      int polyinws,           // Polygon in workspace flag (STD10)
      int tree,               // Polygon tree number or (-1)
      index L_LEVEL level     // Polygon signal level
      )
{
      // Polygon callback function statements
      :
      return(errstat);
}
```

The return value of the polygon callback function must be zero if scan ok or nonzero on error.

Path callback function

```
int pathfuncname(
      index L_LINE path,      // Path index
      int layer,              // Path layer (LAY1)
      int pathinws,           // Path in workspace flag (STD10)
      index L_LEVEL level     // Path signal level
      )
{
      // Path callback function statements
      :
      return(errstat);
}
```

The return value of the path callback function must be zero if scan ok or nonzero on error.

Text callback function

```
int textfuncname(
      index L_TEXT text,      // Text index
      double x,               // Text X coordinate (STD2)
      double y,               // Text Y coordinate (STD2)
      double angle,           // Text rotation angle (STD3)
      int mirr,               // Text mirror mode (STD14)
      int layer,              // Text layer (LAY1)
      double size,            // Text size (STD2)
      string textstr,         // Text string
      int textinws            // Text in workspace flag (STD10)
      )
{
      // Text callback function statements
      :
      return(errstat);
}
```

The return value of the text callback function must be zero if scan ok or nonzero on error.

Drill callback function

```
int drillfuncname(
      index L_DRILL drill,    // Drill index
      double x,               // Drill X coordinate (STD2)
      double y,               // Drill Y coordinate (STD2)
      int drillinws,          // Drill in workspace flag (STD10)
      int tree,               // Drill tree number or (-1)
      index L_LEVEL level     // Drill signal level
      )
{
      // Drill callback function statements
      :
      return(errstat);
}
```

The return value of the drill callback function must be zero if scan ok or nonzero on error.

Layer check function

```
int laycheckfuncname(
      int layer               // Scanned layer (LAY1)
      )
{
      // Layer check function statements
      :
      return(contscan);
}
```

The return value of the layer check function must be 1 for continue scan, 0 for stop scan or (-1) on error. The scan process can be accelerated considerably if restricted to the interesting layers with this function.

Level check function

```
int levcheckfuncname(
      index L_LEVEL level     // Scanned signal level
      )
{
      // Level check function statements
      :
      return(contscan);
}
```

The return value of the level check function must be 1 for continue scan, 0 for stop scan or (-1) on error. The scan process can be accelerated considerably if restricted to interesting signal levels with this function.

See also

Functions **lay_maccoords**, **lay_scanfelem**, **lay_scanpool**.

lay_scanfelem - Scan layout figure list element (LAY)

Synopsis

```
int lay_scanfelem(            // Returns scan status
      index L_FIGURE;         // Figure list element index
      double;                 // Scan X offset (STD2)
      double;                 // Scan Y offset (STD2)
      double;                 // Scan rotation angle (STD3)
      int [0,1];              // Element in workspace flag (STD10)
      int [0,1];              // Connectivity scan allowed flag:
                              //    0 = no scan allowed
                              //    1 = scan allowed
      * int;                  // Macro callback function
      * int;                  // Polygon callback function
      * int;                  // Path callback function
      * int;                  // Text callback function
      * int;                  // Drill callback function
      * int;                  // Layer check function
      * int;                  // Level check function
      );
```

Description

The **lay_scanfelem** function scans the specified layout figure list element with all hierarchy levels. User-defined callback functions for the currently scanned element type are automatically activated. If a certain callback function should not be referenced, then the corresponding parameter must be set to the keyword **NULL**. The return value of **lay_scanfelem** is nonzero on invalid parameter specifications or if one of the referenced user functions has returned a scan error status. See **lay_scanall** for the scan function definitions.

See also
Functions **lay_maccoords**, **lay_scanall**, **lay_scanpool**.

lay_scanpool - Scan layout pool element (LAY)

Synopsis

```
int lay_scanpool(             // Returns scan status
      void;                   // Pool element index
      double;                 // Scan X offset (STD2)
      double;                 // Scan Y offset (STD2)
      double;                 // Scan rotation angle (STD3)
      int [0,1];              // Element in workspace flag (STD10)
      int [0,1];              // Connectivity scan allowed flag:
                              //    0 = no scan allowed
                              //    1 = scan allowed
      * int;                  // Macro callback function
      * int;                  // Polygon callback function
      * int;                  // Path callback function
      * int;                  // Text callback function
      * int;                  // Drill callback function
      * int;                  // Layer check function
      * int;                  // Level check function
      );
```

Description

The **lay_scanpool** function scans the specified layout pool element with all hierarchy levels. User-defined callback functions for the currently scanned element type are automatically activated. If a certain callback function should not be referenced, then the corresponding parameter must be set to the keyword **NULL**. The return value of **lay_scanpool** is nonzero on invalid parameter specifications or if one of the referenced user functions has returned a scan error status. See **lay_scanall** for the callback function definitions.

See also

Functions **lay_maccoords**, **lay_scanall**, **lay_scanfelem**.

lay_setfigcache - Fill layout figure list access cache (LAY)

Synopsis

```
void lay_setfigcache(
      );
```

Description

The **lay_setfigcache** function fills the layout cache for fast figure list element access.

lay_setplanchkparam - Set layout DRC parameters (LAY)

Synopsis

```
int lay_setplanchkparam(      // Returns status
      double ]0.0,[;          // Distance trace - trace (STD2)
      double ]0.0,[;          // Distance trace - copper (STD2)
      double ]0.0,[;          // Distance copper - copper (STD2)
      double;                 // Trace default width (STD2)
      string;                 // Block name
      int [-6,99];            // Signal layer code (LAY1)
                              //  (layer!=(-1) only allowed for BAE HighEnd)
      int [0,0[;              // DRC block number
      );
```

Description

The **lay_setplanchkparam** function sets **Layout Editor** clearance distance values for the design rule check (DRC). The function returns nonzero if invalid distance parameters are specified. **BAE Professional**, **BAE Economy** and **BAE Light** allow only for the specification of global parameters for layer code -1 (All Layers) and DRC parameter block 0. **BAE HighEnd** also allows for specifying DRC parameter blocks with layer-specific clearance distance values to be assigned to any signal layer (layer codes 0 to 99), the top signal layer (layer code -5) and inside signal layers (layer code -5). The DRC block 0 is always defined and contains the global DRC parameters, and there is always a DRC parameter preference defined for layer code -1 (All Layers).

See also

Function **lay_getplanchkparam**.

lay_toplayname - Layout setup top layer name (LAY)

Synopsis

```
string lay_toplayname(        // Returns top layer name
      );
```

Description

The **lay_toplayname** function returns the top signal layer name defined in the BAE setup file.

lay_vecttext - Vectorize layout text (LAY)

Synopsis

```
int lay_vecttext(             // Returns status
      double;                 // Text X coordinate (STD2)
      double;                 // Text Y coordinate (STD2)
      double;                 // Text rotation angle (STD3)
      int [0,1];              // Text mirror mode (STD14)
      double ]0.0,[;          // Text size (STD2)
      int [0,1];              // Text physical flag:
                              //    0 = logical
                              //    1 = physical
      int [0,2];              // Layer mirror mode:
                              //    0 = mirror off
                              //    1 = mirror X
                              //    2 = mirror Y
      int [0,[;               // Text style (LAY14)
      string;                 // Text string
      * int;                  // Text vectorize function
      );
```

Description

The **lay_vecttext** function vectorizes the specified text using the currently loaded text font. The referenced text vectorize user function is automatically called for each text segment. The function returns nonzero if invalid parameters have been specified or if the referenced user function returns nonzero.

Text vectorize function

```
int vecfuncname(
      double x1,              // Start point X coordinate (STD2)
      double y1,              // Start point Y coordinate (STD2)
      double x2,              // End point X coordinate (STD2)
      double y2               // End point Y coordinate (STD2)
      )
{
      // Text vectorize function statements
      :
      return(errstat);
}
```

The return value of the text vectorize function must be zero if scan ok or nonzero on error.

C.4.2 Layout Editor Functions

The following **User Language** system functions are assigned to caller type GED; i.e., they can be called from the **Layout Editor** interpreter environment of the **Bartels AutoEngineer**:

ged_asklayer - GED layer selection (GED)

Synopsis

```
int ged_asklayer(             // Returns status
      & int;                  // Returns selected layer (LAY1|LAY9)
      int [0,7];              // Layer query type:
                              //    0 = Documentary layers and signal layers
                              //    1 = Signal layers
                              //    2 = Signal layers
                              //        (including Top Layer and All Layers)
                              //    3 = Documentary layers
                              //    4 = Signal and power layers
                              //    5 = arbitrary display element types
                              //    6 = Power layers
                              //    7 = Documentary, signal and power layers
      );
```

Description

The **ged_asklayer** function activates a **Layout Editor** layer selection menu. The layer query type designates the type of layers and/or display element types provided for selection. The function returns zero if a valid layer has been selected or (-1) if the layer selection was aborted.

ged_askrefname - GED reference name selection (GED)

Synopsis

```
int ged_askrefname(           // Returns status
      & string;               // Returns reference name
      & index L_CPART;        // Returns connection list part index
                              //    (on layout level only)
      int [0,2];              // Part selection mode:
                              //    0 = All parts
                              //    1 = Parts inside part group
                              //    2 = Parts outside part group
      int [0,1];              // Flag - unplaced part selection
      );
```

Description

The **ged_askrefname** function activates a dialog for selecting a reference, i.e., a part on layout level or a pin on part level. The part selection mode and the unplaced part flag can be used to restrict the list of selectable parts and/or pins. The functions returns zero if a reference was successfully selected or non-zero otherwise.

ged_asktreeidx - GED net selection (GED)

Synopsis

```
int ged_asktreeidx(           // Returns status
      & string;               // Returns tree name (on layout level only)
      & index L_CNET;         // Returns net index (on layout level only)
      int [0,5];              // Net selection mode:
                              //    0 = All trees, including No Net Assignment button
                              //    1 = Visible trees
                              //    2 = Invisible trees
                              //    3 = All trees
                              //    4 = Directly skip to net pick
                              //    5 = All trees, pattern input allowed
      );
```

Description

The **ged_asktreeidx** function activates a dialog for selecting a net. The net selection mode can be used to restrict the list of selectable nets. The functions returns zero if a net was successfully selected, 1 if a net delete operation (without net assignment) was performed, 2 if a net name pattern was specified, 3 if a net name was selected through net pick, or non-zero on invalid paramaters or if the selection was aborted.

ged_attachtextpos - Attach text position to layout element (GED)

Synopsis

```
int ged_attachtextpos(        // Returns status
      index L_FIGURE;         // Layout figure list element
      string;                 // Text string
      int;                    // Text layer (LAY1|LAY9)
      double;                 // Text X coordinate (STD2)
      double;                 // Text Y coordinate (STD2)
      double;                 // Text rotation angle (STD3)
      double;                 // Text size (STD2; negative for text base line)
      int [0,1];              // Text mirror mode (STD14)
      );
```

Description

The **ged_attachtextpos** function assigns a text position modifier with the specified properties for layer, position, rotation, size and mirroring to the text string of the specified layout figure list element. The function returns zero if the assignment was successful, (-1) for invalid parameters or (-2) if the layout element provides no text position modifier for the specified text string.

See also

Function **ged_storetext**.

ged_delelem - Delete GED figure list element (GED)

Synopsis

```
int ged_delelem(              // Returns status
      & index L_FIGURE;       // Element
      );
```

Description

The **ged_delelem** function deletes the given figure list element from the figure list. The function returns zero if the element was successfully deleted or nonzero on error.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

See also

Function **ged_drawelem**.

ged_drawelem - Redraw GED figure list element (GED)

Synopsis

```
void ged_drawelem(
      index L_FIGURE;          // Element
      int [0, 4];              // Drawing mode (STD19)
      );
```

Description

The **ged_drawelem** function updates the display of the given figure list element using the specified drawing mode.

See also

Function **ged_delelem**.

ged_drcerrorhide - Set/reset GED DRC error acceptance mode (GED)

Synopsis

```
int ged_drcerrorhide(          // Returns status
      string;                  // Error Id string
      int;                     // Error hide flag
      );
```

Description

The **ged_drcerrorhide** function sets the display mode for the DRC error specified by the error id. The function returns zero if the display mode was successfully set or non-zero on error.

ged_drcpath - GED trace test placement design rule check (GED)

Synopsis

```
int ged_drcpath(               // Returns status
      int [0,99];              // Path signal layer number (LAY1)
      double ]0.0,[;           // Path width (STD2)
      index L_LEVEL;           // Path connectivity level index
      int [0,3];               // Path connectivity checking mode:
                               //    0 = Show violations for all non-level elements
                               //    1 = Ignore violations against own tree
                               //    2 = Show violations against any tree
                               //    3 = Show violations against any tree beside
pick element
      );
```

Description

The **ged_drcpath** function performs a design rule check for a trace placement with the given parameters without actually placing the trace. The trace polygon points are taken from the internal polygon point list previously stored with **bae_storepoint**. The function returns zero if the trace can be successfully placed without DRC errors, a value greater or equal (1) if the trace placement would cause a DRC error or (-1) on missing/invalid environment/parameters.

See also

Functions **bae_storepoint**, **ged_storepath**.

ged_drcpoly - GED polygon test placement design rule check (GED)

Synopsis

```
int ged_drcpoly(              // Returns status
      int;                    // Polygon layer (LAY1)
      int [1,9];              // Polygon type (LAY4)
      string;                 // Polygon net name (for LAY4 types 4, 6 and 9)
      index L_LEVEL;          // Polygon connectivity level index
      int [0,3];              // Polygon connectivity checking mode:
                              //    0 = Show violations for all non-level elements
                              //    1 = Ignore violations against own tree
                              //    2 = Show violations against any tree
                              //    3 = Show violations against any tree beside
pick element
      );
```

Description

The **ged_drcpoly** function performs a design rule check for a polygon placement with the given parameters without actually placing the polygon. The polygon points are taken from the internal polygon point list previously stored with **bae_storepoint**. The function returns zero if the polygon can be successfully placed without DRC errors, a value greater or equal (1) if the polygon placement would cause a DRC error or (-1) on missing/invalid environment/parameters.

See also

Functions **bae_storepoint**, **ged_storepoly**.

ged_drcvia - GED via test placement design rule check (GED)

Synopsis

```
int ged_drcvia(               // Returns status
      string;                 // Via padstack library name
      double;                 // Via X coordinate (STD2)
      double;                 // Via Y coordinate (STD2)
      index L_LEVEL;          // Via connectivity level index
      int [0,3];              // Via connectivity checking mode:
                              //    0 = Show violations for all non-level elements
                              //    1 = Ignore violations against own tree
                              //    2 = Show violations against any tree
                              //    3 = Show violations against any tree beside
pick element
      );
```

Description

The **ged_drcvia** function performs a design rule check for a via placement with the given parameters without actually placing the via. The function returns zero if the via can be successfully placed without DRC errors, a value greater or equal (1) if the via placement would cause a DRC error, (-1) on missing/invalid environment/parameters or (-2) if the requested via padstack is not available.

See also

Function **ged_storeuref**.

ged_elemangchg - Change GED figure list element rotation angle (GED)

Synopsis

```
int ged_elemangchg(           // Returns status
      & index L_FIGURE;       // Element
      double;                 // New rotation angle (STD3)
      );
```

Description

The **ged_elemangchg** function changes the rotation angle of the given figure list element. The rotation angle must be in radians. The function returns zero if the element has been successfully rotated, (-1) if the figure list element is invalid or (-2) if the figure list element cannot be rotated.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ged_elemfixchg - Change GED figure list element fixed flag (GED)

Synopsis

```
int ged_elemfixchg(           // Returns status
      & index L_FIGURE;       // Element
      int [0,1];              // New fixed flag (STD11)
      );
```

Description

The **ged_elemfixchg** function changes the fixed flag of the given figure list element. The fixed flag value 0 unfixes the element, the fixed flag value 1 fixes the element. The function returns zero if the element fixed flag has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element cannot be fixed.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ged_elemgrpchg - Change GED figure list element group flag (GED)

Synopsis

```
int ged_elemgrpchg(           // Returns status
      index L_FIGURE;         // Element
      int [0,6];              // New group selection status (STD13|0x4)
      );
```

Description

The **ged_elemgrpchg** function changes the group flag of the given figure list element. Setting bit 3 (0x4) of the group status parameter activates a status line message about the selected/deselected element and the total number of group-selected elements. The function returns zero if the element group flag has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element cannot be selected to a group.

ged_elemlaychg - Change GED figure list element layer (GED)

Synopsis

```
int ged_elemlaychg(           // Returns status
      & index L_FIGURE;       // Element
      int;                    // New layer (LAY1)
      );
```

Description

The **ged_elemlaychg** function changes the layer of the given figure list element. The layer can be set for polygons, traces, texts, pads (on padstack level), and drill holes. For drill holes the layer input parameter specifies the drill class code. The function returns zero if the element layer has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element layer cannot be set.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ged_elemmirrchg - Change GED figure list element mirror mode (GED)

Synopsis

```
int ged_elemmirrchg(          // Returns status
      & index L_FIGURE;       // Element
      int [0,18];             // New mirror mode (STD14|LAY3)
      );
```

Description

The **ged_elemmirrchg** function changes the mirror mode of the given figure list element. The mirror mode can be set for polygons, texts and references. The function returns zero if the element mirror mode has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element mirror mode cannot be set.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ged_elemposchg - Change GED figure list element position (GED)

Synopsis

```
int ged_elemposchg(           // Returns status
      & index L_FIGURE;       // Element
      double;                 // New X coordinate (STD2)
      double;                 // New Y coordinate (STD2)
      );
```

Description

The **ged_elemposchg** function changes the position of the given figure list element. Polygons and/or traces are replaced to set the first point of the polygon/trace to the specified position. The function returns zero if the element has been successfully repositioned, (-1) if the figure list element is invalid or (-2) if the figure list element position cannot be set.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ged_elemsizechg - Change GED figure list element size (GED)

Synopsis

```
int ged_elemsizechg(          // Returns status
      & index L_FIGURE;       // Element
      double;                 // New size (STD2)
      );
```

Description

The **ged_elemsizechg** function changes the size of the given figure list element. The size can be changed for texts, drill holes, traces and areas. For traces, a trace width change is performed. For areas, the size parameter is interpreted as area expansion distance. The function returns zero if the element size has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element size cannot be set.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ged_getautocornins - Get GED auto corner insert mode (GED)

Synopsis

```
int ged_getautocornins(       // Returns mode:
                              //    0 = Auto Corner Insert disabled
                              //    1 = Auto Corner Insert Traces
                              //    2 = Auto Corner Insert Areas
                              //    3 = Auto Corner Insert Traces & Areas
      );
```

Description

The **ged_getautocornins** function returns the **Layout Editor** input mode for automatically inserting corners when generating traces and/or polygons. The auto corner insert mode is selected with either option Grid+Rotation octagonal or option Rotation octagonal from the Grids+Rotation function.

ged_getdblpar - Get GED double parameter (GED)

Synopsis

```
int ged_getdblpar(              // Returns status
      int [0,[;                 // Parameter type/number:
                                //    0 = Last group placement x coordinate (STD2)
                                //    1 = Last group placement y coordinate (STD2)
                                //    2 = Default part placement angle (STD3)
                                //    3 = Copper fill isolation distance (STD2)
                                //    4 = Copper fill min. area size (STD2)
                                //    5 = Copper fill heat trap width (STD2)
                                //    6 = Copper fill heat trap isolation (STD2)
                                //    7 = Hatch line spacing (STD2)
                                //    8 = Hatch line width (STD2)
                                //    9 = Hatch line angle (STD3)
                                //    10 = Net visibility dialog net name list
                                //          control element width
                                //    11 = Default text size (STD2)
                                //    12 = DRC distance violation text size (STD2)
                                //    13 = Autoplacement Part Expansion (STD2)
                                //    14 = Autoplacement Part Pin Factor [0, 1.0]
                                //    15 = Autoplacement Segment Fit [0, 1.0]
                                //    16 = Autoplacement Part Outline Offset (STD2)
                                //    17 = Default text placement angle (STD3)
                                //    18 = Autorouter Border to copper
                                //          LE distance (STD2)
                                //    19 = Autorouter Heat trap to drill
                                //          LE distance (STD2)
                                //    20 = Autorouter Isolation to drill
                                //          LE distance (STD2)
                                //    21 = Autorouter Power plane connection
                                //          LE run length (STD2)
                                //    22 = Autorouter Requested special
                                //          routing grid (STD2)
                                //    23 = Autorouter Split power plane
                                //          guard range (STD2)
                                //    24 = Autorouter BGA grid tolerance
                                //          distance (STD2)
                                //    25 = Autorouter SMD power plane connection
                                //          LE run length (STD2)
                                //    26 = Autorouter Pin to via distance (STD2)
                                //    27 = CAM Gerber standard line width (STD2)
                                //    28 = CAM Minimum distance
                                //          heat trap to drilling (STD2)
                                //    29 = CAM Minimum distance
                                //          isolation to drilling (STD2)
                                //    30 = CAM Heat trap to drilling
                                //          distance tolerance (STD2)
                                //    31 = CAM Isolation to drilling
                                //          distance tolerance (STD2)
                                //    32 = CAM Power layer border width (STD2)
                                //    33 = CAM Split power plane
                                //          isolation width (STD2)
                                //    34 = Bus trace width (STD2)
                                //    35 = Bus trace spacing (STD2)
      & double;                 // Returns parameter value
      );
```

Description

The **ged_getdblpar** function is used to query **Layout Editor** double parameters previously set with **ged_setdblpar**. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **ged_getintpar**, **ged_getstrpar**, **ged_setdblpar**, **ged_setintpar**, **ged_setstrpar**.

ged_getdrcmarkmode - Get GED DRC Error Display Mode (GED)

Synopsis

```
int ged_getdrcmarkmode(       // DRC error display mode
      );
```

Description

The **ged_getdrcmarkmode** function returns the currently selected **Layout Editor** DRC error display mode. The function returns zero for error color error marker display and 1 for highlight error marker display.

See also

Function **ged_setdrcmarkmode**.

ged_getdrcstatus - Get GED DRC Completion Status (GED)

Synopsis

```
int ged_getdrcstatus(         // DRC completion status
      );
```

Description

The **ged_getdrcstatus** function return the current **Layout Editor** design rule check completion status. The function returns zero if only the design changes of the current program session are checked, or nonzero if a full design rule check for complete DRC error display has been applied to the currently loaded element.

ged_getgroupdata - GED group placement data query (GED)

Synopsis

```
int ged_getgroupdata(         // Status
      & double;               // Group base X coordinate (STD2)
      & double;               // Group base Y coordinate (STD2)
      & double;               // Group rotation angle (STD3)
      & double;               // Group scale factor
      & int;                  // Group mirror mode
      & double;               // Group quadrant X coordinate (STD2)
      & double;               // Group quadrant Y coordinate (STD2)
      & int;                  // Group quadrant mode
      & int;                  // Group area mode
      );
```

Description

ged_getgroupdata function can be used to retrieve the current **Layout Editor** group placement interaction input data. The function returns nonzero if no group placement interaction is activated.

See also

Function **ged_getinputdata**.

ged_gethighlnet - Get GED net highlight mode/color (GED)

Synopsis

```
int ged_gethighlnet(          // Returns status
      int [-1,[;              // Net tree number or -1 for highlight focus modus
query
      & int;                  // Highlight mode
      & int;                  // Highlight color (bit 1 to 6, STD18) and pattern
(bit 7 to 12)
      );
```

Description

The **ged_gethighlnet** function can be used to get the highlight mode and the highlight color and pattern for the specified net. The highlight mode parameter is set to nonzero if the net highlight is activated or zero if the net highlight is deactivated. The second parameter returns the highlight color (bit 1 to 6) and the highlight pattern (bit 7 to 12). The function returns nonzero if the query was successful or zero on error (net not found, invalid parameters).

See also

Function **ged_highlnet**.

ged_getinputdata - GED input data query (GED)

Synopsis

```
int ged_getinputdata(         // Status
      & double;               // Initial X coordinate (STD2)
      & double;               // Initial Y coordinate (STD2)
      & double;               // Initial width (STD2)
      & int;                  // Initial layer (LAY1)
      & double;               // Current X coordinate (STD2)
      & double;               // Current Y coordinate (STD2)
      & double;               // Current width (STD2)
      & int;                  // Current layer (LAY1)
      & void;                 // Input mode/element (LAY11)
      & void;                 // Input path level index (optional)
      & double;               // Input first segment start X coordinate (STD2) */
      & double;               // Input first segment start Y coordinate (STD2) */
      & double;               // Input first arc center X coordinate (STD2) */
      & double;               // Input first arc center Y coordinate (STD2) */
      & int;                  // Input first arc center type (STD15) */
      & double;               // Input last segment start X coordinate (STD2) */
      & double;               // Input last segment start Y coordinate (STD2) */
      & double;               // Input last arc center X coordinate (STD2) */
      & double;               // Input last arc center Y coordinate (STD2) */
      & int;                  // Input last arc center type (STD15) */
      );
```

Description

The **ged_getinputdata** function can be used to retrieve the current **Layout Editor** placement interaction input data. The placement data has to be interpreted according to the input interaction type and/or placement element function parameter. The function returns nonzero if no placement interaction is activated.

See also

Function **ged_getgroupdata**.

ged_getintpar - Get GED integer parameter (GED)

Synopsis

```
int ged_getintpar(              // Returns status
      int [0,[;                 // Parameter type/number:
                                //    0 = Pick point display mode:
                                //        0 = No pick point display
                                //        1 = Pick point display
                                //        2 = Pick point wide display
                                //        3 = Pick point edit display
                                //    1 = Automatic DRC on layout load mode:
                                //        0 = no automatic DRC
                                //        1 = automatic DRC with verification
                                //        2 = automatic DRC without verification
                                //    2 = Top layer color code
                                //    3 = Info display flag:
                                //        0 = No automatic info display
                                //        1 = Automatic info display
                                //    4 = Info display mode:
                                //        0 = No info display
                                //        1 = Complete info display
                                //        2 = Copper info only display
                                //    5 = Angle edit direction
                                //    6 = Part level element DRC mode:
                                //        0 = Complete DRC
                                //        1 = Consider part macros checked
                                //    7 = Grid corner scan mode:
                                //        0 = No grid corner scan
                                //        1 = Complete grid corner scan
                                //        2 = Current window corner scan
                                //        3 = Dynamic window corner scan
                                //    8 = Mincon update mode
                                //    9 = DRC polygon sub-type exclude bits
                                //    10 = Warning output mode:
                                //        Bit 0: Supress SCM changed warnings
                                //        Bit 1: Supress copper fill problem
                                //                polygon group selection wanrnings
                                //        Bit 2: Supress variant mismatch warnings
                                //        Bit 3: Supress autorouter mode
                                //                termination warnings
                                //    11 = Layer usage scan mode
                                //    12 = Area polygon edit mode:
                                //        0 = don't close polylines
                                //        1 = always close polylines
                                //        2 = polyline close prompt
                                //    13 = DRC distance display pattern
                                //    14 = Trace edit pick mode:
                                //        0 = snap to input grid
                                //        1 = pin/trace snap at first trace corner
                                //    15 = Area mirror visibility mode:
                                //        0 = Normal area mirror visibility
                                //        1 = Disable area mirror visibility
                                //    16 = Trace net deletion query limit
                                //    17 = Plot preview mode:
                                //        0 = none
                                //        1 = plotter pen width
                                //    18 = DRC distance display mode:
                                //        0 = none
                                //        1 = trace distance line
                                //        2 = area distance line
                                //        3 = trace distance pattern
                                //        4 = area distance pattern
                                //    19 = Text layer mirroring mode:
                                //        0 = no layer mirroring
                                //        1 = documentary layer mirroring
                                //        2 = signal and documentary layer mirroring
                                //    20 = Default part mirroring mode
```

```
//    21 = Autosave interval
//    22 = Part airline display mode:
//         0 = No airlines
//         1 = Static airlines
//         2 = Dynamic airlines
//    23 = Angle lock toggle mode:
//         0 = Pick side default
//         1 = Grid toggle
//         2 = Shorter side toggle
//         3 = Edit direction
//    24 = Copper fill heat trap mode:
//         0 = Direct Connect
//         1 = Pin & Via Heat Traps
//         2 = Pin Heat Traps
//         3 = Via Heat Traps
//         |4 = No Neighbour Pins Flag
//         |8 = Heat Trap Trace Flag
//         |16 = Only Unconnected Layers
//    25 = Copper fill trace mode:
//         0 = Round Corners
//         1 = Octagonal Corners
//         2 = Octagonal Circles
//         3 = Octagonal Corners & Circles
//    26 = Copper fill island mode:
//         0 = Keep Islands
//         1 = Delete Islands
//         2 = Select Islands
//    27 = Copper fill inside area mode:
//         0 = Inner Fill Area Fill
//         1 = Inner Fill Area Keepout
//         |2 = Keepout Areas without Distance
//    28 = Copper fill max. heat trap junctions
//    29 = Copper fill acute angle mode:
//         0 = Acute Angles flat
//         1 = Acute Angles round
//    30 = Copper fill hatch mode:
//         0 = Line Hatching
//         1 = Grid Hatching
//         |2 = Create Editable Paths
//    31 = Net visibility dialog box mode:
//         0 = Single column net name list display
//         1 = Multi-column net name list display
//    32 = Group move display mode:
//         0 = Moving Picture Off
//         1 = Display Group Layer Only
//         2 = Moving Picture On
//         3 = Moving Picture All
//    33 = Group trace selection mode:
//         0 = Select Traces & Vias
//         1 = Select Traces Only
//         2 = Select Vias Only
//    34 = Pick preference layer selection (LAY1)
//    35 = Clipboard text placement request flag
//    36 = Edit direction
//    37 = Mincon Area Mode (Bit Patterns):
//         0 = No Area Mincon
//         |1 = Copper Area Mincon
//         |2 = Connected Copper Area Mincon
//    38 = Group angle lock mode:
//         0 = Keep group angle lock
//         1 = Automatically release group angle lock
//    39 = Autoplacement Optimizer Passes
//    40 = Autoplacement Part Swap On/Off Flag
//    41 = Autoplacement Pin/Gate Swap On/Off Flag
```

```
                                //    42 = Autoplacement Mirroring Mode:
                                //         0 = No SMD mirroring
                                //         1 = SMD Mirroring
                                //         2 = SMD 2-Pin Mirroring
                                //         3 = Only SMD Mirroring
                                //    43 = Autoplacement Rotation Mode:
                                //         0 = 0-90 Degree Rotation
                                //         1 = 0-270 Degree Rotation
                                //         2 = 0 Degree Rotation
                                //         3 = 90 Degree Rotation
                                //         4 = 0 XOR 90 Degree Rotation
                                //    44 = Autoplacement Retry Passes
                                //    45 = Autoplacement SMD Rotation Mode:
                                //         0 = 0-90 Degree Rotation
                                //         1 = 0-270 Degree Rotation
                                //         2 = 0 Degree Rotation
                                //         3 = 90 Degree Rotation
                                //         4 = 0 XOR 90 Degree Rotation
                                //    46 = Autoplacement Part Outline Layer (LAY1)
                                //    47 = Group visibility mode:
                                //         0 = Select all elements
                                //         1 = Select visible elements only
                                //    48 = Default text mirror mode and text
                                //           mode (STD14|LAY14)
                                //    49 = Autorouter Number of optimization runs
                                //    50 = Autorouter Optimizer characteristic
                                //    51 = Autorouter Max. number of
                                //           vias per connection
                                //    52 = Autorouter Router via delay at 1/10"
                                //    53 = Autorouter Router pin channel delay
                                //    54 = Autorouter Cross direction delay
                                //    55 = Autorouter Direction change delay
                                //    56 = Autorouter Path packing delay
                                //    57 = Autorouter Statistical delay base
                                //    58 = Autorouter Max. rip-ups per con.
                                //    59 = Autorouter Max. rip-up level
                                //    60 = Autorouter Max. number of rip-up retries
                                //    61 = Autorouter Router via grid index
                                //    62 = Autorouter Bus structure delay
                                //    63 = Autorouter Re-route area 1 delay
                                //    64 = Autorouter Re-route area 2 delay
                                //    65 = Autorouter Skip existing path delay
                                //    66 = Autorouter Router cleaning run enable
                                //    67 = Autorouter Optim. cleaning run enable
                                //    68 = Autorouter Power connection
                                //           vector unroutes
                                //    69 = Autorouter Automatic save enable
                                //    70 = Autorouter Corner connection
                                //           output enable
                                //    71 = Autorouter Unroute output sort mode
                                //    72 = Autorouter Corner mitring mode
                                //    73 = Autorouter Existing traces
                                //           orientation mode
                                //    74 = Autorouter Standard connection
                                //           layer delay
                                //    75 = Autorouter Bus connection layer delay
                                //    76 = Autorouter Wave limitation offset
                                //    77 = Autorouter Gridless via check mode
                                //    78 = Autorouter Input error checking mode
                                //    79 = Autorouter Trace to pin entry mode
                                //    80 = Autorouter Requested subgrid factor
                                //    81 = Autorouter Router off-grid delay
                                //    82 = Autorouter Bus recognition and
                                //           routing mode
                                //    83 = Autorouter SMD pin-via pass enable
                                //    84 = Autorouter Pin/gate swap mode
                                //    85 = Autorouter Requested gridless
                                //           routing mode
                                //    86 = Autorouter Incremental output mode
                                //    87 = Autorouter Router prefered grid shift
                                //    88 = Autorouter Router prefered grid delay
```

```
//    89 = Autorouter Outside net area delay
//    90 = Autorouter Last optimization
//         par. change mode
//    91 = Autorouter Auto rip-up parameter mode
//    92 = Autorouter Prefered routing
//         direction mode
//    93 = Autorouter Optimizer order mode
//    94 = Autorouter Via rip-up flag
//    95 = Autorouter Routing window border size
//    96 = Autorouter BGA fan out enable flag
//    97 = Autorouter Fan out gridded check mode
//    98 = Autorouter Alternate via shift mode
//    99 = Autorouter Full via evaluation mode
//    100 = Autorouter Micro via mode
//    101 = Autorouter Forced dir. max. derivation
//    102 = Autorouter Routing frame window flag
//    103 = Autorouter Requested pad entry subgrid
//    104 = Autorouter Power layer via mode
//    105 = Autorouter Via check mode
//    106 = Autorouter Large net connection count
//    107 = Autorouter Autorouting active flag
//    108 = CAM Heat trap base angle
//    109 = Single corner edit flag
//    110 = Resize with round corners flag
//    111 = Hidden DRC errors display flag
//    112 = DRC violation elements scan flag
//    113 = Part placement trace move mode
//        0 = No trace move
//        1 = Trace end move
//        2 = Trace segment move
//    114 = Polygon edit autocomplete flag
//    115 = Board Outline alternative
//          documentary layer (LAY1)
//    116 = Trace join query mode:
//        0 = Never join traces
//        1 = Always join traces
//        2 = Query for trace join
//    117 = Trace display class bits (LAY15)
//    118 = Text display class bits (LAY15)
//    119 = Copper polygon
//          display class bits (LAY15)
//    120 = Forbidden area polygon
//          display class bits (LAY15)
//    121 = Border polygon
//          display class bits (LAY15)
//    122 = Connected copper polygon
//          display class bits (LAY15)
//    123 = Documentary line
//          display class bits (LAY15)
//    124 = Documentary area
//          display class bits (LAY15)
//    125 = Copper fill with cutout polygon
//          display class bits (LAY15)
//    126 = Hatched copper polygon
//          display class bits (LAY15)
//    127 = Split power plane polygon
//          display class bits (LAY15)
//    128 = Flag - Color table saved
//    129 = Airline color mode:
//        0 = Use unroutes color
//        1 = Use layer color
//    130 = Airline clipping mode:
//        0 = No unroutes clipping
//        1 = Clip unroutes without workspace target
```

```
                              //    131 = Trace collision mode:
                              //        -1 = Query for operation
                              //         0 = Ignore collisions
                              //         1 = Delete colliding traces
                              //         2 = Delete colliding segments
                              //         3 = Cut colliding segments
                              //    132 = Layout trace merge query mode:
                              //         0 = Never merge layout traces
                              //         1 = Always merge layout traces
                              //         2 = Query merge mode
                              //    133 = Part trace merge query mode:
                              //         0 = Never merge part traces
                              //         1 = Always merge part traces
                              //         2 = Query merge mode
                              //    135 = Element move polygon display mode:
                              //         0 = Display outline
                              //         1 = Display filled
                              //    136 = Group move airline display mode:
                              //         0 = Airline Display Off
                              //         1 = Display Group Part Pin Airlines
                              //    137 = Trace collision distance check mode:
                              //         0 = Use DRC distance for collision check
                              //         1 = Consider only crossings as collision
                              //    138 = Trace segment bundle pick mode:
                              //         0 = Continuous segment pick
                              //         1 = Pick first and last bundle segment
                              //    139 = Trace segment insert pick mode:
                              //         0 = 3 click selection
                              //         1 = 2 click selection
                              //    140 = Part edit DRC:
                              //         0 = no part edit online DRC
                              //         1 = part edit online DRC
                              //    141 = Drill tool table optimization flag
                              //    142 = Bus trace count
                              //    143 = Edit bus trace count
                              //    144 = Bus trace creation mode:
                              //         0 = Create trace bundle
                              //         1 = Create seperate traces
                              //    145 = Bus trace corner mode:
                              //         0 = Create angle corners
                              //         1 = Create arc corners
                              //    146 = Silk screen layer (LAY1)
                              //    147 = Macro outline display mode:
                              //         0 = No macro outline display
                              //         1 = Display macro outline
                              //              at moved references
                              //         2 = Display macro outlines
    & int;                    // Returns parameter value
    );
```

Description

The **ged_getintpar** function is used to query **Layout Editor** integer parameters previously set with **ged_setintpar**. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **ged_getdblpar**, **ged_setintpar**, **ged_getstrpar**, **ged_setdblpar**. **ged_setstrpar**.

ged_getlaydefmode - Get GED default layer mode (GED)

Synopsis

```
int ged_getlaydefmode(        // Default layer mode:
                              //    0 = automatic layer default disabled
                              //    1 = used edit layer as layer default
                              //    2 = last used layer as layer default
      );
```

Description

The **ged_getlaydefmode** function returns the current **Layout Editor** default layer mode.

See also

Functions **ged_getlayerdefault**, **ged_setlaydefmode**, **ged_setlayerdefault**.

ged_getlayerdefault - Get GED default layer (GED)

Synopsis

```
int ged_getlayerdefault(      // Layer (LAY1)
      );
```

Description

The **ged_getlayerdefault** function returns the current **Layout Editor** default layer.

See also

Functions **ged_getlaydefmode**, **ged_setlaydefmode**, **ged_setlayerdefault**.

ged_getmincon - Get GED Mincon function type (GED)

Synopsis

```
int ged_getmincon(            // Returns Mincon function type (LAY10)
      );
```

Description

The **ged_getmincon** function returns the currently selected **Layout Editor** Mincon function type, i.e., the airline display mode (LAY10).

ged_getpathwidth - Get GED path standard widths (GED)

Synopsis

```
void ged_getpathwidth(
      & double;               // Returns small standard width (STD2)
      & double;               // Returns wide standard width (STD2)
      );
```

Description

The **ged_getpathwidth** function parameters return the **Layout Editor** standard widths for small and wide traces.

ged_getpickmode - Get GED element pick mode (GED)

Synopsis

```
int ged_getpickmode(          // Element pick mode:
                              //    0 = Pick preference layer pick
                              //    1 = Pick with element selection
                              //    2 = Exclusive pick preference layer pick
      );
```

Description

ged_getpickmode function returns the currently selected **Layout Editor** element pick mode.

See also

Funktion **ged_setpickmode**.

ged_getpickpreflay - Get GED pick preference layer (GED)

Synopsis

```
int ged_getpickpreflay(       // Returns pick preference layer (LAY1)
      );
```

Description

The **ged_getpickpreflay** function returns the currently active **Layout Editor** pick preference layer for element selection (LAY1).

ged_getpowlayerrcnt - Get GED power layer error count (GED)

Synopsis

```
int ged_getpowlayerrcnt(      // Powe layer error count
      );
```

Description

The **ged_getpowlayerrcnt** function returns the current **Layout Editor** power layer error count.

ged_getsegmovmode - Get GED trace segment move mode (GED)

Synopsis

```
int ged_getsegmovmode(        // Returns trace segment move mode:
                              //    0 = Move without neighbours
                              //    1 = Move with neighbours
                              //    2 = Adjust neighbours
                              //    3 = Adjust neighbours without vias
                              //    8 = Adjust next neighbours only
                              //    |4 = Open trace ends follow segment movement
      );
```

Description

The **ged_getsegmovmode** function returns the current **Layout Editor** trace segment move mode.

See also
Function **ged_setsegmovmode**.

ged_getstrpar - Get GED string parameter (GED)

Synopsis

```
int ged_getstrpar(            // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = Last placed named reference name
                              //    1 = Last placed named reference macro name
                              //    2 = Last placed text string
                              //    3 = Default library name
                              //    4 = Next free name
                              //    5 = Drill naming base
                              //    6 = Drill part macro name pattern
                              //    7 = Drill padstack macro name pattern
                              //    8 = Input prompt override string
                              //    9 = Last placed macro library
                              //    10 = Autosave path name
      & string;               // Returns parameter value
      );
```

Description

The **ged_getstrpar** function is used to query **Layout Editor** string parameter settings. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **ged_getdblpar**, **ged_getintpar**, **ged_setdblpar**, **ged_setintpar**, **ged_setstrpar**.

ged_getviaoptmode - Get GED trace via optimization mode (GED)

Synopsis

```
int ged_getviaoptmode(        // Returns trace via optimization mode:
                              //    0 = Via optimization
                              //    1 = Keep vias
      );
```

Description

The **ged_getviaoptmode** function returns the current **Layout Editor** trace via optimization mode.

See also

Function **ged_setviaoptmode**.

ged_getwidedraw - Get GED wide line display start width (GED)

Synopsis

```
double ged_getwidedraw(       // Returns width value (STD2)
      );
```

Description

The **ged_getwidedraw** function returns the current **Layout Editor** wide line display start width, i.e., the minimum trace width for displaying traces like filled polygons.

ged_groupselect - GED group selection (GED)

Synopsis

```
int ged_groupselect(          // Number of changes or (-1) on error
      int [0,9];              // Element selection type:
                              //    0 = select by element type
                              //    1 = select by element layer
                              //    2 = select by element fixed flag
                              //    3 = select by element visibility
                              //    4 = select by element not on layer
                              //    5 = select by element tree/net number
                              //    6 = select by element negated tree/net number
                              //    7 = select elements connected to tree number
                              //    8 = select elements not connected to tree
                              //    9 = select by element polygon type
                              //    10 = select by element glued mode
      int;                    // Element selection value according to type:
                              //    0 - element type (0|LAY6)
                              //    1,4 - element layer (LAY1)
                              //    2 - element fixed flag (STD11)
                              //    3 - element visible flag (0|1)
                              //    5,6 - element tree/net number
                              //    7, 8 - tree/net number
                              //    9 - element polygon type (LAY4)
                              //    10 - element glued mode (STD11 | STD12)
      int [0,2];              // New group selection status (STD13)
      );
```

Description

The **ged_groupselect** function changes the group flag of all elements of the specified type and/or value. The function returns the number of elements (de)selected or (-1) on error (i.e., on invalid and/or incompatible parameter specifications). Element selection value zero for element type selection is used for selecting elements of *any* type.

Warning

Internal layout element types such as the standard via definition(s) are excluded from group (de)selections with **ged_groupselect** to prevent from unintentionally modifying and/or deleting such elements and/or definitions when subsequently using other group functions.

ged_highlnet - Set GED net highlight mode/color (GED)

Synopsis

```
int ged_highlnet(             // Returns status
      int [-1,[;              // Net tree number
      int [0,[;               // Highlight off/on flag || (color/patterndef << 1)
      );
```

Description

The **ged_highlnet** function sets the highlight mode of the net specified by the given net tree number. The least significant bit of the highlight parameter designates whether the net should be highlighted (value 1) or not (value 0). The other bits the highlight parameter can be used to specify a highlight color code (bit 2 to 6) and/or a highlight display pattern (bit 7 to 12). The function returns nonzero if an invalid net tree number and/or highlight mode value has been specified.

See also
Function **ged_gethighlnet**.

ged_layergrpchg - Select GED group by layer (GED)

Synopsis

```
int ged_layergrpchg(          // Number of elements
      int [0,[;               // Layer number (LAY1)
      int [0,1];              // New group selection status (STD13)
      );
```

Description

The **ged_layergrpchg** function changes the group flag of all elements placed on the specified layer. The function returns the number of elements (de)selected or (-1) on error.

ged_partaltmacro - Change GED net list part package type (GED)

Synopsis

```
int ged_partaltmacro(         // Returns status
      string;                 // Part name
      string;                 // New part package type name
      );
```

Description

The **ged_partaltmacro** function changes the package type of the given net list part. The function returns nonzero if the part package type has been successfully changed, (-1) for invalid input parameters, (-2) if the specified package does not contain all pins referenced by the part in the net list (package is changed anyway), (-3) if the specified part does not exist in the net list, (-4) if the new package type isn't allowed for this part, (-5) if the new package couldn't be loaded, (-6) if the new package couldn't be copied to the job file or (-7) for multiple package change requests (e.g., **a** to **b** and then **b** to **c**) in one program run.

Warning

It is strongly recommended not to use this function in **L_CPART** index loops since the current **L_CPART** index variables are invalid after calling **ged_partaltmacro**.

ged_partnamechg - Change GED part name (GED)

Synopsis

```
int ged_partnamechg(          // Returns status
      string;                 // Old name
      string;                 // New name
      );
```

Description

The **ged_partnamechg** function changes the name of part. The function returns nonzero if the part name has been successfully changed, (-1) for invalid input parameters, (-2) if the specified part is not yet placed, (-4) if the new name exists already or (-5) for multiple name change requests (e.g., **a** to **b** and then **b** to **c**) in one program run. On layout part macro level, **ged_partnamechg** can be used for the renaming of pins.

Warning

This function might change the net list in which case a Backannotation is subsequently required. It is strongly recommended not to use this function in **L_CPART** index loops since the current **L_CPART** index variables are invalid after calling **ged_partnamechg**.

ged_pickanyelem - Pick any GED figure list element (GED)

Synopsis

```
int ged_pickanyelem(          // Returns status
      & index L_FIGURE;       // Returns picked element
      int;                    // Pick element type set ((LAY6 except 7)<<1 or'ed)
      );
```

Description

The **ged_pickanyelem** function activates a mouse interaction for selecting a figure list element from the specified pick element type set. The picked figure list element index is returned with the first parameter. The function returns zero if an element has been picked or (-1) if no element was found at the pick position.

See also

Function **ged_pickelem**.

ged_pickelem - Pick GED figure list element (GED)

Synopsis

```
int ged_pickelem(             // Returns status
      & index L_FIGURE;       // Returns picked element
      int [1,10];             // Pick element type (LAY6 except 7)
      );
```

Description

The **ged_pickelem** function activates an interactive figure list element pick request (with mouse). The required pick element type is specified with the second parameter. The picked figure list element index is returned with the first parameter. The function returns zero if an element has been picked or (-1) if no element of the required type has been found at the pick position.

See also

Functions **ged_pickanyelem**, **ged_setpickelem**.

ged_setautocornrins - Set GED auto corner insert mode (GED)

Synopsis

```
int ged_setautocornins(       // Returns status
      int [0,3];              // Auto corner insert mode:
                              //    0 = Auto Corner Insert disabled
                              //    1 = Auto Corner Insert Traces
                              //    2 = Auto Corner Insert Areas
                              //    3 = Auto Corner Insert Traces & Areas
      );
```

Description

The **ged_setautocornins** function sets the **Layout Editor** input mode for automatically inserting corners when generating traces and/or polygons. Usually, the auto corner insert mode is selected with either option Grid+Rotation octagonal or option Rotation octagonal from the Grids+Rotation function. The function returs nonzero if an invalid octagon input mode has been specified.

ged_setdblpar - Set GED double parameter (GED)

Synopsis

```
int ged_setdblpar(              // Returns status
      int [0,[;                 // Parameter type/number:
                                //    0 = Last group placement x coordinate (STD2)
                                //    1 = Last group placement y coordinate (STD2)
                                //    2 = Default part placement angle
                                //    3 = Copper fill isolation distance (STD2)
                                //    4 = Copper fill min. area size (STD2)
                                //    5 = Copper fill heat trap width (STD2)
                                //    6 = Copper fill heat trap isolation (STD2)
                                //    7 = Hatch line spacing (STD2)
                                //    8 = Hatch line width (STD2)
                                //    9 = Hatch line angle (STD3)
                                //    10 = Net visibility dialog net name list
                                //          control element width
                                //    11 = Default text size (STD2)
                                //    12 = DRC distance violation text size (STD2)
                                //    13 = Autoplacement Part Expansion (STD2)
                                //    14 = Autoplacement Part Pin Factor [0, 1.0]
                                //    15 = Autoplacement Segment Fit [0, 1.0]
                                //    16 = Autoplacement Part Outline Offset (STD2)
                                //    17 = Default text placement angle (STD3)
                                //    18 = Autorouter Border to copper
                                //          LE distance (STD2)
                                //    19 = Autorouter Heat trap to drill
                                //          LE distance (STD2)
                                //    20 = Autorouter Isolation to drill
                                //          LE distance (STD2)
                                //    21 = Autorouter Power plane connection
                                //          LE run length (STD2)
                                //    22 = Autorouter Requested special
                                //          routing grid (STD2)
                                //    23 = Autorouter Split power plane
                                //          guard range (STD2)
                                //    24 = Autorouter BGA grid tolerance
                                //          distance (STD2)
                                //    25 = Autorouter SMD power plane connection
                                //          LE run length (STD2)
                                //    26 = Autorouter Pin to via distance (STD2)
                                //    27 = CAM Gerber standard line width (STD2)
                                //    28 = CAM Minimum distance
                                //          heat trap to drilling (STD2)
                                //    29 = CAM Minimum distance
                                //          isolation to drilling (STD2)
                                //    30 = CAM Heat trap to drilling
                                //          distance tolerance (STD2)
                                //    31 = CAM Isolation to drilling
                                //          istance tolerance (STD2)
                                //    32 = CAM Power layer border width (STD2)
                                //    33 = CAM Split power plane
                                //          isolation width (STD2)
                                //    34 = Bus trace width (STD2)
                                //    35 = Bus trace spacing (STD2)
      double;                   // Parameter value
      );
```

Description

The **ged_setdblpar** function is used to set **Layout Editor** double system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **ged_getdblpar** function can be used to query parameter values set with **ged_setdblpar**.

See also

Functions **ged_getdblpar**, **ged_getintpar**, **ged_getstrpar**, **ged_setintpar**, **ged_setstrpar**.

ged_setdrcmarkmode - Get GED DRC error display mode (GED)

Synopsis

```
int ged_setdrcmarkmode(       // Returns status
      int [0,1];              // DRC error display mode:
                              //    0 = Error color error marker display
                              //    1 = Highlight color error marker display
      );
```

Description

The **ged_setdrcmarkmode** function sets the DRC error marker display mode. The functions returns nonzero for invalid display mode specifications.

See also

Function **ged_getdrcmarkmode**.

ged_setintpar - Set GED integer parameter (GED)

Synopsis

```
int ged_setintpar(            // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = Pickpunktanzeigemodus:
                              //        0 = keine Pickpunktanzeige
                              //        1 = Pickpunktanzeige
                              //        2 = Pick point wide display
                              //        3 = Pick point edit display
                              //    1 = Automatic DRC on layout load mode:
                              //        0 = no automatic DRC
                              //        1 = automatic DRC with verification
                              //        2 = automatic DRC without verification
                              //  [ 2 = use bae_setcolor instead ]
                              //    3 = Info display flag:
                              //        0 = No automatic info display
                              //        1 = Automatic info display
                              //    4 = Info display mode:
                              //        0 = No info display
                              //        1 = Complete info display
                              //        2 = Copper info only display
                              //    5 = Angle edit direction
                              //    6 = Part level element DRC mode:
                              //        0 = Complete DRC
                              //        1 = Consider part macros checked
                              //    7 = Grid corner scan mode:
                              //        0 = No grid corner scan
                              //        1 = Complete grid corner scan
                              //        2 = Current window corner scan
                              //        3 = Dynamic window corner scan
                              //             updated window
                              //    8 = Mincon update mode
                              //    9 = DRC polygon sub-type exclude bits
                              //    10 = Warning output mode:
                              //        Bit 0: Supress SCM changed warnings
                              //        Bit 1: Supress copper fill problem polygon
                              //                group selection wanrnings
                              //        Bit 2: Supress variant mismatch warnings
                              //        Bit 3: Supress autorouter mode
                              //                termination warnings
                              //    11 = Layer usage scan mode
                              //    12 = Area polygon edit mode:
                              //        0 = don't close polylines
                              //        1 = always close polylines
                              //        2 = polyline close prompt
                              //    13 = DRC distance display pattern
                              //    14 = Trace edit pick mode:
                              //        0 = snap to input grid
                              //        1 = pin/trace snap at first trace corner
                              //  [ 15 = System parameter - no write access ]
                              //    16 = Trace net deletion query limit
```

```
//    17 = Plot preview mode:
//         0 = none
//         1 = plotter pen width
//    18 = DRC distance display mode:
//         0 = none
//         1 = trace distance line
//         2 = area distance line
//         3 = trace distance pattern
//         4 = area distance pattern
//    19 = Text layer mirroring mode:
//         0 = no layer mirroring
//         1 = documentary layer mirroring
//         2 = signal and documentary layer mirroring
//    20 = Default part mirroring mode
//    21 = Autosave interval
//    22 = Part airline display mode:
//         0 = No airlines
//         1 = Static airlines
//         2 = Dynamic airlines
//    23 = Angle lock toggle mode:
//         0 = Pick side default
//         1 = Grid toggle
//         2 = Shorter side toggle
//         3 = Edit direction
//    24 = Copper fill heat trap mode:
//         0 = Direct Connect
//         1 = Pin & Via Heat Traps
//         2 = Pin Heat Traps
//         3 = Via Heat Traps
//         |4 = No Neighbour Pins Flag
//         |8 = Heat Trap Trace Flag
//         |16 = Only Unconnected Layers
//    25 = Copper fill trace mode:
//         0 = Round Corners
//         1 = Octagonal Corners
//         2 = Octagonal Circles
//         3 = Octagonal Corners & Circles
//    26 = Copper fill island mode:
//         0 = Keep Islands
//         1 = Delete Islands
//         2 = Select Islands
//    27 = Copper fill inside area mode:
//         0 = Inner Fill Area Fill
//         1 = Inner Fill Area Keepout
//         |2 = Keepout Areas without Distance
//    28 = Copper fill max. heat trap junctions
//    29 = Copper fill acute angle mode:
//         0 = Acute Angles flat
//         1 = Acute Angles round
//    30 = Copper fill hatch mode:
//         0 = Line Hatching
//         1 = Grid Hatching
//         |2 = Create Editable Paths
//    31 = Net visibility dialog box mode:
//         0 = Single column net name list display
//         1 = Multi-column net name list display
//    32 = Group move display mode:
//         0 = Moving Picture Off
//         1 = Display Group Layer Only
//         2 = Moving Picture On
//         3 = Moving Picture All
//    33 = Group trace selection mode:
//         0 = Select Traces & Vias
//         1 = Select Traces Only
//         2 = Select Vias Only
//    34 = Pick prefered layer (LAY1)
//           without actions
//    35 = Clipboard text placement request flag
```

```
//    36 = Edit direction
//    37 = Mincon Area Mode (Bit Patterns):
//         0 = No Area Mincon
//         |1 = Copper Area Mincon
//         |2 = Connected Copper Area Mincon
//    38 = Group angle lock mode:
//         0 = Keep group angle lock
//         1 = Automatically release group angle lock
//    39 = Autoplacement Optimizer Passes
//    40 = Autoplacement Part Swap On/Off Flag
//    41 = Autoplacement Pin/Gate Swap On/Off Flag
//    42 = Autoplacement Mirroring Mode:
//         0 = No SMD mirroring
//         1 = SMD Mirroring
//         2 = SMD 2-Pin Mirroring
//         3 = Only SMD Mirroring
//    43 = Autoplacement Rotation Mode:
//         0 = 0-90 Degree Rotation
//         1 = 0-270 Degree Rotation
//         2 = 0 Degree Rotation
//         3 = 90 Degree Rotation
//         4 = 0 XOR 90 Degree Rotation
//    44 = Autoplacement Retry Passes
//    45 = Autoplacement SMD Rotation Mode:
//         0 = 0-90 Degree Rotation
//         1 = 0-270 Degree Rotation
//         2 = 0 Degree Rotation
//         3 = 90 Degree Rotation
//         4 = 0 XOR 90 Degree Rotation
//    46 = Autoplacement Part Outline Layer (LAY1)
//    47 = Group visibility mode:
//         0 = Select all elements
//         1 = Select visible elements only
//    48 = Default text mirror mode and
//           text mode (STD14|LAY14)
//    49 = Autorouter Number of optimization runs
//    50 = Autorouter Optimizer characteristic
//    51 = Autorouter Max. number of
//           vias per connection
//    52 = Autorouter Router via delay at 1/10"
//    53 = Autorouter Router pin channel delay
//    54 = Autorouter Cross direction delay
//    55 = Autorouter Direction change delay
//    56 = Autorouter Path packing delay
//    57 = Autorouter Statistical delay base
//    58 = Autorouter Max. rip-ups per con.
//    59 = Autorouter Max. rip-up level
//    60 = Autorouter Max. number of rip-up retries
//    61 = Autorouter Router via grid index
//    62 = Autorouter Bus structure delay
//    63 = Autorouter Re-route area 1 delay
//    64 = Autorouter Re-route area 2 delay
//    65 = Autorouter Skip existing path delay
//    66 = Autorouter Router cleaning run enable
//    67 = Autorouter Optim. cleaning run enable
//    68 = Autorouter Power connection
//           vector unroutes
//    69 = Autorouter Automatic save enable
//    70 = Autorouter Corner connection
//           output enable
//    71 = Autorouter Unroute output sort mode
//    72 = Autorouter Corner mitring mode
//    73 = Autorouter Existing traces
//           orientation mode
//    74 = Autorouter Standard connection
//           layer delay
//    75 = Autorouter Bus connection layer delay
//    76 = Autorouter Wave limitation offset
//    77 = Autorouter Gridless via check mode
//    78 = Autorouter Input error checking mode
//    79 = Autorouter Trace to pin entry mode
```

```
//    80 = Autorouter Requested subgrid factor
//    81 = Autorouter Router off-grid delay
//    82 = Autorouter Bus recognition and
//          routing mode
//    83 = Autorouter SMD pin-via pass enable
//    84 = Autorouter Pin/gate swap mode
//    85 = Autorouter Requested gridless
//          routing mode
//    86 = Autorouter Incremental output mode
//    87 = Autorouter Router prefered grid shift
//    88 = Autorouter Router prefered grid delay
//    89 = Autorouter Outside net area delay
//    90 = Autorouter Last optimization
//          par. change mode
//    91 = Autorouter Auto rip-up parameter mode
//    92 = Autorouter Prefered routing
//          direction mode
//    93 = Autorouter Optimizer order mode
//    94 = Autorouter Via rip-up flag
//    95 = Autorouter Routing window border size
//    96 = Autorouter BGA fan out enable flag
//    97 = Autorouter Fan out gridded check mode
//    98 = Autorouter Alternate via shift mode
//    99 = Autorouter Full via evaluation mode
//    100 = Autorouter Micro via mode
//    101 = Autorouter Forced dir. max. derivation
//    102 = Autorouter Routing frame window flag
//    103 = Autorouter Requested pad entry subgrid
//    104 = Autorouter Power layer via mode
//    105 = Autorouter Via check mode
//    106 = Autorouter Large net connection count
//    107 = Autorouter Autorouting active flag
//    108 = CAM Heat trap base angle
//    109 = Single corner edit flag
//    110 = Resize with round corners flag
//    111 = Hidden DRC errors display flag
//    112 = DRC violation elements scan flag
//    113 = Part placement trace move mode
//        0 = No trace move
//        1 = Trace end move
//        2 = Trace segment move
//    114 = Polygon edit autocomplete flag
//    115 = Board Outline alternative
//          documentary layer (LAY1)
//    116 = Trace join query mode:
//        0 = Never join traces
//        1 = Always join traces
//        2 = Query for trace join
//    117 = Trace display class bits (LAY15)
//    118 = Text display class bits (LAY15)
//    119 = Copper polygon
//          display class bits (LAY15)
//    120 = Forbidden area polygon
//          display class bits (LAY15)
//    121 = Border polygon
//          display class bits (LAY15)
//    122 = Connected copper polygon
//          display class bits (LAY15)
//    123 = Documentary line
//          display class bits (LAY15)
//    124 = Documentary area
//          display class bits (LAY15)
//    125 = Copper fill with cutout polygon
//          display class bits (LAY15)
//    126 = Hatched copper polygon
//          display class bits (LAY15)
```

```
                              //    127 = Split power plane polygon
                              //          display class bits (LAY15)
                              //    128 = Flag - Color table saved
                              //    129 = Airline color mode:
                              //        0 = Use unroutes color
                              //        1 = Use layer color
                              //    130 = Airline clipping mode:
                              //        0 = No unroutes clipping
                              //        1 = Clip unroutes without workspace target
                              //    131 = Trace collision mode:
                              //        -1 = Query for operation
                              //        0 = Ignore collisions
                              //        1 = Delete colliding traces
                              //        2 = Delete colliding segments
                              //        3 = Cut colliding segments
                              //    132 = Layout trace merge query mode:
                              //        0 = Never merge layout traces
                              //        1 = Always merge layout traces
                              //        2 = Query merge mode
                              //    133 = Part trace merge query mode:
                              //        0 = Never merge part traces
                              //        1 = Always merge part traces
                              //        2 = Query merge mode
                              //    135 = Element move polygon display mode:
                              //        0 = Display outline
                              //        1 = Display filled
                              //    136 = Group move airline display mode:
                              //        0 = Airline Display Off
                              //        1 = Display Group Part Pin Airlines
                              //    137 = Trace collision distance check mode:
                              //        0 = Use DRC distance for collision check
                              //        1 = Consider only crossings as collision
                              //    138 = Trace segment bundle pick mode:
                              //        0 = Continuous segment pick
                              //        1 = Pick first and last bundle segment
                              //    139 = Trace segment insert pick mode:
                              //        0 = 3 click selection
                              //        1 = 2 click selection
                              //    140 = Part edit DRC:
                              //        0 = no part edit online DRC
                              //        1 = part edit online DRC
                              //    141 = Drill tool table optimization flag
                              //    142 = Bus trace count
                              //    143 = Edit bus trace count
                              //    144 = Bus trace creation mode:
                              //        0 = Create trace bundle
                              //        1 = Create seperate traces
                              //    145 = Bus trace corner mode:
                              //        0 = Create angle corners
                              //        1 = Create arc corners
                              //    146 = Silk screen layer (LAY1)
                              //    147 = Macro outline display mode:
                              //        0 = No macro outline display
                              //        1 = Display macro outline
                              //            at moved references
                              //        2 = Display macro outlines
    int;                      // Parameter value
    );
```

Description

The **ged_setintpar** function is used to set **Layout Editor** integer system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **ged_getintpar** function can be used to query parameter values set with **ged_setintpar**.

See also

Functions **ged_getdblpar**, **ged_getintpar**, **ged_getstrpar**, **ged_setdblpar**, **ged_setstrpar**.

ged_setlaydefmode - Set GED default layer mode (GED)

Synopsis

```
int ged_setlaydefmode(        // Returns status
      int [0,2];              // Default layer mode:
                              //    0 = automatic layer default disabled
                              //    1 = used edit layer as layer default
                              //    2 = last used layer as layer default
      );
```

Description

The **ged_setlaydefmode** function sets the **Layout Editor** default layer mode. The function returns zero if the assignment was successful or non-zero on error.

See also

Functions **ged_getlaydefmode**, **ged_getlayerdefault**, **ged_setlayerdefault**.

ged_setlayerdefault - Set GED default layer (GED)

Synopsis

```
int ged_setlayerdefault(      // Returns status
      int;                    // Layer (LAY1)
      );
```

Description

The **ged_setlayerdefault** function sets the **Layout Editor** default layer. The function returns zero if the assignment was successful or non-zero on error.

See also

Functions **ged_getlaydefmode**, **ged_getlayerdefault**, **ged_setlaydefmode**.

ged_setmincon - Set GED Mincon function type (GED)

Synopsis

```
int ged_setmincon(            // Returns status
      int [0,8];              // Required Mincon function type (LAY10)
      );
```

Description

The **ged_setmincon** function sets the currently active **Layout Editor** Mincon function type, i.e., the airline display mode (LAY10). The function returns nonzero if an invalid Mincon function type value has been specified.

ged_setnetattrib - Set GED net attribute value (GED)

Synopsis

```
int ged_setnetattrib(         // Returns status
      string;                 // Net name
      string;                 // Attribute name
      string;                 // Attribute value
      );
```

Description

The **ged_setnetattrib** function assigns a value to the given attribute of the name-specified net. Attribute values with a maximum length of up to 40 characters can be stored. The function returns zero on successful attribute value assignment, (-1) if no valid element is loaded, (-2) on missing and/or invalid parameters, (-3) if the net has not been found or (-4) if the attribute with the given name is not defined on the specified net.

ged_setpathwidth - Set GED path standard width (GED)

Synopsis

```
int ged_setpathwidth(          // Returns status
      double ]0.0,[;           // Required small path width (STD2)
      double ]0.0,[;           // Required wide path width (STD2)
      );
```

Description

The **ged_setpathwidth** function sets the currently active **Layout Editor** standard widths for small and wide traces. The function returns nonzero if invalid an invalid width value has been specified.

ged_setpickelem - GED Defaultpickelement setzen (GED)

Synopsis

```
int ged_setpickelem(           // Returns status
      index L_FIGURE;          // Default pick element
      );
```

Description

The **ged_setpickelem** function sets a default element for subsequent **Layout Editor** pick operations. The function returns zero if done or nonzero on error.

See also

Function **ged_pickelem**.

ged_setpickmode - Set GED element pick mode (GED)

Synopsis

```
int ged_setpickmode(           // Returns status
      int [0,2];               // Element pick mode:
                               //    0 = Pick peference layer pick
                               //    1 = Pick with element selection
                               //    2 = Exclusive pick preference layer pick
      );
```

Description

The **ged_setpickmode** function sets the **Layout Editor** element pick mode. The function returns nonzero for invalid pick mode specifications.

See also

Function **ged_getpickmode**.

ged_setpickpreflay - Set GED pick preference layer (GED)

Synopsis

```
int ged_setpickpreflay(        // Returns status
      int;                     // Required pick preference layer (LAY1)
      );
```

Description

The **ged_setpickpreflay** function sets the currently active **Layout Editor** pick preference layer for element selection (LAY1). The function returns nonzero if an invalid pick preference layer has been specified.

ged_setplantoplay - Set GED layout top layer (GED)

Synopsis

```
int ged_setplantoplay(        // Returns status
      int [0,99];             // Required layout top layer (LAY1)
      );
```

Description

The **ged_setplantoplay** function defines the **Layout Editor** and/or layout element top layer setting. The function returns nonzero if an invalid signal layer has been specified.

ged_setsegmovmode - Set GED trace segment move mode (GED)

Synopsis

```
int ged_setsegmovmode(        // Returns status
      int [0,12];             // Trace segment move mode:
                              //    0 = Move without neighbours
                              //    1 = Move with neighbours
                              //    2 = Adjust neighbours
                              //    3 = Adjust neighbours without vias
                              //    8 = Adjust next neighbours only
                              //    |4 = Open trace ends follow segment movement
      );
```

Description

The **ged_setsegmovmode** function sets the **Layout Editor** trace segment move mode. The function returns zero if the assignment was successful or non-zero on error.

See also

Function **ged_getsegmovmode**.

ged_setstrpar - Set GED string parameter (GED)

Synopsis

```
int ged_setstrpar(            // Returns status
      int [0,[;               // Parameter type/number:
                              //  [ 0 = System parameter - no write access ]
                              //  [ 1 = System parameter - no write access ]
                              //    2 = Last placed text string
                              //    3 = Standard library name
                              //  [ 4 = System parameter - no write access ]
                              //    5 = Drill naming base
                              //    6 = Drill part macro name pattern
                              //    7 = Drill padstack macro name pattern
                              //    8 = Input prompt override string
                              //  [ 9 = System parameter - no write access ]
                              //    10 = Autosave path name
      string;                 // Parameter value
      );
```

Description

The **ged_setstrpar** function is used to set **Layout Editor** string system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **ged_getstrpar** function can be used to query parameter values set with **ged_setstrpar**.

See also

Functions **ged_getdblpar**, **ged_getintpar**, **ged_getstrpar**, **ged_setdblpar**, **ged_setintpar**.

ged_setviaoptmode - Set GED trace via optimization mode (GED)

Synopsis

```
int ged_setviaoptmode(        // Returns status
      int [0,1];              // Trace via optimization mode:
                              //    0 = Via optimization
                              //    1 = Keep vias
      );
```

Description

The **ged_setviaoptmode** function sets the **Layout Editor** trace via optimization mode. The function returns zero if the assignment was successful or non-zero on error.

See also

Function **ged_getviaoptmode**.

ged_setwidedraw - Set GED wide line display start width (GED)

Synopsis

```
int ged_setwidedraw(          // Returns status
      double ]0.0,[;          // Required width value (STD2)
      );
```

Description

The **ged_setwidedraw** function sets the current **Layout Editor** wide line display start width, i.e., the minimum trace width for displaying traces like filled polygons. The function returns nonzero if an invalid width value is specified.

ged_storedrill - Place GED drill hole (GED)

Synopsis

```
int ged_storedrill(           // Returns status
      double;                 // Drill X coordinate (STD2)
      double;                 // Drill Y coordinate (STD2)
      double ]0.0,[;          // Drill radius (STD2)
      int [0,[;               // Drill class (LAY5)
      );
```

Description

The **ged_storedrill** function stores a drill hole with the given placement parameters to the currently loaded layout element. The function returns nonzero on wrong environment or missing/invalid parameters.

Warning

This function changes the current figure list and should be used carefully in **`forall`** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ged_storepart - Place GED part or padstack (GED)

Synopsis

```
int ged_storepart(            // Returns status
      string;                 // Reference name
      string;                 // Library symbol name
      double;                 // X coordinate (STD2)
      double;                 // Y coordinate (STD2)
      double;                 // Rotation angle (STD3)
      int [0,1];              // Mirror mode (STD14)
      );
```

Description

The **ged_storepart** function stores a part (or padstack) with the given placement parameters to the currently loaded layout (or part) element. The next unplaced net list part is used if an empty string is passed for the reference name. The function returns zero if the part has been successfully placed, (1) if the part pins do not match the net list specifications, (-1) on wrong environment or missing/invalid parameters, (-2) if all parts are placed already, (-3) if the specified part is placed already, (-4) if the part cannot be loaded or (-6) if the part data could not be copied to the current job file.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ged_storepath - Place GED internal polygon as path (GED)

Synopsis

```
int ged_storepath(            // Returns status
      int [0,99];             // Path layer (LAY1)
      double ]0.0,[;          // Path width (STD2)
      );
```

Description

The **ged_storepath** function generates a trace on the currently loaded layout and/or part using the specified placement parameters. The trace polygon points are taken from the internal polygon point list previously stored with **bae_storepoint**. The function returns zero if the trace has been successfully generated, (-1) on invalid environment, (-2) on missing and/or invalid parameters or (-3) if the point list is invalid.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

See also
Functions **bae_storepoint**, **ged_drcpath**.

ged_storepoly - Place GED internal polygon (GED)

Synopsis

```
int ged_storepoly(            // Returns status
      int;                    // Polygon layer (LAY1)
      int [1,9];              // Polygon type (LAY4)
      string;                 // Polygon net name (for LAY4 types 4, 6 and 9)
      int [0,18];             // Polygon mirror mode (LAY3)
      );
```

Description

The **ged_storepoly** function generates a polygon on the currently loaded layout element using the specified placement parameters. The polygon points are taken from the internal polygon point list previously stored with **bae_storepoint**. The function returns zero if the polygon has been successfully generated, (-1) on invalid environment, (-2) on missing and/or invalid parameters or (-3) if the point list is not valid for the specified polygon type.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

See also

Functions **bae_storepoint**, **ged_drcpoly**.

ged_storetext - Place GED text (GED)

Synopsis

```
int ged_storetext(            // Returns status
      string;                 // Text string
      double;                 // Text X coordinate (STD2)
      double;                 // Text Y coordinate (STD2)
      double;                 // Text rotation angle (STD3)
      double ]0.0,[;          // Text size (STD2)
      int;                    // Text layer (LAY1)
      int;                    // Text mirror mode and style (STD14|LAY14)
      );
```

Description

The **ged_storetext** function generates a text on the currently loaded layout element using the specified placement parameters. The function return value is nonzero on wrong environment or missing/invalid parameters.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops. The input text string can be stored to a maximum of up to 40 characters; longer strings cause the function to return with an invalid parameter error code.

See also

Function **ged_attachtextpos**.

ged_storeuref - Place GED unnamed reference (via or pad) (GED)

Synopsis

```
int ged_storeuref(            // Returns status
      string;                 // Library symbol name
      double;                 // Reference X coordinate (STD2)
      double;                 // Reference Y coordinate (STD2)
      double;                 // Reference rotation angle (STD3)
      int;                    // Reference layer (LAY1)
      int [0,1];              // Reference mirror (STD14)
      );
```

Description

The **ged_storeuref** function stores an unnamed reference (via or pad) with the given placement parameters to the currently loaded layout element (layout, part or padstack). For vias, the reference mirror mode, the reference layer and the rotation angle are ignored. The function returns zero if the reference has been successfully placed, (-1) on wrong environment or missing/invalid parameters, (-2) if the reference cannot be loaded or (-3) if the reference data could not be copied to the current job file.

Warning

This function changes the current figure list and should be used carefully in `forall` loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

C.4.3 Autorouter Functions

The following **User Language** system functions are assigned to caller type AR; i.e., they can be called from the **Autorouter** interpreter environment of the **Bartels AutoEngineer**:

ar_asklayer - Autorouter layer selection (AR)

Synopsis

```
int ar_asklayer(              // Returns status
      & int;                  // Returns selected layer (LAY1|LAY9)
      int [0,5];              // Layer query type:
                              //    0 = Documentary layers and signal layers
                              //    1 = Signal layers
                              //    2 = Signal layers
                              //        (including Top Layer and All Layers)
                              //    3 = Documentary layers
                              //    4 = Signal and power layers
                              //    5 = arbitrary display element types
      );
```

Description

The **ar_asklayer** function activates an **Autorouter** layer selection menu. The layer query type designates the type of layers and/or display element types provided for selection. The function returns zero if a valid layer has been selected or (-1) if the layer selection was aborted.

ar_delelem - Delete Autorouter figure list element (AR)

Synopsis

```
int ar_delelem(               // Returns status
      & index L_FIGURE;       // Element
      );
```

Description

The **ar_delelem** function deletes the given figure list element from the figure list. The function returns zero if the element was successfully deleted or nonzero on error.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

See also

Function **ar_drawelem**.

ar_drawelem - Redraw Autorouter figure list element (AR)

Synopsis

```
void ar_drawelem(
      index L_FIGURE;         // Element
      int [0, 4];             // Drawing mode (STD19)
      );
```

Description

The **ar_drawelem** function updates the display of the given figure list element using the specified drawing mode.

See also

Function **ar_delelem**.

ar_elemangchg - Change Autorouter figure list element rotation angle (AR)

Synopsis

```
int ar_elemangchg(            // Returns status
      & index L_FIGURE;       // Element
      double;                 // New rotation angle (STD3)
      );
```

Description

The **ar_elemangchg** function changes the rotation angle of the given figure list element. The rotation angle must be in radians. The function returns zero if the element has been successfully rotated, (-1) if the figure list element is invalid or (-2) if the figure list element cannot be rotated.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ar_elemfixchg - Change Autorouter figure list element fixed flag (AR)

Synopsis

```
int ar_elemfixchg(            // Returns status
      & index L_FIGURE;       // Element
      int [0,1];              // New fixed flag (STD11)
      );
```

Description

The **ar_elemfixchg** function changes the fixed flag of the given figure list element. The fixed flag value 0 unfixes the element, the fixed flag value 1 fixes the element. The function returns zero if the element fixed flag has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element cannot be fixed.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ar_elemlaychg - Change Autorouter figure list element layer (AR)

Synopsis

```
int ar_elemlaychg(            // Returns status
      & index L_FIGURE;       // Element
      int;                    // New layer (LAY1)
      );
```

Description

The **ar_elemlaychg** function changes the layer of the given figure list element. The layer can be set for polygons, traces, texts, pads (on padstack level) and drill holes. For drill holes the layer input parameter specifies the drill class code. The function returns zero if the element layer has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element layer cannot be set.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ar_elemmirrchg - Change Autorouter figure list element mirror mode (AR)

Synopsis

```
int ar_elemmirrchg(           // Returns status
      & index L_FIGURE;       // Element
      int [0,18];             // New mirror mode (STD14|LAY3)
      );
```

Description

The **ar_elemmirrchg** function changes the mirror mode of the given figure list element. The mirror mode can be set for polygons, texts and references. The function returns zero if the element mirror mode has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element mirror mode cannot be set.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ar_elemposchg - Change Autorouter figure list element position (AR)

Synopsis

```
int ar_elemposchg(            // Returns status
      & index L_FIGURE;       // Element
      double;                 // New X coordinate (STD2)
      double;                 // New Y coordinate (STD2)
      );
```

Description

The **ar_elemposchg** function changes the position of the given figure list element. Polygons and/or traces are replaced to set the first point of the polygon/trace to the specified position. The function returns zero if the element has been successfully repositioned, (-1) if the figure list element is invalid or (-2) if the figure list element position cannot be set.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ar_elemsizechg - Change Autorouter figure list element size (AR)

Synopsis

```
int ar_elemsizechg(           // Returns status
      & index L_FIGURE;       // Element
      double;                 // New size (STD2)
      );
```

Description

The **ar_elemsizechg** function changes the size of the given figure list element. The size can be changed for texts, drill holes and traces. For traces, a trace width change is performed. The functions value is zero if the element size has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element size cannot be set.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ar_getdblpar - Get Autorouter double parameter (AR)

Synopsis

```
int ar_getdblpar(             // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = Net visibility dialog net name list
                              //         control element width
                              //    1 = User-defined routing grid
      & double;               // Returns parameter value
      );
```

Description

The **ar_getdblpar** function is used to query **Autorouter** double parameters previously set with **ar_setdblpar**. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **ar_getintpar**, **ar_getstrpar**, **ar_setdblpar**, **ar_setintpar**, **ar_setstrpar**.

ar_getintpar - Get Autorouter integer parameter (AR)

Synopsis

```
int ar_getintpar(             // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = Top layer color code
                              //    1 = Mincon update mode
                              //    2 = Warning output mode:
                              //        Bit 0: Supress SCM changed warnings
                              //        Bit 1: not used in Autorouter
                              //        Bit 2: Suppress variant mismatch warnings
                              //    3 = Autosave interval
                              //    4 = Net visibility dialog box mode:
                              //        0 = Single column net name list display
                              //        1 = Multi-column net name list display
                              //    5 = Routing layer count
                              //    6 = Routing grid code:
                              //        0 = 1/20 Inch (1.27 mm) Standard
                              //        1 = 1/40 Inch (0.635 mm) Standard
                              //        2 = 1/50 Inch (0.508 mm) Standard
                              //        3 = 1/60 Inch (0.4233 mm) Standard
                              //        4 = 1/80 Inch (0.3175 mm) Standard
                              //        5 = 1/100 Inch (0.254 mm) Standard
                              //        6 = 1/40 Inch (0.635 mm) no Offset
                              //        7 = 1/60 Inch (0.4233 mm) no Offset
                              //        8 = 1/80 Inch (0.3175 mm) no Offset
                              //        9 = 1/100 Inch (0.254 mm) with Offset
                              //        -1 = Other Grid
                              //        -2 = Other Grid with Offset
                              //    7 = Mincon-Flächenmodus (Bitmuster):
                              //        0 = Kein Flächen-Mincon
                              //        |1 = Kupferflächen-Mincon
                              //        |2 = Potentialflächen-Mincon
                              //    8 = Flag - Color table saved
                              //    9 = Airline color mode:
                              //        0 = Use unroutes color
                              //        1 = Use layer color
                              //    10 = Drill tool table optimization flag
      & int;                  // Returns parameter value
      );
```

Description

The **ar_getintpar** function is used to query **Autorouter** integer parameters previously set with **ar_setintpar**. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **ar_getdblpar**, **ar_getstrpar**, **ar_setdblpar**, **ar_setintpar**, **ar_setstrpar**.

ar_getmincon - Get Autorouter Mincon function type (AR)

Synopsis

```
int ar_getmincon(             // Returns Mincon function type (LAY10)
      );
```

Description

The **ar_getmincon** function returns the currently active **Autorouter** Mincon function type, i.e., the airline display mode (LAY10).

ar_getpickpreflay - Get Autorouter pick preference layer (AR)

Synopsis

```
int ar_getpickpreflay(        // Returns pick preference layer (LAY1)
      );
```

Description

The **ar_getpickpreflay** function returns the currently active **Autorouter** pick preference layer for element selection (LAY1).

ar_getstrpar - Get Autorouter string parameter (AR)

Synopsis

```
int ar_getstrpar(             // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = Autosave path name
      & string;               // Returns parameter value
      );
```

Description

The **ar_getstrpar** function is used to query **Autorouter** string parameter settings. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **ar_getdblpar**, **ar_getintpar**, **ar_setdblpar**, **ar_setintpar**, **ar_setstrpar**.

ar_getwidedraw - Get Autorouter wide line display start width (AR)

Synopsis

```
double ar_getwidedraw(        // Returns width value (STD2)
      );
```

Description

The **ar_getwidedraw** function returns the current **Autorouter** wide line display start width, i.e., the minimum trace width for displaying traces like filled polygons.

ar_highlnet - Set Autorouter net highlight mode (AR)

Synopsis

```
int ar_highlnet(              // Returns status
      int [0,[;               // Net tree number
      int [0,1];              // Highlight mode (0 = off, 1 = on)
      );
```

Description

The **ar_highlnet** function sets the highlight mode of the net specified by the given net tree number. A highlight mode value of 1 highlights the net, a highlight mode value of 0 de-highlight the net. The function returns nonzero if an invalid net tree number and/or highlight mode value has been specified.

ar_partnamechg - Change Autorouter net list part name (AR)

Synopsis

```
int ar_partnamechg(           // Returns status
      string;                 // Old part name
      string;                 // New part name
      );
```

Description

The **ar_partnamechg** function changes the name of a net list part. The function returns nonzero if the part name has been successfully changed, (-1) for invalid input parameters, (-2) if the specified part is not yet placed, (-3) if the specified part does not exist in the net list, (-4) if the new name exists already and (-5) on multiple name change requests (e.g., **a** to **b** and then **b** to **c**) in one program run.

Warning

This function changes the net list and therefore requires a **Backannotation**. It is strongly recommended not to use this function in **L_CPART** index loops since the current **L_CPART** index variables are invalid after calling **ar_partnamechg**.

ar_pickelem - Pick Autorouter figure list element with mouse (AR)

Synopsis

```
int ar_pickelem(              // Returns status
      & index L_FIGURE;       // Returns picked element
      int [1,9];              // Pick element type (LAY6 except 7)
      );
```

Description

The **ar_pickelem** function activates an interactive figure list element pick request (with mouse). The required pick element type is specified with the second parameter. The picked figure list element index is returned with the first parameter. The function returns zero if an element has been picked or (-1) if no element of the required type has been found at the pick position.

ar_setdblpar - Set Autorouter double parameter (AR)

Synopsis

```
int ar_setdblpar(             // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = Net visibility dialog net name list
                              //         control element width
                              //  [ 1 = System parameter - no write access ]
      double;                 // Parameter value
      );
```

Description

The **ar_setdblpar** function is used to set **Autorouter** double system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **ar_getdblpar** function can be used to query parameter values set with **ar_setdblpar**.

See also
Functions **ar_getdblpar**, **ar_getintpar**, **ar_getstrpar**, **ar_setintpar**, **ar_setstrpar**.

ar_setintpar - Set Autorouter integer parameter (AR)

Synopsis

```
int ar_setintpar(              // Returns status
      int [0,[;                // Parameter type/number:
                               //  [ 0 = use bae_setcolor instead ]
                               //    1 = Mincon update mode
                               //    2 = Warning output mode:
                               //        Bit 0: Supress SCM changed warnings
                               //        Bit 1: not used in Autorouter
                               //        Bit 2: Supress variant mismatch warnings
                               //    3 = Autosave interval
                               //    4 = Net visibility dialog box mode:
                               //        0 = Single column net name list display
                               //        1 = Multi-column net name list display
                               //  [ 5 = System parameter - no write access ]
                               //  [ 6 = System parameter - no write access ]
                               //    7 = Mincon Area Mode (Bit Patterns):
                               //        0 = No Area Mincon
                               //        |1 = Copper Area Mincon
                               //        |2 = Connected Copper Area Mincon
                               //    8 = Flag - Color table saved
                               //    9 = Airline color mode:
                               //        0 = Use unroutes color
                               //        1 = Use layer color
                               //   10 = Drill tool table optimization flag
      int;                     // Parameter value
      );
```

Description

The **ar_setintpar** function is used to set **Autorouter** integer system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **ar_getintpar** function can be used to query parameter values set with **ar_setintpar**.

See also

Functions **ar_getdblpar**, **ar_getintpar**, **ar_getstrpar**, **ar_setdblpar**, **ar_setstrpar**.

ar_setmincon - Set Autorouter Mincon function type (AR)

Synopsis

```
int ar_setmincon(              // Returns status
      int [0,8];               // Required Mincon function type (LAY10)
      );
```

Description

The **ar_setmincon** function sets the currently active **Autorouter** Mincon function type, i.e., the airline display mode (LAY10). The function returns nonzero if an invalid Mincon function type value has been specified.

ar_setnetattrib - Set Autorouter net attribute value (AR)

Synopsis

```
int ar_setnetattrib(           // Returns status
      string;                  // Net name
      string;                  // Attribute name
      string;                  // Attribute value
      );
```

Description

The **ar_setnetattrib** function assigns a value to the given attribute of the name-specified net. Attribute values with a maximum length of up to 40 characters can be stored. The function returns zero if the attribute value assignment was successful, (-1) if no valid element is loaded, (-2) on missing and/or invalid parameters, (-3) if the net has not been found or (-4) if the attribute with the given name is not defined on the specified net.

ar_setpickpreflay - Set Autorouter pick preference layer (AR)

Synopsis

```
int ar_setpickpreflay(        // Returns status
      int;                    // Required pick preference layer (LAY1)
      );
```

Description

The **ar_setpickpreflay** function sets the currently active **Autorouter** pick preference layer for element selection (LAY1). The function returns nonzero if an invalid pick preference layer has been specified.

ar_setplantoplay - Set Autorouter layout top layer (AR)

Synopsis

```
int ar_setplantoplay(         // Returns status
      int [0,99];             // Required layout top layer (LAY1)
      );
```

Description

The **ar_setplantoplay** function defines the **Autorouter** and/or layout element top layer setting. The function returns nonzero if an invalid signal layer has been specified.

ar_setstrpar - Set Autorouter string parameter (GED)

Synopsis

```
int ar_setstrpar(             // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = Autosave path name
      string;                 // Parameter value
      );
```

Description

The **ar_setstrpar** function is used to set **Autorouter** string system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **ar_getstrpar** function can be used to query parameter values set with **ar_setstrpar**.

See also

Functions **ar_getdblpar**, **ar_getintpar**, **ar_getstrpar**, **ar_setdblpar**, **ar_setintpar**.

ar_setwidedraw - Set Autorouter wide line display start width (AR)

Synopsis

```
int ar_setwidedraw(           // Returns status
      double ]0.0,[;          // Required width value (STD2)
      );
```

Description

The **ar_setwidedraw** function sets the current **Autorouter** wide line display start width, i.e., the minimum trace width for displaying traces like filled polygons. The function returns nonzero if an invalid width value has been specified.

ar_storepart - Place Autorouter part or padstack (AR)

Synopsis

```
int ar_storepart(             // Returns status
      string;                 // Reference name
      string;                 // Library symbol name
      double;                 // X coordinate (STD2)
      double;                 // Y coordinate (STD2)
      double;                 // Rotation angle (STD3)
      int [0,1];              // Mirror mode (STD14)
      );
```

Description

The **ar_storepart** function stores a part (or padstack) with the given placement parameters to the currently loaded layout (or part) element. The next unplaced and selected net list part is used if an empty string is passed for the reference name. The function returns zero if the part has been successfully placed, (-1) on wrong environment or missing/invalid parameters, (-2) if all parts are already placed, (-3) if the specified part is already placed, (-4) if the part cannot be loaded, (-5) if the part pins do not match the net list specifications or (-6) if the part data could not be copied to the current job file.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ar_storepath - Place Autorouter internal polygon as path (AR)

Synopsis

```
int ar_storepath(             // Returns status
      int [0,99];             // Path layer (LAY1)
      double ]0.0,[;          // Path width (STD2)
      );
```

Description

The **ar_storepath** function generates a trace on the currently loaded layout or part using the specified placement parameters. The trace polygon points are taken from the internal polygon point list previously stored with **bae_storepoint**. The function returns zero if the trace has been successfully generated, (-1) on invalid environment, (-2) on missing and/or invalid parameters or (-3) if the point list is invalid.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ar_storeuref - Place Autorouter unnamed reference (via or pad) (AR)

Synopsis

```
int ar_storeuref(             // Returns status
      string;                 // Library symbol name
      double;                 // Reference X coordinate (STD2)
      double;                 // Reference Y coordinate (STD2)
      double;                 // Reference rotation angle (STD3)
      int;                    // Reference layer (LAY1)
      int [0,1];              // Reference mirror (STD14)
      );
```

Description

The **ar_storeuref** function stores an unnamed reference (via or pad) with the given placement parameters to the currently loaded layout element (layout, part or padstack). For vias, the reference mirror mode, the reference layer and the rotation angle are ignored. The function returns zero if the reference has been successfully placed, (-1) on wrong environment or missing/invalid parameters, (-2) if the reference cannot be loaded or (-3) if the reference data could not be copied to the current job file.

Warning

This function changes the current figure list and should be used carefully in `forall` loops for iterating **L_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

C.4.4 CAM Processor Functions

The following **User Language** system functions are assigned to caller type CAM; i.e., they can be called from the **CAM Processor** interpreter environment of the **Bartels AutoEngineer**:

cam_askplotlayer - CAM plot layer selection (CAM)

Synopsis

```
int cam_askplotlayer(         // Returns status
      & int;                  // Returns selected layer (LAY1)
      );
```

Description

The **cam_askplotlayer** function activates a **CAM Processor** layer selection menu. The function returns zero if a valid layer has been selected or (-1) if the layer selection has been aborted.

cam_getdblpar - Get CAM double parameter (CV)

Synopsis

```
int cam_getdblpar(            // Returns status
      int [0,[;               // Parameter type/number:
                              //    [ 0 = System parameter; write-only access ]
                              //    1 = Pixel bitmap resolution (STD2)
                              //    2 = Last bitmap plot pixel ratio
      & double;               // Returns parameter value
      );
```

Description

The **cam_getdblpar** function is used to query **CAM Processor** `double` parameters previously set with **cam_setdblpar**. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **cam_getintpar**, **cam_setdblpar**, **cam_setintpar**.

cam_getdrlaccuracy - CAM drill tool tolerance query (CAM)

Synopsis

```
double cam_getdrlaccuracy(    // Drill tool tolerance (STD2)
      );
```

Description

The **cam_getdrlaccuracy** function returns the **CAM Processor** drill tool tolerance.

See also
Function **cam_setdrlaccuracy**.

cam_getgenpltparam - CAM general plot parameter query (CAM)

Synopsis

```
void cam_getgenpltparam(
      & int;                  // Plot all layers off/on (1=on, 0=off)
      & int;                  // Plot border off/on flag (1=on, 0=off)
      & int;                  // Plot rotate off/on flag:
                              //    0 = rotate 0 degree
                              //    1 = rotate 90 degree left
      & int;                  // Plot mirror mode (CAM1)
      & int;                  // Plot markers off/on flag:
                              //    0 = off
                              //    1 = on
      & double;               // Plot accuracy (STD2)
      & double;               // Plot origin X coordinate (STD2)
      & double;               // Plot origin Y coordinate (STD2)
      );
```

Description

The **cam_getgenpltparam** function returns the **CAM Processor** general plot parameters.

cam_getgerberapt - CAM Gerber aperture definition query (CAM)

Synopsis

```
int cam_getgerberapt(         // Returns status
      int [1,900];            // Aperture table index
      & int;                  // Aperture D-code:
                              //    10..999 = valid D-codes
                              //       (-1) = aperture not defined
      & int;                  // Aperture type:
                              //    0 = special aperture
                              //    1 = round aperture
                              //    2 = square aperture
                              //    3 = thermal aperture (heat trap)
                              //    4 = rectangular aperture
      & int;                  // Aperture drawing mode:
                              //    0 = aperture for all drawing modes
                              //    1 = aperture for flash structures
                              //    2 = aperture for line structures
      & double;               // Aperture dimension/X size (STD2)
      & double;               // Aperture dimension/Y size (STD2)
      );
```

Description

The **cam_getgerberapt** function gets the definition of the Gerber aperture stored at the given table index of the aperture table currently loaded to the **CAM Processor**. A D-code of (-1) is returned if there is no aperture defined at the specified table position. The function returns nonzero on missing or invalid parameters.

cam_getgerberparam - CAM Gerber plot parameter query (CAM)

Synopsis

```
void cam_getgerberparam(
      & string;               // Gerber plot file name
      & double;               // Gerber standard line width (STD2)
      & int;                  // Gerber format (CAM4)
      & int;                  // Optimized Gerber output mode:
                              //    0 = Optimization off
                              //    1 = Optimization on
      & int;                  // Gerber fill mode:
                              //    0 = line fill
                              //    1 = multi-aperture fill
                              //    2 = G36/G37 fill
      & int;                  // Gerber arc output mode:
                              //    0 = use arc interpolation
                              //    1 = use Gerber I/J arc commands
      & int;                  // Extended Gerber (RS-274-X) mode:
                              //    0 = no Extended Gerber
                              //    1 = Extended Gerber with
                              //         standard aperture table
                              //    2 = Extended Gerber with
                              //         dynamic aperture table
      );
```

Description

The **cam_getgerberparam** function returns the **CAM Processor** Gerber plot parameters.

cam_gethpglparam - CAM HP-GL plot parameter query (CAM)

Synopsis

```
void cam_gethpglparam(
      & string;               // HP-GL plot file name
      & double;               // HP-GL plot scaling factor
      & double;               // HP-GL plotter speed (-1.0=full speed)
      & double;               // HP-GL plotter pen width (STD2)
      & int;                  // HP-GL plot area fill mode:
                              //    0 = fill off
                              //    1 = fill on
      );
```

Description

The **cam_gethpglparam** function returns the **CAM Processor** HP-GL plot parameters.

cam_getintpar - Get CAM integer parameter (CAM)

Synopsis

```
int cam_getintpar(            // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = Top layer color code
                              //    1 = Heat trap base angle
                              //    2 = Warning output mode:
                              //        Bit 0: Supress SCM changed warnings
                              //        Bit 1: not used in CAM Processor
                              //        Bit 2: Suppress variant mismatch warnings
                              //    3 = Area mirror visibility mode:
                              //        0 = Normal area mirror visibility
                              //        1 = Disable area mirror visibility
                              //    4 = Last pixel plot result type:
                              //        -1 = No pixel plot yet
                              //        0 = Board outline pixel ratio
                              //        1 = Element borders pixel ratio
                              //    5 = Last pixel plot total pixel count
                              //    6 = Last pixel plot copper pixel count
                              //    7 = Generic printer scale mode:
                              //        0 = Fixed scale factor
                              //        1 = Scale to paper size
                              //    8 = Flag - Color table saved
                              //    9 = Airline color mode:
                              //        0 = Use unroutes color
                              //        1 = Use layer color
                              //    10 = Bitmap outline milling mode:
                              //        0 = No Milling
                              //        1 = Draw filled outline with millings
                              //    11 = Generic printer drawing mode:
                              //        0 = Set Color
                              //        1 = Merge Color
                              //    12 = Batch output flag
                              //    13 = Drill tool table optimization flag
                              //    14 = Plot preview mode:
                              //        0 = 0 = None
                              //        1 = 1 = Plotter pen width
      & int;                  // Returns parameter value
      );
```

Description

The **cam_getintpar** function is used to query **CAM Processor** integer parameters previously set with **cam_setintpar**. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **cam_getdblpar**, **cam_setdblpar**, **cam_setintpar**.

cam_getplotlaycode - CAM plot layer code query (CAM)

Synopsis

```
int cam_getplotlaycode(       // Returns HP-GL plot pen number (CAM4)
      int;                    // Layer number (LAY1)
      );
```

Description

The **cam_getplotlaycode** function returns the layer-specific HP-GL plot pen number currently selected for multilayer plots. The layer-specific HP-GL pen number is also used for non-HP-GL multilayer plots where positive pen numbers denote layers currently selected for output and negative pen numbers denote layers not selected for output.

See also

Function **cam_setplotlaycode**.

cam_getpowpltparam - CAM power layer plot parameter query (CAM)

Synopsis

```
void cam_getpowpltparam(
      & double;               // Min. distance heat-trap to drill (STD2)
      & double;               // Min. distance isolation to drill (STD2)
      & double;               // Tolerance heat-trap to drill (STD2)
      & double;               // Tolerance isolation to drill (STD2)
      & double;               // Power layer border width (STD2)
      & double;               // Power plane isolation width (STD2)
      );
```

Description

The **cam_getpowpltparam** function returns the **CAM Processor** power layer plot parameters.

cam_getwidedraw - CAM wide line display start width query (CAM)

Synopsis

```
double cam_getwidedraw(       // Returns width value (STD2)
      );
```

Description

The **cam_getwidedraw** function returns the **CAM Processor** wide line display start width, i.e., the minimum trace width for displaying traces like filled polygons.

cam_plotgerber - CAM Gerber photo plot output (CAM)

Synopsis

```
int cam_plotgerber(           // Returns status
      int;                    // Gerber plot layer (LAY1)
      string;                 // Gerber plot file name
      double [0.00001,0.01];  // Gerber standard line width (STD2)
      double ]0.00000000053,[;// Gerber plotter unit length (CAM2)
      int [0,1];              // Optimized Gerber output mode:
                              //    0 = Optimization off
                              //    1 = Optimization on
      int [0,2];              // Gerber fill mode:
                              //    0 = line fill
                              //    1 = multi-aperture fill
                              //    2 = G36/G37 fill
      int [0,1];              // Gerber arc output mode:
                              //    0 = use arc interpolation
                              //    1 = use Gerber I/J arc commands
      int [0,2];              // Extended Gerber (RS-274-X) mode:
                              //    0 = no Extended Gerber
                              //    1 = Extended Gerber with
                              //         standard aperture table
                              //    2 = Extended Gerber with
                              //         dynamic aperture table
      int [0,1];              // Error highlight reset flag:
                              //    0 = keep error highlight
                              //    1 = de-highlight errors
      & int;                  // Returns flashed structure count
      & int;                  // Returns rect. filled structure count
      & int;                  // Returns circle filled structure count
      & int;                  // Returns multi filled structure count
      & int;                  // Returns line filled structure count
      & int;                  // Returns line drawn heat-traps count
      & int;                  // Returns overdraw error count
      );
```

Description

The **cam_plotgerber** function generates the Gerber photo plot data for the specified layer and writes it to a file. The function only sets the specified Gerber standard line width, fill mode, and arc output mode plot parameters and reset the error highlight, if no output file name is specified (empty string). The function returns zero for successfully generated plots, 1 for invalid plot parameter specifications (i.e., parameter out of range, no aperture for standard line width, etc.) or (-1) on plot errors. Plot overdraw errors are automatically highlighted.

cam_plothpgl - CAM HP-GL pen plot output (CAM)

Synopsis

```
int cam_plothpgl(             // Returns status
      int;                    // HP-GL plot layer (LAY1)
      int [1,99];             // HP-GL pen number
      string;                 // HP-GL plot file name
      double [0.1,100];       // HP-GL scaling factor
      double [-1.0,99];       // HP-GL speed ([centimetres/second]) or:
                              //    -1.0 = full speed
      double [0.00001,0.01];  // HP-GL pen width (STD2)
      int [0,1];              // HP-GL area fill mode:
                              //    0 = fill off
                              //    1 = fill on
      int [0,1];              // Error highlight reset flag:
                              //    0 = keep error highlight
                              //    1 = de-highlight errors
      & int;                  // Returns overdraw error count
      );
```

Description

The **cam_plothpgl** function generates the HP-GL pen plot data for the specified layer and writes it to a file. The function only sets the specified HP-GL plot parameters if no output file name is specified (empty string). The function returns zero for successfully generated plots or (-1) on plot errors or invalid plot parameter specifications. Plot overdraw errors are automatically highlighted.

cam_setdblpar - Set CAM double parameter (CAM)

Synopsis

```
int cam_setdblpar(            // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = Add extra dynamic aperture width, 0.0
clear list (STD2)
                              //    1 = Pixel bitmap resolution (STD2)
                              //    2 = Last bitmap plot pixel ratio
      double;                 // Parameter value
      );
```

Description

The **cam_setdblpar** function is used to set **CAM Processor** `double` system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **cam_getdblpar** function can be used to query parameter values set with **cam_setdblpar**.

See also

Functions **cam_getdblpar**, **cam_getintpar**, **cam_setintpar**.

cam_setdrlaccuracy - Set CAM drill tool tolerance (CAM)

Synopsis

```
int cam_setdrlaccuracy(       // Returns status
      double [0.0,0.01];      // Drill tool tolerance (STD2)
      );
```

Description

The **cam_getdrlaccuracy** function sets the **CAM Processor** drill tool tolerance. The function returns nonzero if invalid parameters are specified.

See also

Function **cam_getdrlaccuracy**.

cam_setgenpltparam - Set CAM general plot parameters (CAM)

Synopsis

```
int cam_setgenpltparam(       // Returns status
      int [0,4];              // Plot all layers off/on:
                              //    0 = all layer plot mode off
                              //    1 = all layer plot mode on
                              //    2 = plot connected pins/vias only
                              //    3 = plot all pins and connected vias
                              //    4 = plot all vias and connected pins
      int [0,1];              // Plot border off/on flag:
                              //    0 = off
                              //    1 = on
      int [0,1];              // Plot rotate off/on flag:
                              //    0 = rotate 0 degree
                              //    1 = rotate 90 degree left
      int [0,5];              // Plot mirror mode (CAM1)
      int [0,1];              // Plot markers off/on flag:
                              //    0 = off
                              //    1 = on
      double [0.0,0.01];      // Plot accuracy (STD2)
      double;                 // Plot origin X coordinate (STD2)
      double;                 // Plot origin Y coordinate (STD2)
      );
```

Description

The **cam_setgenpltparam** function sets the **CAM Processor** general plot parameters. The function returns nonzero if invalid parameters are specified.

cam_setgerberapt - Set CAM Gerber aperture definition (CAM)

Synopsis

```
int cam_setgerberapt(         // Returns status
      int [1,900];            // Aperture table index
      int;                    // Aperture D-code:
                              //    10..999 = valid D-codes
                              //       (-1) = delete aperture
      int [0,4];              // Aperture type:
                              //    0 = special aperture
                              //    1 = round aperture
                              //    2 = square aperture
                              //    3 = thermal aperture (heat trap)
                              //    4 = rectangular aperture
      int [0,2];              // Aperture drawing mode:
                              //    0 = aperture for all drawing modes
                              //    1 = aperture for flash structures
                              //    2 = aperture for line structures
      double [0.0,[;          // Aperture dimension/X size (STD2)
      double [0.0,[;          // Aperture dimension/Y size (STD2)
      );
```

Description

The **cam_setgerberapt** function sets the definition of the Gerber aperture at the given table index in the aperture table currently loaded to the **CAM Processor**. A D-code value of (-1) resets the aperture definition at the specified table position. The aperture size is ignored for special aperture types. The function returns nonzero on missing or invalid parameters.

cam_setintpar - Set CAM integer parameter (CAM)

Synopsis

```
int cam_setintpar(              // Returns status
      int [0,[;                 // Parameter type/number:
                                //  [ 0 = use bae_setcolor instead ]
                                //    1 = heat trap base angle
                                //    2 = Warning output mode:
                                //        Bit 0: Supress SCM changed warnings
                                //        Bit 1: not used in CAM Processor
                                //        Bit 2: Suppress variant mismatch warnings
                                //  [ 3 = System parameter - no write access ]
                                //  [ 4 = System parameter - no write access ]
                                //  [ 5 = System parameter - no write access ]
                                //  [ 6 = System parameter - no write access ]
                                //    7 = Generic printer scale mode:
                                //        0 = Fixed scale factor
                                //        1 = Scale to paper size
                                //    8 = Flag - Color table saved
                                //    9 = Airline color mode:
                                //        0 = Use unroutes color
                                //        1 = Use layer color
                                //   10 = Bitmap outline milling mode:
                                //        0 = No Milling
                                //        1 = Draw filled outline with millings
                                //   11 = Generic printer drawing mode:
                                //        0 = Set Color
                                //        1 = Merge Color
                                //   12 = Batch output flag
                                //   13 = Drill tool table optimization flag
                                //   14 = Plot preview mode:
                                //        0 = 0 = None
                                //        1 = 1 = Plotter pen width
      int;                      // Parameter value
      );
```

Description

The **cam_setintpar** function is used to set **CAM Processor** integer system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **cam_getintpar** function can be used to query parameter values set with **cam_setintpar**.

See also

Functions **cam_getdblpar**, **cam_getintpar**, **cam_setdblpar**.

cam_setplotlaycode - Set CAM plot layer code (CAM)

Synopsis

```
void cam_setplotlaycode(
      int;                      // Layer number (LAY1)
      int;                      // HP-GL plot pen number (CAM4)
      );
```

Description

The **cam_setplotlaycode** function selects and/or sets the specified layer-specific HP-GL plot pen number for multilayer plots. The layer-specific HP-GL pen number is also used for non-HP-GL multilayer plots where positive pen numbers denote layers currently selected for output and negative pen numbers denote layers not selected for output.

See also

Function **cam_getplotlaycode**.

cam_setpowpltparam - Set CAM power layer plot parameters (CAM)

Synopsis

```
int cam_setpowpltparam(       // Returns status
      double [0.0,0.01];      // Min. distance heat-trap to drill (STD2)
      double [0.0,0.01];      // Min. distance isolation to drill (STD2)
      double [0.0,0.01];      // Tolerance heat-trap to drill (STD2)
      double [0.0,0.01];      // Tolerance isolation to drill (STD2)
      double [0.0,0.02];      // Power layer border width (STD2)
      double [0.0,0.02];      // Power plane isolation width (STD2)
      );
```

Description

The **cam_setpowpltparam** function sets the **CAM Processor** power plot parameters. The function returns nonzero if invalid plot parameters are specified.

cam_setwidedraw - Set CAM wide line display start width (CAM)

Synopsis

```
int cam_setwidedraw(          // Returns status
      double ]0.0,[;          // Input width value (STD2)
      );
```

Description

The **cam_setwidedraw** function sets the current **CAM Processor** wide line display start width, i.e., the minimum trace width for displaying traces like filled polygons. The function returns nonzero if an invalid width value is specified.

C.4.5 CAM View Functions

The following **User Language** system functions are assigned to caller type CV; i.e., they can be called from the **CAM View** interpreter environment of the **Bartels AutoEngineer**:

cv_aptgetcolor - Get CAM View aperture color (CV)

Synopsis

```
int cv_aptgetcolor(           // Color value (STD18)
      int;                    // Aperture index
      int;                    // Aperture mode
      );
```

Description

The **cv_aptgetcolor** function returns the color value which is currently assigned in **CAM View** for displaying the specified Gerber aperture type.

See also

Function **cv_aptsetcolor**.

cv_aptsetcolor - Set CAM View aperture color (CV)

Synopsis

```
int cv_aptsetcolor(           // Returns status
      int;                    // Aperture index
      int;                    // Aperture mode
      int [-33554432,33554431];
                              // Color value (STD18)
      );
```

Description

The **cv_aptsetcolor** function sets the color value to be used in **CAM View** for displaying the specified Gerber aperture type. The function returns zero if the assignment was successful or nonzero otherwise.

See also

Function **cv_aptgetcolor**.

cv_deldataset - Delete CAM View data set (CV)

Synopsis

```
int cv_deldataset(            // Returns status
      int [0,[;               // Data set index
      );
```

Description

The **cv_deldataset** function removes the specified **CAM View** data set from the workspace. The function returns zero if the operation was successfully completed or nonzero otherwise.

See also
Function **cv_movedataset**.

cv_getdblpar - Get CAM View double parameter (CV)

Synopsis

```
int cv_getdblpar(             // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = Input X offset (STD2)
                              //    1 = Input Y offset (STD2)
                              //    2 = Heat trap isolation width (STD2)
                              //    3 = Wide line display start width (STD2)
                              //    4 = Length of one Gerber plotter unit (STD2)
      & double;               // Returns parameter value
      );
```

Description

The **cv_getdblpar** function is used to query **CAM View** `double` parameters previously set with **cv_setdblpar**. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **cv_getintpar**, **cv_setdblpar**, **cv_setintpar**.

cv_getintpar - Get CAM View integer parameter (CV)

Synopsis

```
int cv_getintpar(              // Returns status
      int [0,[;                // Parameter type/number:
                               //    0 = Gerber import layer selection mode:
                               //        0 = Assume single flash and
                               //             line input layer
                               //        1 = Select both flash and line input layer
                               //    1 = Gerber layer query:
                               //        0 = Layer not used
                               //        1 = Layer used
                               //    2 = Display color table/assignment:
                               //        0 = Aperture color table/assignment
                               //        1 = Layer color table/assignment
                               //    3 = Area display mode:
                               //        0 = Filled display
                               //        1 = Outline display
                               //    4 = Via D-Code
                               //    5 = Heat trap base angle
                               //    6 = Gerber Optimization:
                               //        0 = Coordinate optimization off
                               //        1 = Coordinate optimization on
                               //    7 = Gerber Circle/Arc Mode:
                               //        0 = Arbitrary Gerber arc angles
                               //        1 = Max. 90 Gerber arc angles
                               //    8 = Input Mirror Mode:
                               //        0 = Mirroring Off
                               //        1 = Mirroring at X-Axis
                               //        2 = Mirroring at Y-Axis
                               //        3 = Mirroring at Origin
                               //    9 = Zero Supression Mode:
                               //        0 = Suppress leading zeros
                               //        1 = Suppress trailing zeros
                               //    10 = Extended Gerber:
                               //        0 = Extended Gerber (Header) off
                               //        1 = Extended Gerber (Header) on
                               //    11 = Gerber Coordinate Specification:
                               //        0 = Absolute Coordinates
                               //        1 = Incremental Coordinates with Reset
                               //        2 = Incremental Coordinates without Reset
                               //    12 = Gerber Documentary Layer Mode
                               //        0 = Flashes as Documentary Line
                               //        1 = Flashes as Documentary Area
      & int;                   // Returns parameter value
      );
```

Description

The **cv_getintpar** function is used to query **CAM View** `int` parameters previously set with **cv_setintpar**. The functions returns zero if the query was successful or (-1) otherwise.

See also

Functions **cv_getdblpar**, **cv_setdblpar**, **cv_setintpar**.

cv_movedataset - Move CAM View data set (CV)

Synopsis

```
int cv_movedataset(           // Returns status
      int [0,[;               // Data set index
      int;                    // Data set movement mirror flag
      double;                 // Data set movement X offset (STD2)
      double;                 // Data set movement Y offset (STD2)
      );
```

Description

The **cv_movedataset** function moves (and mirrors) the specified **CAM View** data set using the specified offset parameters. The function returns zero if the operation was successfully completed or nonzero otherwise.

See also

Function **cv_deldataset**.

cv_setdblpar - Set CAM View double parameter (CV)

Synopsis

```
int cv_setdblpar(             // Returns status
      int [0,[;               // Parameter type/number:
                              //    0 = Input X offset (STD2)
                              //    1 = Input Y offset (STD2)
                              //    2 = Heat trap isolation width (STD2)
                              //    3 = Wide line display start width (STD2)
                              //    4 = Length of one Gerber plotter unit (STD2)
      double;                 // Parameter value
      );
```

Description

The **cv_setdblpar** function is used to set **CAM View** `double` system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **cv_getdblpar** function can be used to query parameter values set with **cv_setdblpar**.

See also

Functions **cv_getdblpar**, **cv_getintpar**, **cv_setintpar**.

cv_setintpar - Set CAM View integer parameter (CV)

Synopsis

```
int cv_setintpar(              // Returns status
      int [0,[;                // Parameter type/number:
                               //    0 = Gerber import layer selection mode:
                               //        0 = Assume single flash and
                               //             line input layer
                               //        1 = Select both flash and line input layer
                               //    1 = Gerber layer usage:
                               //        Read-only parameter!
                               //    2 = Display color table/assignment:
                               //        0 = Aperture color table/assignment
                               //        1 = Layer color table/assignment
                               //    3 = Area display mode:
                               //        0 = Filled display
                               //        1 = Outline display
                               //    4 = Via D-Code
                               //    6 = Gerber Optimization:
                               //        0 = Coordinate optimization off
                               //        1 = Coordinate optimization on
                               //    7 = Gerber Circle/Arc Mode:
                               //        0 = Arbitrary Gerber arc angles
                               //        1 = Max. 90 Gerber arc angles
                               //    8 = Input Mirror Mode:
                               //        0 = Mirroring Off
                               //        1 = Mirroring at X-Axis
                               //        2 = Mirroring at Y-Axis
                               //        3 = Mirroring at Origin
                               //    9 = Zero Supression Mode:
                               //        0 = Suppress leading zeros
                               //        1 = Suppress trailing zeros
                               //    10 = Extended Gerber:
                               //        0 = Extended Gerber (Header) off
                               //        1 = Extended Gerber (Header) on
                               //    11 = Gerber Coordinate Specification:
                               //        0 = Absolute Coordinates
                               //        1 = Incremental Coordinates with Reset
                               //        2 = Incremental Coordinates without Reset
                               //    12 = Gerber Documentary Layer Mode
                               //        0 = Flashes as Documentary Line
                               //        1 = Flashes as Documentary Area
      int;                     // Parameter value
      );
```

Description

The **cv_setintpar** function is used to set **CAM View** integer system parameters. The function returns zero if the parameter assignment was successful, or (-1) otherwise. The **cv_getintpar** function can be used to query parameter values set with **cv_setintpar**.

See also

Functions **cv_getdblpar**, **cv_getintpar**, **cv_setdblpar**.

C.5 IC Design System Functions

This section describes (in alphabetical order) the IC design system functions of the **Bartels User Language**. See Appendix C.1 for function description notations.

C.5.1 IC Design Data Access Functions

The following **User Language** system functions are assigned to caller type ICD; i.e., they can be called from the **Chip Editor** interpreter environment of the **Bartels AutoEngineer**:

icd_altpinlay - IC Design setup alternate pin layer (ICD)

Synopsis

```
int icd_altpinlay(            // Returns layer number (ICD1)
      );
```

Description

The **icd_altpinlay** function returns the alternate pin layer number defined in the BAE IC setup file for GDS input.

icd_cellconlay - IC Design setup internal cell connection layer (ICD)

Synopsis

```
int icd_cellconlay(           // Returns layer number (ICD1)
      );
```

Description

The **icd_cellconlay** function returns the layer number defined in the BAE IC setup for internal cell connections.

icd_cellscan - IC Design setup DRC on cell level mode (ICD)

Synopsis

```
int icd_cellscan(             // Returns cell DRC mode
      );
```

Description

The **icd_cellscan** function returns the cell DRC mode defined in the BAE IC setup (0 = no DRC for cell structures, 1 = DRC for cell structures).

icd_cellshr - IC Design setup cell keepout area shrink (ICD)

Synopsis

```
double icd_cellshr(           // Cell keepout area shrink value (STD2)
      );
```

Description

The **icd_cellshr** function returns the shrink value defined in the BAE IC setup for the automatic autorouter keepout area generation.

icd_ciflayname - IC Design setup CIF output layer name (ICD)

Synopsis

```
string icd_ciflayname(        // Returns layer name
      int [0,99];             // Layer number (ICD1)
      );
```

Description

The **icd_ciflayname** function returns the CIF output layer name defined for the given layer number (ICD1) in the BAE IC setup file.

icd_cstdsiz - IC Design setup standard cell height (ICD)

Synopsis

```
double icd_cstdsiz(           // Returns standard cell height (STD2)
      );
```

Description

The **icd_cellshr** function returns the standard cell height value defined in the BAE IC setup for the automatic cell placer.

icd_defelemname - IC Design setup default element name (ICD)

Synopsis

```
string icd_defelemname(       // Returns default element name
      );
```

Description

The **icd_defelemname** function returns the default **IC Design** element name defined in the BAE IC setup file.

icd_deflibname - IC Design setup default library name (ICD)

Synopsis

```
string icd_deflibname(        // Returns default library name
      );
```

Description

The **icd_deflibname** function returns the default **IC Design** library name defined in the BAE IC setup file.

icd_drcarc - IC Design setup DRC arc mode (ICD)

Synopsis

```
int icd_drcarc(               // Returns DRC arc mode
      );
```

Description

The **icd_drcarc** function returns the DRC arc mode defined in the BAE IC setup (0 = arcs allowed, 1 = no arcs allowed).

icd_drcgrid - IC Design setup DRC grid (ICD)

Synopsis

```
double icd_drcgrid(           // Returns DRC grid value (STD2)
      );
```

Description

The **icd_drcgrid** function returns the DRC grid value defined in the BAE IC setup.

icd_drclaymode - IC Design setup layer DRC mode (ICD)

Synopsis

```
int icd_drclaymode(           // Returns layer DRC mode
      int [0,99];             // Layer number (ICD1)
      );
```

Description

The **icd_drclaymode** function returns the DRC mode for the given layer defined in the BAE IC setup (0 = no DRC on the given layer, 1 = DRC on the given layer).

icd_drcmaxpar - IC Design setup DRC parallel check length (ICD)

Synopsis

```
double icd_drcmaxpar(         // Returns parallel check length (STD2)
      );
```

Description

The **icd_drcmaxpar** function returns the DRC maximal parallel structures length value value defined in the BAE IC setup.

icd_drcminwidth - IC Design setup DRC layer minimal dimensions (ICD)

Synopsis

```
double icd_drcminwidth(       // Returns DRC minimal value (STD2)
      int [0,99];             // Layer number (ICD1)
      );
```

Description

The **icd_drcminwidth** function returns the DRC minimal structure size value for the given layer defined in the BAE IC setup.

icd_drcrect - IC Design setup DRC orthogonal mode (ICD)

Synopsis

```
int icd_drcrect(              // Returns DRC orthogonal mode
      );
```

Description

The **icd_drcarc** function returns the DRC arc mode defined in the BAE IC setup (0 = arbitrary angles allowed, 1 = only right angles allowed).

icd_ecnlaymode - IC Design setup layer connectivity check (ICD)

Synopsis

```
int icd_ecnlaymode(           // Returns layer connectivity mode
      int [0,99];             // Layer number (ICD1)
      );
```

Description

The **icd_ecnlaymode** function returns the connectivity mode for the given layer defined in the BAE IC setup (0 = no connectivity on the given layer, 1 = connectivity on the given layer).

icd_findconpart - Find IC Design part index of a named part (ICD)

Synopsis

```
int icd_findconpart(          // Returns status
      string;                 // Part name
      & index I_CPART;        // Returns part index
      );
```

Description

The **icd_findconpart** function searches the **IC Design** connection list part index with the specified part name. The function returns zero if the part has been found or nonzero otherwise.

See also

Functions **icd_findconpartpin**, **icd_findcontree**.

icd_findconpartpin - Find IC Design part pin index of a named part pin (ICD)

Synopsis

```
int icd_findconpartpin(       // Returns status
      string;                 // Pin name
      index I_CPART;          // Net list part index
      & index I_CPIN;         // Returns net list part pin index
      );
```

Description

The **icd_findconpartpin** function searches an IC Design connection list part for the part pin index with the specified pin name. The function returns zero if the part pin has been found or nonzero otherwise.

See also

Functions **icd_findconpart**, **icd_findcontree**.

icd_findcontree - Find IC Design net index of a named net (ICD)

Synopsis

```
int icd_findcontree(          // Returns status
      string;                 // Net name
      & index I_CNET;         // Returns net index
      );
```

Description

The **icd_findcontree** function searches the **IC Design** connection list net index with the specified net name. The function returns zero if the net has been found or nonzero otherwise.

See also

Functions **icd_findconpart**, **icd_findconpartpin**.

icd_getrulecnt - Get rule count for specific object (ICD)

Synopsis

```
int icd_getrulecnt(           // Returns rule count or (-1) on error
      int;                    // Object class code
      int;                    // Object ident code (int or index type)
      );
```

Description

The **icd_getrulecnt** function is used for determining the number of rules attached to a specific object. The object can be the currently loaded element (object class code 0 with **int** value 0 passed for the object ident code), a figure list element of the currently loaded element (object class code 1 with valid **I_FIGURE** index type value passed for the object ident code), or a pool list element (object class code 2 with valid **I_POOL** index type value passed for the object ident code). The function returns a (non-negative) rule count or (-1) on error. The rule count determines the valid range for rule list indices to be passed to the **icd_getrulename** function for getting object-specific rule names. The **icd_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **icd_getrulecnt** function.

See also

Functions **icd_getrulename**, **icd_ruleerr**, **icd_rulefigatt**, **icd_rulefigdet**, **icd_ruleplanatt**, **icd_ruleplandet**, **icd_rulequery**; **Neural Rule System** and Rule System Compiler..

icd_getrulename - Get rule name from specific object (ICD)

Synopsis

```
int icd_getrulename(          // Returns nonzero on error
      int;                    // Object class code
      int;                    // Object ident code (int or index type)
      int [0,[;               // Rule name list index
      & string;               // Rule name result
      );
```

Description

The **icd_getrulename** function is used to get the name of an index-specified rule assigned to the specified object. The object can be the currently loaded element (object class code 0 with `int` value 0 passed for the object ident code), a figure list element of the currently loaded element (object class code 1 with valid **I_FIGURE** index type value passed for the object ident code), or a pool list element (object class code 2 with valid **I_POOL** index type value passed for the object ident code). The rule name list index to be specified can be determined using the **icd_getrulecnt** function. The rule name is returned with the last function parameter. The function returns zero on success or nonzero on error. The **icd_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **icd_getrulename** function.

See also

Functions **icd_getrulecnt**, **icd_ruleerr**, **icd_rulefigatt**, **icd_rulefigdet**, **icd_ruleplanatt**, **icd_ruleplandet**, **icd_rulequery**; **Neural Rule System** and Rule System Compiler..

icd_gettreeidx - Find IC Design net index of a tree (ICD)

Synopsis

```
int icd_gettreeidx(           // Returns status
      int;                    // Net tree number
      & index I_CNET;         // Returns net index
      );
```

Description

The **icd_gettreeidx** function searches the **IC Design** connection list net index with the specified net tree number. The function returns zero if the net has been found or nonzero otherwise.

icd_grpdisplay - IC Design setup group display layer (ICD)

Synopsis

```
int icd_grpdisplay(           // Returns layer number (ICD1)
      );
```

Description

The **icd_grpdisplay** function returns the group display layer number defined in the BAE IC setup file.

icd_lastfigelem - Get last modified IC Design figure list element (ICD)

Synopsis

```
int icd_lastfigelem(          // Returns status
      & index I_FIGURE;       // Returns figure list index
      );
```

Description

The **icd_lastfigelem** function gets the last created and/or modified **IC Design** figure list element and returns the corresponding figure list index with the return parameter. The function returns zero if such an element exists or nonzero else.

icd_maccoords - Get IC Design (scanned) macro coordinates (ICD)

Synopsis

```
void icd_maccoords(
      & double;               // Macro X coordinate (STD2)
      & double;               // Macro Y coordinate (STD2)
      & double;               // Macro rotation angle (STD3)
      & double;               // Macro scaling factor
      & int;                  // Macro mirror mode (STD14)
      );
```

Description

The **icd_maccoords** function returns with its parameters the placement data of the currently scanned macro. This function is intended for use in the macro callback function of **icd_scanall**, **icd_scanfelem** or **icd_scanpool** only (otherwise zero/default values are returned).

See also

Functions **icd_scanall**, **icd_scanfelem**, **icd_scanpool**.

icd_nrefsearch - Search named IC Design reference (ICD)

Synopsis

```
int icd_nrefsearch(           // Returns status
      string;                 // Reference name
      & index I_FIGURE;       // Returns figure list index
      );
```

Description

The **icd_nrefsearch** function searches for the specified named reference on the currently loaded **IC Design** element. The figure list index is set accordingly if the named reference is found. The function returns zero if the named reference has been found or nonzero otherwise.

icd_outlinelay - IC Design setup cell outline layer (ICD)

Synopsis

```
int icd_outlinelay(           // Returns layer number (ICD1)
      );
```

Description

The **icd_outlinelay** function returns the cell outline layer number defined in the BAE IC setup.

icd_pindist - IC Design setup pin keepout distance (ICD)

Synopsis

```
double icd_pindist(           // Returns pin keepout distance (STD2)
      );
```

Description

The **icd_pindist** function returns the pin keepout distance value defined in the BAE IC setup for the automatic keepout area generation.

icd_plcxgrid - IC Design setup placement grid (ICD)

Synopsis

```
double icd_plcxgrid(          // Returns placement grid value (STD2)
      );
```

Description

The **icd_plcxgrid** function returns the horizontal cell placement grid value defined in the BAE IC setup for the automatic cell placement.

icd_plcxoffset - IC Design setup placement offset (ICD)

Synopsis

```
double icd_plcxoffset(        // Returns placement offset value (STD2)
      );
```

Description

The **icd_plcxoffset** function returns the horizontal cell placement offset value defined in the BAE IC setup for the automatic cell placement.

icd_routcellcnt - IC Design setup number of power supply cells (ICD)

Synopsis

```
int icd_routcellcnt(          // Returns cell count
      );
```

Description

The **icd_routcellcnt** function returns the number of power supply cells defined in the BAE IC setup.

icd_routcellname - IC Design setup name of power supply cell (ICD)

Synopsis

```
string icd_routcellname(
                              // Returns cell name
      int [0,[;               // Cell index
      );
```

Description

The **icd_routcellname** function returns the name of a power supply cell defined in the BAE IC setup file. The index can be in the range of 0 to **icd_routcellcnt**()-1.

icd_ruleerr - Rule System error status query (ICD)

Synopsis

```
void icd_ruleerr(
      & int;              // Error item code
      & string;           // Error item string
      );
```

Description

The **icd_ruleerr** function provides information on the current **Rule System** error state, and thus can be used to determine the error reason after an unsuccessful call to one of the **Rule System** management functions.

Diagnosis

The **Rule System** error state can be determined by evaluating the parameters returned with the **icd_ruleerr** function. The returned error item string identifies the error-causing element if needed. The possible error code values correspond with **Rule System** error conditions according to the following table:

Error Code	Meaning
0	Rule System operation completed without errors
1	Rule System out of memory
2	Rule System internal error **<e>**
3	Rule System function parameter invalid
128	Rule System DB file create error
129	Rule System DB file read/write error
130	Rule System DB file wrong type
131	Rule System DB file structure bad
132	Rule System DB file not found
133	Rule System DB file other error (internal error)
134	Rule System rule **<r>** not found in rule database
135	Rule System rule bad DB format (internal error **<e>**)
136	Rule System object not found
137	Rule System object double defined (internal error)
138	Rule System incompatible variable **<v>** definition
139	Rule System Rule **<r>** compiled with incompatible RULECOMP version

Depending on the error condition the error item string can describe a rule **<r>**, a variable **<v>** or an (internal) error status **<e>**. DB file errors refer to problems accessing the **Rule System** database file **brules.vdb** in the BAE programs directory. Internal errors usually refer to **Rule System** implementation gaps and should be reported to Bartels.

See also

Functions **icd_getrulecnt**, **icd_getrulename**, **icd_rulefigatt**, **icd_rulefigdet**, **icd_ruleplanatt**, **icd_ruleplandet**, **icd_rulequery**; **Neural Rule System** and Rule System Compiler..

icd_rulefigatt - Attach rule(s) to figure list element (ICD)

Synopsis

```
int icd_rulefigatt(           // Returns nonzero on error
      index I_FIGURE;         // Figure list element index
      void;                   // Rule name string or rule name list array
      );
```

Description

The **icd_rulefigatt** function is used to attach a *new* set of name-specified rules to the figure list element specified with the first function parameter. Either a single rule name (i.e., a value of type **string**) or a set of rule names (i.e., an array of type **string**) can be specified with the second function parameter. Note that any rules previously attached to the figure list element are detached before attaching the new rule set. The function returns zero on success or nonzero on error. The **icd_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **icd_rulefigatt** function.

See also

Functions **icd_getrulecnt**, **icd_getrulename**, **icd_ruleerr**, **icd_rulefigdet**, **icd_ruleplanatt**, **icd_ruleplandet**, **icd_rulequery**; **Neural Rule System** and Rule System Compiler..

icd_rulefigdet - Detach rules from figure list element (ICD)

Synopsis

```
int icd_rulefigdet(           // Returns nonzero on error
      index I_FIGURE;         // Figure list element index
      );
```

Description

The **icd_rulefigdet** function is used to detach *all* currently attached rules from the figure list element specified with the function parameter. The function returns zero on success or nonzero on error. The **icd_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **icd_rulefigdet** function.

See also

Functions **icd_getrulecnt**, **icd_getrulename**, **icd_ruleerr**, **icd_rulefigatt**, **icd_ruleplanatt**. **icd_ruleplandet**, **icd_rulequery**; **Neural Rule System** and Rule System Compiler..

icd_ruleplanatt - Attach rule(s) to currently loaded element (ICD)

Synopsis

```
int icd_ruleplanatt(          // Returns nonzero on error
      void;                   // Rule name string or rule name list array
      );
```

Description

The **icd_ruleplanatt** function is used to attach a *new* set of name-specified rules to the currently loaded element. Either a single rule name (i.e., a value of type **string**) or a set of rule names (i.e., an array of type **string**) can be specified with the function parameter. Note that any rules previously attached to the current element are detached before the new rule set is attached. The function returns zero on success or nonzero on error. The **icd_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **icd_ruleplanatt** function.

See also

Functions **icd_getrulecnt**, **icd_getrulename**, **icd_ruleerr**, **icd_rulefigatt**, **icd_rulefigdet**, **icd_ruleplandet**, **icd_rulequery**; **Neural Rule System** and Rule System Compiler..

icd_ruleplandet - Detach rules from currently loaded element (ICD)

Synopsis

```
int icd_ruleplandet(          // Returns nonzero on error
      );
```

Description

The **icd_ruleplandet** function to detach *all* currently attached rules from the currently loaded element. The function returns zero on success or nonzero on error. The **icd_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **icd_ruleplandet** function.

See also

Functions **icd_getrulecnt**, **icd_getrulename**, **icd_ruleerr**, **icd_rulefigatt**, **icd_rulefigdet**, **icd_ruleplanatt**, **icd_rulequery**; **Neural Rule System** and Rule System Compiler..

icd_rulequery - Perform rule query on specific object (ICD)

Synopsis

```
int icd_rulequery(            // Returns hit count or (-1) on error
      int;                    // Object class code
      int;                    // Object ident code (int or index type)
      string;                 // Subject name
      string;                 // Predicate name
      string;                 // Query command string
      & void;                 // Query result
      []                      // Optional query parameters of requested type
      );
```

Description

The **icd_rulequery** function is used to perform a rule query on a specific object. The object can be the currently loaded element (object class code 0 with **int** value 0 passed for the object ident code), a figure list element of the currently loaded element (object class code 1 with valid **I_FIGURE** index type value passed for the object ident code), or a pool list element (object class code 2 with valid **I_POOL** index type value passed for the object ident code). The rule query function requires a rule subject, a rule predicate and a query command string to be specified with the corresponding function parameters. The query command string can contain one query operator and a series of value definition operators. The following query operators are implemented:

?d	for querying **int** values
?f	for querying **double** values
?s	for querying **string** values

The query operator can optionally be preceded with one of the following selection operators:

+	for selecting the maximum of all matching values
-	for selecting the minimum of all matching values

The **+** operator is used on default (e.g., when omitting the selection operator). The rule query resulting value is passed back to the caller with the query result parameter. This means that the query result parameter data type must comply with the query operator (**int** for **?d**, **double** for **?f**, **string** for **?s**). The query command string can also contain a series of value definition operators such as:

%d	for specifying **int** values
%f	for specifying **double** values
%s	for specifying **string** values

Each value definition parameter is considered a placeholder for specific data to be passed with optional parameters. Note that these optional parameters must comply with the query command in terms of specified sequence and data types. The **icd_rulequery** function returns a (non-negative) hit count denoting the number of value set entries matched by the query. The function returns (-1) on error. The **icd_ruleerr** function can be used to determine the error reason after an unsuccessful call of the **icd_rulequery** function.

Examples

With the rule

```
rule somerule
{
    subject subj
    {
        pred := ("A", 2);
        pred := ("A", 4);
        pred := ("B", 1);
        pred := ("C", 3);
        pred := ("B", 6);
        pred := ("D", 5);
        pred := ("D", 6);
        pred := ("A", 3);
    }
}
```

defined and attached to the currently loaded element, the **icd_rulequery** call

```
    hitcount = icd_rulequery(0,0,"subj","pred","%s ?d",intresult,"A") ;
```

sets the **int** variable **hitcount** to 3 and the **int** variable **intresult** to 4, whilst a call such as

```
    hitcount = icd_rulequery(0,0,"subj","pred","-?s %d",strresult,6) ;
```

sets **hitcount** to 2 and **string** variable **strresult** to **B**.

See also

Functions **icd_getrulecnt**, **icd_getrulename**, **icd_ruleerr**, **icd_rulefigatt**, **icd_rulefigdet**, **icd_ruleplanatt**, **icd_ruleplandet**; **Neural Rule System** and Rule System Compiler..

icd_scanall - Scan all IC Design figure list elements (ICD)

Synopsis

```
int icd_scanall(              // Returns scan status
      double;                 // Scan X offset (STD2)
      double;                 // Scan Y offset (STD2)
      double;                 // Scan rotation angle (STD3)
      int [0,1];              // Element in workspace flag (STD10)
      int [0,1];              // Connectivity scan allowed flag:
                              //    0 = no scan allowed
                              //    1 = scan allowed
      * int;                  // Macro callback function
      * int;                  // Polygon callback function
      * int;                  // Path callback function
      * int;                  // Text callback function
      * int;                  // Layer check function
      * int;                  // Level check function
      );
```

Description

The **icd_scanall** function scans all figure list elements placed on the currently loaded IC Design element with all hierarchy levels. User-defined callback functions are activated automatically according to the currently scanned element type. If a certain callback function should not be referenced, then the corresponding parameter must be set to the keyword **NULL**. The return value of **icd_scanall** is nonzero on invalid parameter specifications, or if one of the referenced user functions has returned a scan error status.

Macro callback function

```
int macrofuncname(
      index I_MACRO macro,    // Macro index
      index I_POOL pool,      // Pool element index
      int macinws,            // Macro in workspace flag (STD10)
      string refname,         // Macro Reference name
      index I_LEVEL level,    // Macro signal level
      int stkcnt              // Macro stack depth
      )
{
      // Macro callback function statements
      :
      return(contscan);
}
```

The **icd_maccoords** function can be used for determining the macro placement coordinates. The return value of the macro callback function must be 1 for continue scan, 0 for stop scan or (-1) on error.

Polygon callback function

```
int polyfuncname(
      index I_POLY poly,      // Polygon index
      int layer,              // Polygon layer (ICD1)
      int polyinws,           // Polygon in workspace flag (STD10)
      int tree,               // Polygon tree number or (-1)
      index I_LEVEL level     // Polygon signal level
      )
{
      // Polygon callback function statements
      :
      return(errstat);
}
```

The return value of the polygon callback function must be zero if scan ok or nonzero on error.

Path callback function

```
int pathfuncname(
      index I_LINE path,      // Path index
      int layer,              // Path layer (ICD1)
      int pathinws,           // Path in workspace flag (STD10)
      index I_LEVEL level     // Path signal level
      )
{
      // Path callback function statements
      :
      return(errstat);
}
```

The return value of the path callback function must be zero if scan ok or nonzero on error.

Text callback function

```
int textfuncname(
      index I_TEXT text,      // Text index
      double x,               // Text X coordinate (STD2)
      double y,               // Text Y coordinate (STD2)
      double angle,           // Text rotation angle (STD3)
      int mirr,               // Text mirror mode (STD14)
      int layer,              // Text layer (ICD1)
      double size,            // Text size (STD2)
      string textstr,         // Text string
      int textinws            // Text in workspace flag (STD10)
      )
{
      // Text callback function statements
      :
      return(errstat);
}
```

The return value of the text callback function must be zero if scan ok or nonzero on error.

Layer check function

```
int laycheckfuncname(
      int layer,              // Scanned layer (ICD1)
      int class               // Element class (STD1)
      )
{
      // Layer check function statements
      :
      return(contscan);
}
```

The return value of the layer check function must be 1 for continue scan, 0 for stop scan or (-1) on error. The scan process can be accelerated considerably if restricted to the interesting layers with this function.

Level check function

```
int levcheckfuncname(
      index I_LEVEL level     // Scanned signal level
      )
{
      // Level check function statements
      :
      return(contscan);
}
```

The return value of the level check function must be 1 for continue scan, 0 for stop scan or (-1) on error. The scan process can be accelerated considerably if restricted to interesting signal levels with this function.

See also

Functions **icd_maccoords**, **icd_scanfelem**, **icd_scanpool**.

icd_scanfelem - Scan IC Design figure list element (ICD)

Synopsis

```
int icd_scanfelem(            // Returns scan status
      index I_FIGURE;         // Figure list element index
      double;                 // Scan X offset (STD2)
      double;                 // Scan Y offset (STD2)
      double;                 // Scan rotation angle (STD3)
      int [0,1];              // Element in workspace flag (STD10)
      int [0,1];              // Connectivity scan allowed flag:
                              //    0 = no scan allowed
                              //    1 = scan allowed
      * int;                  // Macro callback function
      * int;                  // Polygon callback function
      * int;                  // Path callback function
      * int;                  // Text callback function
      * int;                  // Layer check function
      * int;                  // Level check function
      );
```

Description

The **icd_scanfelem** function scans the specified **IC Design** figure list element with all hierarchy levels. User-defined callback functions for the currently scanned element type are automatically activated. If a certain callback function should not be referenced, then the corresponding parameter must be set to the keyword **NULL**. The return value of **icd_scanfelem** is nonzero on invalid parameter specifications or if one of the referenced user functions has returned a scan error status. See **icd_scanall** for the scan function definitions.

See also

Functions **icd_maccoords**, **icd_scanall**, **icd_scanpool**.

icd_scanpool - Scan IC Design pool element (ICD)

Synopsis

```
int icd_scanpool(             // Returns scan status
      void;                   // Pool element index
      double;                 // Scan X offset (STD2)
      double;                 // Scan Y offset (STD2)
      double;                 // Scan rotation angle (STD3)
      int [0,1];              // Element in workspace flag (STD10)
      int [0,1];              // Connectivity scan allowed flag:
                              //    0 = no scan allowed
                              //    1 = scan allowed
      * int;                  // Macro callback function
      * int;                  // Polygon callback function
      * int;                  // Path callback function
      * int;                  // Text callback function
      * int;                  // Drill callback function
      * int;                  // Layer check function
      * int;                  // Level check function
      );
```

Description

The **icd_scanpool** function scans the specified **IC Design** pool element with all hierarchy levels. User-defined callback functions for the currently scanned element type are automatically activated. If a certain callback function should not be referenced, then the corresponding parameter must be set to the keyword **NULL**. The return value of **icd_scanpool** is nonzero on invalid parameter specifications or if one of the referenced user functions has returned a scan error status. See **icd_scanall** for the callback function definitions.

See also

Functions **icd_maccoords**, **icd_scanall**, **icd_scanfelem**.

icd_stdlayname - IC Design setup standard layer name (ICD)

Synopsis

```
string icd_stdlayname(        // Returns layer name
      int [0,99];             // Layer number (ICD1)
      );
```

Description

The **icd_stdlayname** function returns the layer name defined for the given layer number (ICD1) in the BAE IC setup file.

icd_stdpinlay - IC Design setup standard pin layer (ICD)

Synopsis

```
int icd_stdpinlay(            // Returns layer number (ICD1)
      );
```

Description

The **icd_stdpinlay** function returns the standard pin layer number defined in the BAE IC setup file for GDS input.

icd_vecttext - Vectorize IC Design text (ICD)

Synopsis

```
int icd_vecttext(             // Returns status
      double;                 // Text X coordinate (STD2)
      double;                 // Text Y coordinate (STD2)
      double;                 // Text rotation angle (STD3)
      int [0,1];              // Text mirror mode (STD14)
      double ]0.0,[;          // Text size (STD2)
      int [0,1];              // Text physical flag:
                              //    0 = logical
                              //    1 = physical
      int [0,2];              // Layer mirror mode:
                              //    0 = mirror off
                              //    1 = mirror X
                              //    2 = mirror Y
      int [0,[;               // Text style
      string;                 // Text string
      * int;                  // Text vectorize function
      );
```

Description

The **icd_vecttext** function vectorizes the specified text using the currently loaded text font. The text vectorize user function is automatically called for each text segment. The function returns nonzero if invalid parameters have been specified or if the referenced user function returns nonzero.

Text vectorize function

```
int vecfuncname(
      double x1,              // Start point X coordinate (STD2)
      double y1,              // Start point Y coordinate (STD2)
      double x2,              // End point X coordinate (STD2)
      double y2               // End point Y coordinate (STD2)
      )
{
      // Text vectorize function statements
      :
      return(errstat);
}
```

The return value of the text vectorize function must be zero if scan ok or nonzero on error.

C.5.2 Chip Editor Functions

The following **User Language** system functions are assigned to caller type CED; i.e., they can be called from the **Chip Editor** interpreter environment of the **Bartels AutoEngineer**:

ced_asklayer - CED layer selection (CED)

Synopsis

```
int ced_asklayer(            // Returns status
      & int;                 // Returns selected layer (ICD1)
      );
```

Description

The **ced_asklayer** function activates a **Chip Editor** layer selection menu. The function returns zero if a valid layer has been selected or (-1) if the layer selection has been aborted.

ced_delelem - Delete CED figure list element (CED)

Synopsis

```
int ced_delelem(             // Returns status
      & index I_FIGURE;      // Element
      );
```

Description

The **ced_delelem** function deletes the given figure list element from the figure list. The function returns zero if the element was successfully deleted or nonzero on error.

Warning

This function changes the current figure list and should be used carefully in `forall` loops for iterating **I_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

See also

Function **ced_drawelem**.

ced_drawelem - Redraw CED figure list element (CED)

Synopsis

```
void ced_drawelem(
      index I_FIGURE;        // Element
      int [0, 4];            // Drawing mode (STD19)
      );
```

Description

The **ced_drawelem** function updates the display of the given figure list element using the specified drawing mode.

See also
Function **ced_delelem**.

ced_elemangchg - Change CED figure list element rotation angle (CED)

Synopsis

```
int ced_elemangchg(           // Returns status
      & index I_FIGURE;       // Element
      double;                 // New rotation angle (STD3)
      );
```

Description

The **ced_elemangchg** function changes the rotation angle of the given figure list element. The rotation angle must be in radians. The function returns zero if the element has been successfully rotated, (-1) if the figure list element is invalid or (-2) if the figure list element cannot be rotated.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **I_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ced_elemfixchg - Change CED figure list element fixed flag (CED)

Synopsis

```
int ced_elemfixchg(           // Returns status
      & index I_FIGURE;       // Element
      int [0,1];              // New fixed flag (STD11)
      );
```

Description

The **ced_elemfixchg** function changes the fixed flag of the given figure list element. A fixed flag value of 0 resets the element fixed flag, a fixed flag value of 1 sets the element fixed flag. The function returns zero if the element fixed flag has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element cannot be fixed.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **I_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ced_elemgrpchg - Change CED figure list element group flag (CED)

Synopsis

```
int ced_elemgrpchg(           // Returns status
      index I_FIGURE;         // Element
      int [0,2];              // New group flag (STD13)
      );
```

Description

The **ced_elemgrpchg** function changes the group flag of the given figure list element. A group flag value of 0 deselects the element, a group flag value of 1 selects the element. The function returns zero if the element group flag has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element cannot be selected to a group.

ced_elemlaychg - Change CED figure list element layer (CED)

Synopsis

```
int ced_elemlaychg(           // Returns status
      & index I_FIGURE;       // Element
      int;                    // New layer (ICD1)
      );
```

Description

The **ced_elemlaychg** function changes the layer of the given figure list element. The layer can be set for polygons, traces and texts. The function returns zero if the element layer has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element layer cannot be set.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **I_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ced_elemmirrchg - Change CED figure list element mirror mode (CED)

Synopsis

```
int ced_elemmirrchg(          // Returns status
      & index I_FIGURE;       // Element
      int [0,2];              // New mirror mode (STD14|ICD3)
      );
```

Description

The **ced_elemmirrchg** function changes the mirror mode of the given figure list element. The mirror mode can be set for polygons, texts, named and unnamed references. The function returns zero if the element mirror mode has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element mirror mode cannot be set.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **I_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ced_elemposchg - Change CED figure list element position (CED)

Synopsis

```
int ced_elemposchg(           // Returns status
      & index I_FIGURE;       // Element
      double;                 // New X coordinate (STD2)
      double;                 // New Y coordinate (STD2)
      );
```

Description

The **ced_elemposchg** function changes the position of the given figure list element. Polygons and/or traces are replaced to set the first point of the polygon/trace to the specified position. The function returns zero if the element has been successfully repositioned, (-1) if the figure list element is invalid or (-2) if the figure list element position cannot be set.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **I_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ced_elemsizechg - Change CED figure list element size (CED)

Synopsis

```
int ced_elemsizechg(          // Returns status
      & index I_FIGURE;       // Element
      double;                 // New size (STD2)
      );
```

Description

The **ced_elemsizechg** function changes the size of the given figure list element. The size can be changed for texts, traces, named and unnamed references. For traces, a trace width change is performed. For named and unnamed references, the size specifies the scaling factor. The function returns zero if the element size has been successfully changed, (-1) if the figure list element is invalid or (-2) if the figure list element size cannot be set.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **I_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ced_getlaydispmode - Get CED layer display mode (CED)

Synopsis

```
int ced_getlaydispmode(       // Returns display mode (ICD9)
      int [0,99];             // Layer (ICD1)
      );
```

Description

The **ced_getlaydispmode** function returns the layer display mode for the given layer in the **Chip Editor** or (-1) on error.

ced_getmincon - Get CED Mincon function type (CED)

Synopsis

```
int ced_getmincon(            // Returns Mincon function type (ICD10)
      );
```

Description

The **ced_getmincon** function returns the currently active **Chip Editor** Mincon function type, i.e., the airline display mode (ICD10).

ced_getpathwidth - Get CED path standard widths (CED)

Synopsis

```
void ced_getpathwidth(
      & double;               // Returns small standard width (STD2)
      & double;               // Returns wide standard width (STD2)
      );
```

Description

The **ced_getpathwidth** function returns with its parameters the currently active **Chip Editor** standard widths for small and wide traces.

ced_getpickpreflay - Get CED pick preference layer (CED)

Synopsis

```
int ced_getpickpreflay(       // Returns pick preference layer (ICD1)
      );
```

Description

The **ced_getpickpreflay** function returns the currently active **Chip Editor** pick preference layer for element selection (ICD1).

ced_getwidedraw - Get CED wide line display start width (CED)

Synopsis

```
double ced_getwidedraw(       // Returns width value (STD2)
      );
```

Description

The **ced_getwidedraw** function returns the current **Chip Editor** wide line display start width, i.e., the minimum trace width for displaying traces like filled polygons.

ced_groupselect - CED group selection (CED)

Synopsis

```
int ced_groupselect(          // Number of changes or (-1) on error
      int [0,3];              // Element selection type:
                              //    0 = select by element type
                              //    1 = select by element layer
                              //    2 = select by element fixed flag
                              //    3 = select by element visibility
      int;                    // Element selection value according to type:
                              //    element type (0|ICD5) for selection type 0
                              //    element layer (ICD1) for select. type 1
                              //    element fixed flag (STD11) for
                              //     selection type 2
                              //    element visible flag (0|1) for
                              //     selection type 3
      int [0,2];              // New group flag (STD13)
      );
```

Description

The **ced_groupselect** function changes the group flag of all elements of the specified type and/or value. The function returns the number of elements (de)selected or (-1) on error (i.e., on invalid and/or incompatible parameter specifications). Element selection value zero for element type selection is used for selecting elements of *any* type.

Warning

Internal **IC Design** element types such as the standard via definition(s) are excluded from group (de)selections with **ced_groupselect** to prevent from unintentionally modifying and/or deleting such elements and/or definitions when subsequently using other group functions.

ced_highlnet - Set CED net highlight mode (CED)

Synopsis

```
int ced_highlnet(             // Returns status
      int [0,[;               // Net tree number
      int [0,1];              // Highlight mode (0 = off, 1 = on)
      );
```

Description

The **ced_highlnet** function sets the highlight mode of the net specified by the given net tree number. A highlight mode value of 1 highlight the net, a highlight mode value of 0 de-highlights the net. The function returns nonzero if an invalid net tree number and/or highlight mode value has been specified.

ced_layergrpchg - Select CED group by layer (CED)

Synopsis

```
int ced_layergrpchg(          // Number of elements
      int [0,[;               // Layer number (ICD1)
      int [0,2];              // New group flag (STD13)
      );
```

Description

The **ced_layergrpchg** function changes the group flag of all elements placed on the specified layer. The function returns the number of elements (de)selected or (-1) on error.

ced_partaltmacro - Change CED net list part cell type (CED)

Synopsis

```
int ced_partaltmacro(         // Returns status
      string;                 // Part name
      string;                 // New part cell type name
      );
```

Description

The **ced_partaltmacro** function changes the cell type of the given net list part. The function returns nonzero if the part cell type has been successfully changed, (-1) for invalid input parameters, (-2) if the specified package does not contain all pins referenced by the part in the net list (cell is changed anyway), (-3) if the specified part does not exist in the net list, (-4) if the new cell type isn't allowed for this part, (-5) if the new cell macro couldn't be loaded, (-6) if the new cell couldn't be copied to the job file or (-7) on multiple cell change requests (e.g., **a** to **b** and then **b** to **c**) in one program run.

Warning

It is strongly recommended not to use this function in **I_CPART** index loops since the current **I_CPART** index variables are invalid after calling **ced_partaltmacro**.

ced_partnamechg - Change CED net list part name (CED)

Synopsis

```
int ced_partnamechg(          // Returns status
      string;                 // Old part name
      string;                 // New part name
      );
```

Description

The **ced_partnamechg** function changes the name of a net list part. The function returns nonzero if the part name has been successfully changed, (-1) for invalid input parameters, (-2) if the specified part is not yet placed, (-3) if the specified part does not exist in the net list, (-4) if the new name exists already or (-5) on multiple name change requests (e.g., **a** to **b** and then **b** to **c**) in one program run.

Warning

This function changes the net list and therefore requires a Backannotation It is strongly recommended not to use this function in **I_CPART** index loops since the current **I_CPART** index variables are invalid after calling **ced_partnamechg**.

ced_pickelem - Pick CED figure list element (CED)

Synopsis

```
int ced_pickelem(             // Returns status
      & index I_FIGURE;       // Returns picked element
      int [1,8];              // Pick element type (ICD5 except 6)
      );
```

Description

The **ced_pickelem** function activates an interactive figure list element pick request (with mouse). The required pick element type is specified with the second parameter. The picked figure list element index is returned with the first parameter. The function returns zero if an element has been picked or (-1) if no element of the required type has been found at the pick position.

ced_setlaydispmode - Set CED layer display mode (CED)

Synopsis

```
int ced_setlaydispmode(       // Returns status
      int [0,99];             // Layer (ICD1)
      int [0,127];            // Display mode (ICD9)
      );
```

Description

The **ced_setlaydispmode** function sets the layer display mode for the given layer in the **Chip Editor**. The function returns nonzero if the display mode couldn't be set.

ced_setmincon - Set CED Mincon function type (CED)

Synopsis

```
int ced_setmincon(            // Returns status
      int [0,8];              // Required Mincon function type (ICD10)
      );
```

Description

The **ced_setmincon** function sets the currently active **Chip Editor** Mincon function type, i.e., the airline display mode (ICD10). The function returns nonzero if an invalid Mincon function type value has been specified.

ced_setpathwidth - Set CED path standard width (CED)

Synopsis

```
int ced_setpathwidth(         // Returns status
      double ]0.0,[;          // Required small path width (STD2)
      double ]0.0,[;          // Required wide path width (STD2)
      );
```

Description

The **ced_setpathwidth** function sets the currently active **Chip Editor** standard widths for small and wide traces. The function returns nonzero if invalid an invalid width value has been specified.

ced_setpickpreflay - Set CED pick preference layer (CED)

Synopsis

```
int ced_setpickpreflay(       // Returns status
      int;                    // Required pick preference layer (ICD1)
      );
```

Description

The **ced_setpickpreflay** function sets the currently active **Layout Editor** pick preference layer for element selection (ICD1). The function returns nonzero if an invalid pick preference layer has been specified.

ced_setwidedraw - Set CED wide line display start width (CED)

Synopsis

```
int ced_setwidedraw(          // Returns status
      double ]0.0,[;          // Required width value (STD2)
      );
```

Description

The **ced_setwidedraw** function sets the current **Chip Editor** wide line display start width, i.e., the minimum trace width for displaying traces like filled polygons. The function returns nonzero if an invalid width value is specified.

ced_storepart - Place CED part or pin (CED)

Synopsis

```
int ced_storepart(            // Returns status
      string;                 // Reference name
      string;                 // Library symbol name
      double;                 // X coordinate (STD2)
      double;                 // Y coordinate (STD2)
      double;                 // Rotation angle (STD3)
      double;                 // Scaling factor
      int [0,1];              // Mirror mode (STD14)
      );
```

Description

The **ced_storepart** function stores a cell (or pin) with the given placement parameters to the currently loaded IC design (or cell) element. The next unplaced net list part is used if an empty string is passed for the reference name. The function returns zero if the part has been successfully placed, (-1) on wrong environment or missing/invalid parameters, (-2) if all parts are placed already, (-3) if the specified part is placed already, (-4) if the part cannot be loaded, (-5) if the part pins do not match the net list specifications or (-6) if the part data could not be copied to the current job file.

Warning

This function changes the current figure list and should be used carefully in `forall` loops for iterating **I_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ced_storepath - Place CED internal polygon as path (CED)

Synopsis

```
int ced_storepath(            // Returns status
      int [0,99];             // Path layer (ICD1)
      double ]0.0,[;          // Path width (STD2)
      );
```

Description

The **ced_storepath** function generates a trace on the currently loaded **IC Design** using the specified placement parameters. The trace polygon points are taken from the internal polygon point list previously stored with **bae_storepoint**. The function returns zero if the trace has been successfully generated, (-1) on invalid environment, (-2) on missing and/or invalid parameters or (-3) if the point list is invalid.

Warning

This function changes the current figure list and should be used carefully in `forall` loops for iterating **I_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ced_storepoly - Place CED internal polygon (CED)

Synopsis

```
int ced_storepoly(            // Returns status
      int;                    // Polygon layer (ICD1)
      int [1,4];              // Polygon type (ICD4)
      int [0,2];              // Mirror mode (ICD3)
      );
```

Description

The **ced_storepoly** function generates a polygon on the currently loaded **IC Design** element using the specified placement parameters. The polygon points are taken from the internal polygon point list previously stored with **bae_storepoint**. The function returns zero if the polygon has been successfully generated, (-1) on invalid environment, (-2) on missing and/or invalid parameters or (-3) if the point list is not valid for the specified polygon type.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **I_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.

ced_storetext - Place CED text (CED)

Synopsis

```
int ced_storetext(            // Returns status
      string;                 // Text string
      double;                 // Text X coordinate (STD2)
      double;                 // Text Y coordinate (STD2)
      double;                 // Text rotation angle (STD3)
      double ]0.0,[;          // Text size (STD2)
      int;                    // Text layer (ICD1)
      int [0,1];              // Text mirror mode (STD14)
      );
```

Description

The **ced_storetext** function generates a text on the currently loaded **IC Design** element using the specified placement parameters. The function returns nonzero on wrong environment or missing/invalid parameters.

Warning

This function changes the current figure list and should be used carefully in **forall** loops for iterating **I_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops. The input text string can be stored to a maximum of up to 40 characters; longer strings cause the function to return with an invalid parameter error code.

ced_storeuref - Place CED unnamed reference (via or subpart) (CED)

Synopsis

```
int ced_storeuref(            // Returns status
      string;                 // Library symbol name
      double;                 // Reference X coordinate (STD2)
      double;                 // Reference Y coordinate (STD2)
      double;                 // Reference rotation angle (STD3)
      double;                 // Reference scaling factor
      int [0,1];              // Reference mirror (STD14)
      );
```

Description

The **ced_storeuref** function stores an unnamed reference (via or subpart) with the given placement parameters to the currently loaded layout element (layout or part). For vias, the reference mirror mode, the reference scaling factor, and the rotation angle are ignored. The function returns zero if the reference has been successfully placed, (-1) on wrong environment or missing/invalid parameters, (-2) if the reference cannot be loaded or (-3) if the reference data could not be copied to the current job file.

Warning

This function changes the current figure list and thus should be used carefully in **forall** loops for iterating **I_FIGURE** index variables to avoid unpredictable results on figure list access and prevent from running into endless loops.