Arbortext® IsoDraw®
Macro Language Reference

Arbortext IsoDraw Foundation 7.1 M020

Arbortext IsoDraw CADprocess 7.1 M020

December 2009

Parametric Technology Corporation, 140 Kendrick Street, Needham, MA 02494 USA

Contents

About This Guide

This *Arbortext IsoDraw Macro Language Reference* provides the information you need to write and edit macros in IsoDraw Macro Language (IML); a proprietary, lightweight programming language for automating tasks in Arbortext IsoDraw.

Prerequisite Knowledge

You need a basic understanding of Arbortext IsoDraw macros to use IML. See About Macros on page 21 to review these concepts. To learn more about using macros in Arbortext IsoDraw, see Macros Menu in the *Arbortext IsoDraw User's Reference*.

In addition, since IML is a programming language, some experience with simple object-oriented/event-driven programming or scripting would be helpful. However, you can still get started writing and editing macros using IML by copying the macro code examples in this guide and modifying them to suit your needs.

Organization of This Guide

This *Arbortext IsoDraw Macro Language Reference* is organized as follows:

Introduction on page 19	• Gives you basic information about creating, running, debugging, and managing macros using the Arbortext IsoDraw **Macros** menu and IML. (See About Macros on page 21.) • Covers the lexical structure of IML, such as syntax and naming rules for command statements, variables, keywords, and comments. (See Language Basics on page 29.)
Menu Commands on page 45	Describes macro commands that directly access features in Arbortext IsoDraw's

	menus. For example, Objects Menu on page 75 covers macro commands that correspond to options on the **Objects** menu.
Window Commands on page 89	Describes macro commands that directly access features in windows you can show or hide using Arbortext IsoDraw's **Window** menu. For example, Palette Window Toolbox on page 95 describes macro commands that correspond to tools in the **Palette** window.
3D and User Interaction Commands on page 127	• Covers commands you can execute in Arbortext IsoDraw CADprocess 3D mode if the current drawing contains 3D CAD data. (See 3D Commands on page 129.) • Contains useful commands for macro development tasks such as debugging, file handling, and running extensions. (See Further Macro Commands on page 139.) • Provides commands that enable macros to display messages, prompt for keyboard input, and respond to mouse events. (See Interacting with the User on page 153.)
Functions and Data Types on page 157	• Describes IML-supported functions that can evaluate mathematical expressions, interpret character strings, return or track time, apply logical conditions, and control program flow. (See Functions on page 159.) • Defines single-value data in IML used for integers, floating point numbers, character strings, and boolean (true/false) values. (See Simple Data Types on page 167.) • Defines multi-value data in IML; data types that have multiple properties with values that can be returned or set separately. (See Complex Data Types on page .)

Object Data Types on page 173	Describes four “objects” in IML; uniquely complex data types that have large numbers of properties. The objects’ names are: `document`, `element`, `layer`, and `application`. The first three objects return and set attributes for Arbortext IsoDraw documents, elements, and layers. The `application` object returns and sets preferences for both user interface functions and data exchange. “Sub data types” are also discussed in this section, since they can be used to access attributes of `document` or `application` objects.
Appendix on page 297	• Lists international names for IML object attributes. Use these names rather than language-specific attribute names to ensure that your macro will run regardless of the currently selected Arbortext IsoDraw language. (See International Names on page 299.) • Provides a list of IML file format names for use in the `EXPORT` and `PROCESS` commands. (See IML File Format Names on page 301.) • Lists CGM profiles and their corresponding numbers returned and set in the `app.cgm.profile` property of the Application object. (See CGM Profile Numbers and Names on page 303.)

Related Documentation

For more information on Arbortext IsoDraw products refer to the following documentation found in the Arbortext IsoDraw Help Center. Help Center includes both HTML and PDF versions of the documentation. Choose **Help ▸ Help Center** to access it.

Documentation	Description
Arbortext IsoDraw Release Notes	Information about new, changed, and deleted features in this Arbortext IsoDraw release.
Installing Arbortext IsoDraw	Installation and licensing information for Arbortext IsoDraw.
Drawing Basics Tutorial	Hands-on examples for learning Arbortext IsoDraw basic functions.

Documentation	Description
3D Mode Tutorial	Hands-on examples for learning 3D CAD data editing functions using Arbortext IsoDraw.
Arbortext IsoDraw User's Reference	Comprehensive guide to using the tools and functions in Arbortext IsoDrawproducts.
Arbortext IsoDraw Macro Language Reference	(This guide) Reference for writing macros that you can run in Arbortext IsoDraw.
Arbortext IsoDraw Data Exchange Reference	Instructions for importing and exporting graphics data in various formats to and from Arbortext IsoDraw.

Technical Support

Contact PTC Technical Support via the PTC Web site, phone, fax, or e-mail if you encounter problems using your product or the product documentation.

For complete details, refer to Contacting Technical Support in the *PTC Customer Service Guide*. This guide can be found under the Related Resources section of the PTC Web site at:

http://www.ptc.com/support/

The PTC Web site also provides a search facility for technical documentation of particular interest. To access this search facility, use the URL above and select Search the Knowledge Base.

You must have a Service Contract Number (SCN) before you can receive technical support. If you do not have an SCN, contact PTC Maintenance Department using the instructions found in your *PTC Customer Service Guide* under Contacting Your Maintenance Support Representative.

Documentation for PTC Products

You can access PTC product documentation using the following resources:

- Online Help Click **Help** from the user interface for online help available for the product.
- Product CD or Download All relevant PTC product documentation is included on the CD or in the download file for the product.
- Reference Documents Web Site Individual product manuals are available from the Download Reference Documents link of the PTC Web site at the following URL:

  http://www.ptc.com/support/
- Help Center Web Site A searchable product documentation knowledge base is available from the Help Center link of the PTC Web site at the following URL:

  http://www.ptc.com/support/

You must have a Service Contract Number (SCN) before you can access the Reference Documents or Help Center Web site. If you do not have an SCN, contact PTC Maintenance Department using the instructions found in your *PTC Customer Service Guide* under Contacting Your Maintenance Support Representative.

Global Services

PTC Global Services delivers the highest quality, most efficient and most comprehensive deployments of the PTC Product Development System including Pro/ENGINEER, Windchill, Arbortext and Mathcad. PTC's Implementation and Expansion solutions integrate the process consulting, technology implementation, education and value management activities customers need to be successful. Customers are led through Solution Design, Solution Development and Solution Deployment phases with the continuous driving objective of maximizing value from their investment.

Contact your PTC sales representative for more information on Global Services.

Comments

PTC welcomes your suggestions and comments on our documentation. You can submit your feedback to the following email address:

arbortext-documentation@ptc.com

Please include the following information in your email:

- Name
- Company
- Product
- Product Version
- Document or Online Help Topic Title
- Level of Expertise in the Product (Beginning, Intermediate, Advanced)
- Comments (including page numbers where applicable)

Documentation Conventions

This guide uses the following notational conventions:

- **Bold text** represents exact text that appears in the program's user interface. This includes items such as button text, menu selections, and dialog box elements. For example,

  Click **OK** to begin the operation.

- A right arrow represents successive menu selections. For example,

  Choose **File** ▸ **Print** to print the document.

- `Monospaced text` represents code, command names, file paths, or other text that you would type exactly as described. For example,

  At the command line, type `version` to display version information.

- *`Italicized monospaced text`* represents variable text that you would type. For example,

  `installation-dir``\custom\scripts\`

- *Italicized text* represents a reference to other published material. For example,

  If you are new to the product, refer to the *Getting Started Guide* for basic interface information.

Introduction

1 About Macros

This section provides basic information about Arbortext IsoDraw macros. It explains what an macro is, the various ways you can create, run, modify, and debug macros using the **Macros** menu and IML—and where and how macros are stored.

Macros and Macro Language

A macro is a small computer program stored in IML, the Arbortext IsoDraw Macro Language. You run macros inside Arbortext IsoDraw, typically to automate repetitive manual actions, such as selecting menu options, setting predefined views, or performing a series of graphical transformations.

For example: If you frequently perform tasks such as…

- Importing a third-party graphics file into Arbortext IsoDraw for editing and saving in native Arbortext IsoDraw ISO (`.iso`) file format
- Exporting an ISO file to a file in a third-party graphics format
- Batch-converting files in one graphics format to a different format

…you can create a macro that automatically configures import or export preference settings in Arbortext IsoDraw and performs the graphics conversion.

Note

If a macro only performs 2D graphics tasks, you can run it in either Arbortext IsoDraw Foundation or Arbortext IsoDraw CADprocess. If a macro performs 3D graphics tasks, it will only run successfully in Arbortext IsoDraw CADprocess. (In this guide, Arbortext IsoDraw refers to both Arbortext IsoDraw Foundation and Arbortext IsoDraw CADprocess unless otherwise noted.)

Most drawing tasks that can be done manually through the Arbortext IsoDraw interface can also be done using macros. Furthermore, programming logic structures in IML enable you to do more with macros than you can using the Arbortext IsoDraw interface alone.

What's In IML?

IML consists of the following:

- Commands that correspond to many Arbortext IsoDraw menu and toolbar functions.
- Objects and other data types with properties that match attributes, preferences, and option settings in Arbortext IsoDraw windows and dialog boxes.
- Program logic and flow constructs common to many high-level computer and scripting languages. (These enable familiar and useful programming techniques such as conditional looping and remote calls in Arbortext IsoDraw macros.)

When you create or edit a macro in IML, use these commands, data types, and program control features in statements that conform to the syntax rules and examples in this guide.

How are Macros Created and Updated?

You can create and update a macro two different ways:

- In Arbortext IsoDraw: record a sequence of tasks using the **Record macro** command on the **Macros** menu. Before recording, you must specify a macro name and a macro file to store it in. After recording, you can run the entire macro, or, choose **Debug macro** from the **Macros** menu and execute one macro line at a time while you observe

the results. (For more information, see Recording Macros on page 24 and Debugging Macros on page 26.)

- In a text editing program (such as Windows Notepad): Open a macro file and edit the macro code directly. When you're ready to test the macro, save the macro file, then load, run, and debug it in Arbortext IsoDraw. You can also use a text editor to edit recorded macros. (In fact, recording a macro first, then editing the code in a text editor is often easier than starting from scratch.) When you edit IML code in a macro file, be sure to follow the IML syntax and usage examples described in this guide. (For more information, see Editing Macro Files on page 28.

Macro File Structure

Macros are stored in ASCII text macro (`*.ism`) files. A macro file must contain at least one macro, and can contain more as shown in the example below.

Example

The macro file `Example.ism` listed below contains two macros named `Example1` and `Example2`.

Note

Comment lines in a macro file begin with a hash (#) mark and are not executed.

```
#Listing of macro file Example.ism:
#
#The macro Example1 performs 3D mode view-setting commands
MACRO Example1
   3D VIEW DIMETRIC_2
   3D HLRMODE WIREFRAME
   3D SETVIEW 903
END MACRO
#
#The macro Example2 prompts for a custom Pen name
#and checks to see if the name exists
MACRO Example2
   DEFINE penName AS String
   penName = Get String "Name of your new Pen?"
   IF (exists (activeDoc.Pens[penName]) = false) THEN
      Add Pen penName
   ELSE
      MESSAGE "Pen " + penName + " already exists!"
   END IF
END MACRO
```

Macro File Storage

Arbortext IsoDraw looks for macros in all macro (`*.ism`) files stored in the two folders below (subfolders included):

- *Arbortext-IsoDraw-install-path*\Program\Macros
- *Arbortext-IsoDraw-install-path*\User Profiles\username\Application Data\PTC\IsoDraw\Macros

The name of each macro found appears on the Arbortext IsoDraw **Macros** menu.

Updating the List of Macro Names

If you do any of the following file or editing operations while Arbortext IsoDraw is running, choose **Macros ▸ more macros…** to update the list of macro names in the **Macros** menu:

- Create a new macro file with one or more new macros in it
- Create a new macro in an existing macro file
- Rename a macro or macro file
- Move a macro to a different macro file
- Move a macro file to a different folder

Storing Macro Files on Shared Network Server

When Arbortext IsoDraw reloads macros, it will include any macros that Windows shortcuts in the `\Macros` folders point to. This enables you to, for example, store macro files for a workgroup in a shared folder on a network server.

Creating Macros

You can create an IML macro in either of the following ways:

- By recording the macro in Arbortext IsoDraw: Use options on the **Macros** menu to record a set of actions, assign them a macro name, and save them in a macro file.
- Editing a macro file in a text editor: Open a new or existing macro (`*.ism`) file in (for example) Windows Notepad. Add IML code to it using this guide as a reference—then save the file with an `.ism` extension in the `\Macros` folder.

After you create a macro, it appears in the list of macro names on the Arbortext IsoDraw **Macros** menu. If you create a macro using a text editor while Arbortext IsoDraw is running, select **Macros ▸ more macros…** to refresh the list.

Recording Macros

Note

This topic summarizes more detailed information on macro recording found in Macros Menu in the Arbortext IsoDraw User's Reference.

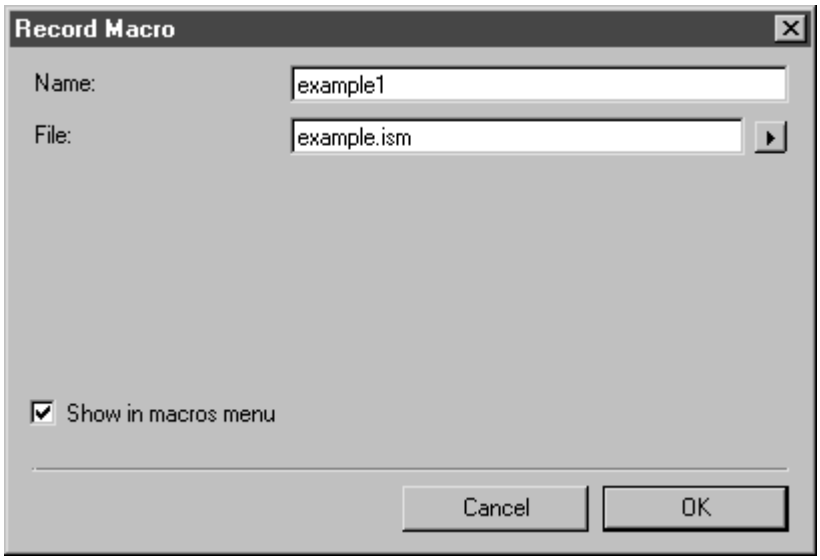
To record a macro in Arbortext IsoDraw you do the following:

- Use the **Macros** menu to record a set of actions.
- Assign a macro name to the recording.
- Save the macro name in a macro file.

For example:

1. Choose **File** ▸ **New** ▸ **Empty page**. Save the new ISO file as `example1.iso`.
2. Choose **Macros** ▸ **Macros** ▸ **Record macro**.

   The following dialog box appears.



3. Enter `example1` in the macro **Name** box.

   Note

   When you record a macro, you should choose a name that matches the macro's function.

4. Enter `example.ism` in the macro **File** box.

   Note

   One macro file can contain multiple macros.

5. Select **Show in macros menu** if you want the macro name you entered to appear in the **Macros** menu list
6. Click **OK** to save your entries and close the **Record Macro** dialog box.

   Macro recording begins immediately after dialog box closes.

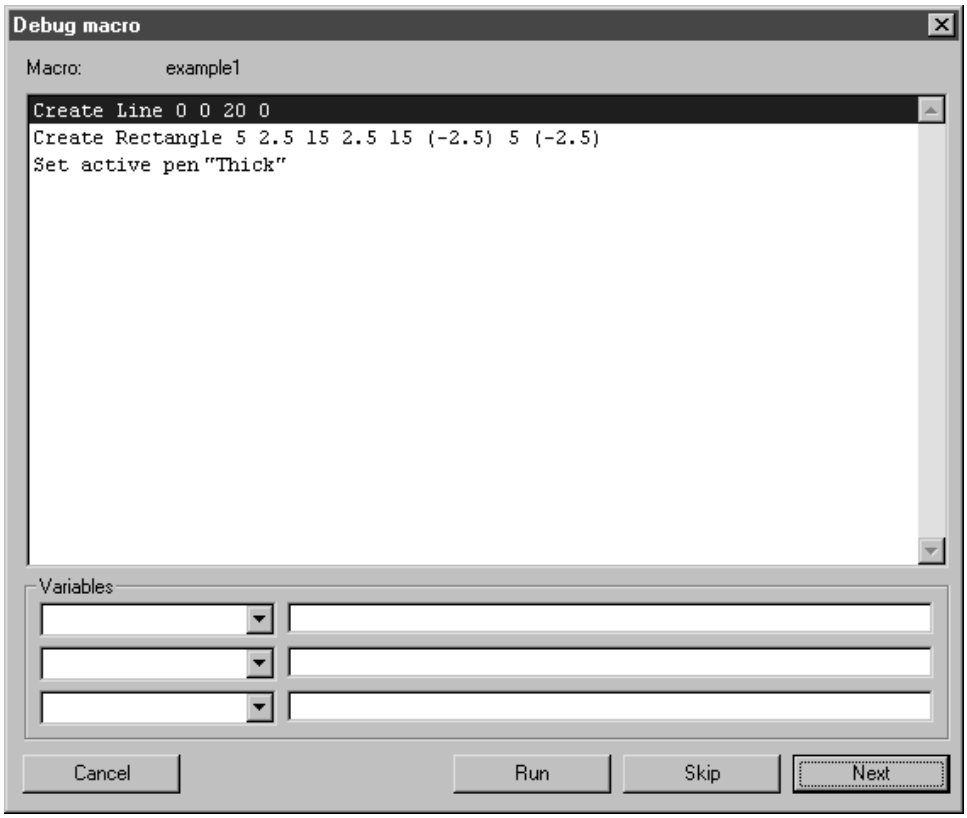
7. Perform the sequence of actions in Arbortext IsoDraw that you want the `example1` macro to perform, then choose **Macros** ▸ **Macros** ▸ **Stop recording**.

   Note

   *Selecting **Objects** ▸ **Edit Animation** pauses macro recording. No actions are recorded while **Edit Animation** is selected. To resume macro recording, close the **Edit Animation** or **Timeline** dialog box. (Closing one closes both.)*

8. The macro is complete. The macro name `example1` will now appear on the list of macros when you open the **Macros** menu. If you want to view the individual steps of the macro, select the **Debug macro** command, click on `example1` and click **OK** to confirm.

The macro steps should be displayed in the dialog box as illustrated below.



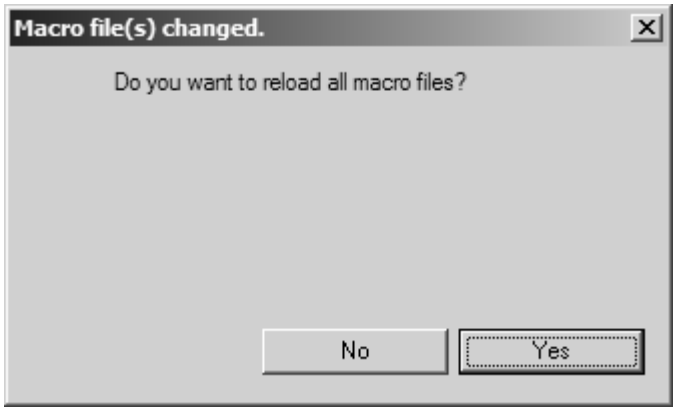
Debugging Macros

Note

This topic summarizes more detailed information on macro debugging found in Macros Menu in the Arbortext IsoDraw User's Reference.

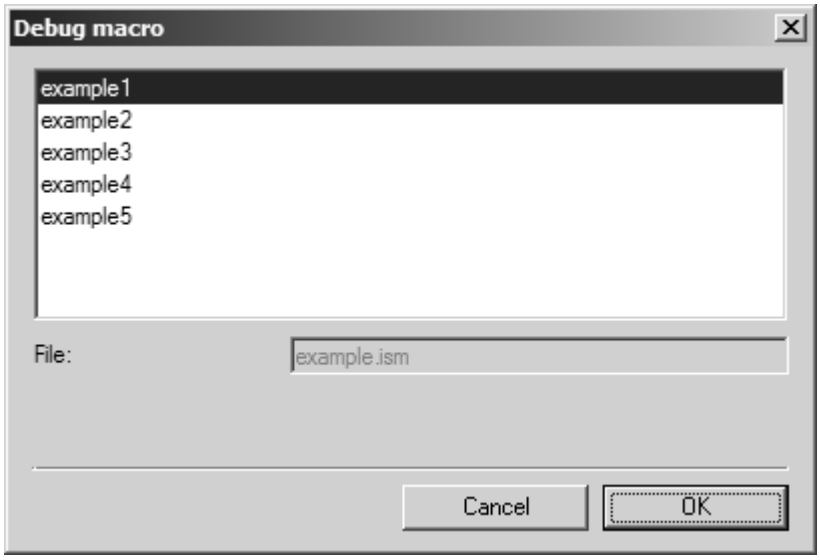
To debug a macro in Arbortext IsoDraw do the following:

1. Choose **Macros** ▸ **Macros** ▸ **Debug macro…**.
2. If the content of any macro file changed since you last opened the **Macros** menu, the message dialog box below appears and asks if you want to reload all macro files.



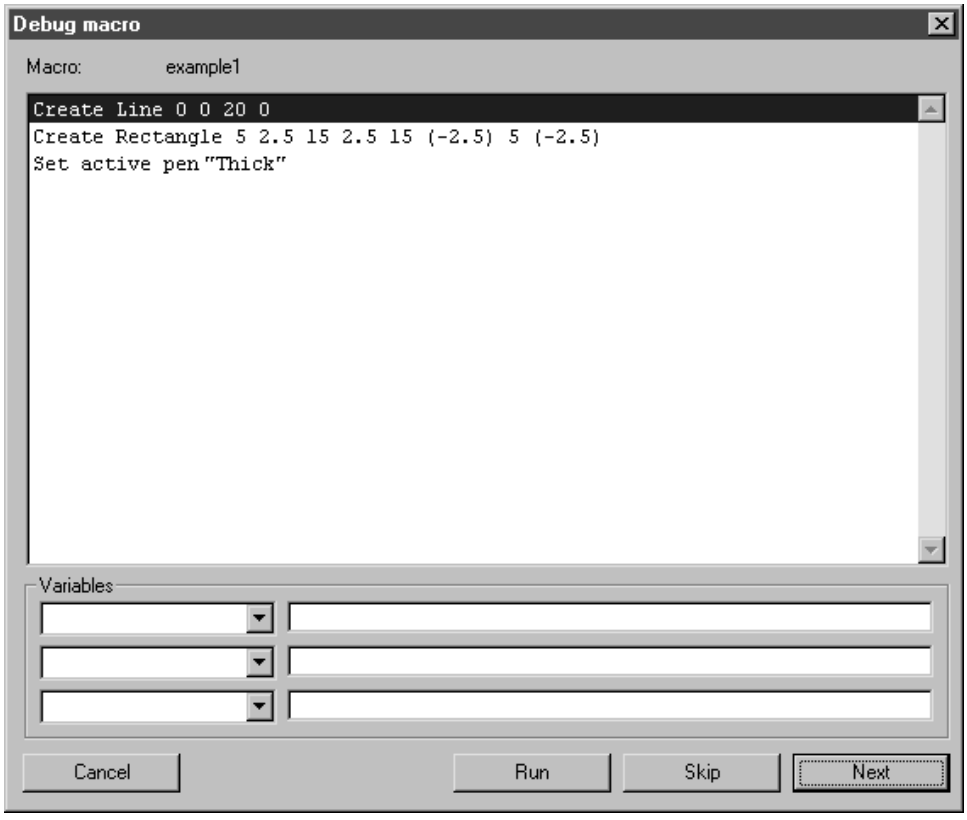
Click **Yes**.

3. The **Debug macro** dialog box opens showing the list of available macros.



Select the macro you want to debug, such as `example1`, then click **OK**.

4. The **Debug macro** dialog box shows the individual actions in the macro `example1`.



5. Use the **Skip** and **Next** buttons to step through the macro and debug it line-by-line.

   - Click **Next** to execute the currently selected line (only). If it contains a drawing action, you can see the result in the drawing window.

- Click **Skip** to go to the next line without executing the current line.
- If the currently selected line contains local or global variables, their names appear in the drop-down lists under **Variables** and their values appear in the text boxes. You can change variable values in these text boxes, but the changes are not saved in the macro file.

  Note

  To change variable values permanently, you must re-record the macro and select or enter the new values, or, you can edit the variable values in the macro file directly. (See Editing Macro Files on page 28.)
- Click **Run** to run the macro starting from the currently selected line. (The **Debug macro** dialog box closes before the macro runs.)

Editing Macro Files

Arbortext IsoDraw stores macro files in ASCII text format. You can use a Windows text editing program to edit the macros in macro files.

Note

If you use a Unix text editor, make sure to save the file with Windows-compatible CR-LF line breaks.

When you create a new macro file, save it in the folder *`Arbortext-IsoDraw-install-path`*`\Program\Macros` with the file name extension `*.ism`.

Built-in IML Limits

When you edit IML code in a macro file, do not exceed the built-in IML limits below. If you do, errors might occur when you save the macro file or run the macro.

Maximum length of a single macro code line	1024
Maximum length of string variable content	512
Maximum length of macro names	256
Maximum length of variable names	256
Maximum depth of function recursion	100
Maximum character length of single macro (excluding comments)	32766

2 Language Basics

Lexical Structure

The lexical structure of a programming language is the set of basic rules that governs how you write programs in that language. It is the lowest syntax level of the language and specifies such things as how variable names are written, what characters are used for comments, and how program statements are separated from one another.

Case Sensitivity

The names of user defined macros, subMacros, variables and keywords such as MESSAGE, CREATE LINE, WHILE, etc. are case-insensitive. These lines are equivalent:

```
#example 1:
MESSAGE "hello, world"
MESSAGE "hello, world"
MESSAGE "hello, world"

######################

#example 2:
DEFINE out AS string
DEFINE OUT AS string
```

Only string constants are case-sensitive:

```
#CORRECT:
Save "C:\temp\output.com" "CGM"

#INCORRECT:
Save "C:\temp\output.com" "cgm"
```

Statements and Line Breaks

A statement is a collection of code that performs a task. It can be as simple as a variable assignment or as complicated as a loop with several exit points. A simple example:

```
DEFINE name AS string
name = "Barney Gumble"
MESSAGE name
```

The Arbortext IsoDraw Macro Language uses line breaks to separate simple statements. Code blocks like `while`-loops or conditional tests must be terminated with an `end` statement.

```
IF ( a > b) THEN
   MESSAGE "hello world"
END IF
```

Line Continuation

Statements can be continued from one line to the next with the use of a backslash (\).

```
CREATE RECTANGLE 50.798 209.544 \
50.798 184.145 \
95.248 184.145 \
95.248 209.544
```

Spaces and Tabs

Every part of a statement that consists of more than one word has to be separated by a space:

```
CREATE LINE
APPEND LINE SEGMENT
CREATE CALLOUT
```

Multiple spaces have no effect on the macro. You can take advantage of this flexible formatting to make your code more readable (by lining up assignments, indenting, etc.):

CREATE LINE	125	678	254	223
CREATE LINE	658.225	568.554	874.974	336.889

Comments

Comments give useful information to people who read your code, but are ignored by Arbortext IsoDraw. Even if you are the only person who will ever read your code it can be beneficial to include comments. It will make the code easier to understand when reviewing it in the future.

When Arbortext IsoDraw encounters a hash mark (#) within the code, everything from that hash mark to the end of the line is recognized as a comment.

```
#the next line draws an ellipse
create ellipse 113.34 112.99 99 23.5 27
```

Literals

Literals are data values that appear directly in an Arbortext IsoDraw macro. The following are examples of literals in Arbortext IsoDraw:

```
100
off
23.987
(-27)
245,666
'Hello'
"good evening"
false
```

Identifiers

An identifier is used to name variables, macros and subMacros.

Variable Names

The first character of a variable name must be an ASCII letter (uppercase or lowercase). After the initial character the underscore character (_) and the digits 0-9 are also valid. Spaces are always invalid. Some examples of valid variable names:

```
george
positionX
y9
element_description
```

Macro and SubMacro Names

Macro and subMacro names are case-insensitive. After the keywords `MACRO` or `SUBMACRO`, letters (uppercase or lowercase), digits, special characters and spaces are allowed in any order you choose. Some examples of valid macro or subMacro names:

```
geoffrey
myLines
27 Ellipses FOR your pleasure
IsoDraw_4_ever
>>> !do not execute! <<<
```

Keywords

A keyword is a word reserved by the Arbortext IsoDraw Macro Language for its own functionality. Keywords must not be used as variable names. The following lists Arbortext IsoDraw macro keywords:

BREAK

DEFINE

ELSE

END

FOR

GLOBAL

IF

MACRO

RUN

STEP

SUBMACRO

THEN

TO

WHILE

All command names

Macro

A single macro file may contain several macros. This is useful for grouping different macros of the same topic.

Note

Macros cannot be nested.

Syntax

```
MACRO macroname[PROTECTED][NOT_IN_MENU]
END MACRO
```

PROTECTED	Optional parameter to protect the macro from being deleted using the **Delete Macro** dialog or being overwritten by a recording.
NOT_IN_MENU	Optional keyword that hides macro from the menu **Macros**. The macro can still be started through the **Run Macro** dialog.

Example

```
MACRO Do Something
   MESSAGE "I'm doing something"
END MACRO

###########################################

MACRO littleHelper NOT_IN_MENU
   #this macro does not appear in the
   #menu "macros"
END MACRO

###########################################

MACRO necessary PROTECTED
   #this macro can not be deleted using
   #the menu command "Delete macro"
END MACRO

###########################################

MACRO bothOptions PROTECTED NOT_IN_MENU
   #you can use both parameters in one macro
END MACRO
```

SubMacro

A single macro may consist of several subMacros.

Note

SubMacros can not be nested.

Syntax

```
SUBMACRO macroname [PROTECTED]
END SUBMACRO
```

PROTECTED	Optional parameter to protect the subMacro from being deleted using the **Delete Macro** dialog or being overwritten by a recording.

Example

```
SUBMACRO messenger
   MESSAGE "Here's your message"
END SUBMACRO
```

Variables

There are two types of variables in Arbortext IsoDraw macros:

local	Local variables have to be defined within a macro and can only be accessed within that macro.
global	Global variables can be accessed from all loaded macros within Arbortext IsoDraw. They have to be defined outside a macro with the keyword GLOBAL.

Example — Global

```
global gText AS string

MACRO globalsTest
   gText = "Hi, I'm a global."
   MESSAGE gText
END MACRO
```

Example — Local

```
MACRO localsTest
   DEFINE myInt   AS integer
   DEFINE myFloat AS float

   myInt   = 21
   myFloat = 2.77889

   MESSAGE myInt
   MESSAGE myFloat
END MACRO
```

Dispose

DISPOSE deletes the value and content of a previously defined variable, removes it from the list of defined variables, and frees the memory allocated to it. After the DISPOSE command executes, the variable can no longer be used. To use the variable again, redefine it first using the DEFINE command.

Syntax

`Dispose` *varname*

varname	The name of the previously defined variable to remove.

Example

```
# ...
DEFINE v AS string
v = el.info.attributes["XFile"].value
s = getfilename(v)
# Free allocated memory:
DISPOSE v
# ...
# Reuse variable name:
DEFINE v AS integer
# ...
```

Operators and Expressions

An expression is a part of an Arbortext IsoDraw macro that can be used to produce a value. The simplest expressions are:

literals	A literal value evaluates to itself.
variables	Avariable evaluates to the value stored in that variable.

More complex expressions can be formed using simple expressions and operators.

Comparison Operators

Comparison operators compare operands. The result is always either true or false.

Operands to the comparison operators must be numeric.

The comparison operators are:

Equality (=)	If both operands are equal, this operator returns true; otherwise, it returns false.
Inequality (<>)	If both operands are not equal, this operator returns true; otherwise, it returns false.
Greater than (>)	If the lefthand operator is greater than the righthand operator, this operator returns true; otherwise, it returns false.
Greater than or equal to (>=)	If the lefthand operator is greater than or equal to the righthand operator, this operator returns true; otherwise, it returns false.
Less than (<)	If the lefthand operator is less than the righthand operator, this operator returns true; otherwise, it returns false.
Less than or equal to (<=)	If the lefthand operator is less than or equal to the righthand operator, this operator returns true; otherwise, it returns false.

Flow Control Statements

The Arbortext IsoDraw Macro Language supports basic flow control statements for controlling the execution of a macro.

Conditional statements allow a macro to execute different pieces of code, or none at all, depending on some condition. Loops support the repeated execution of particular code.

If

The `IF` statement checks the boolean result of an expression. If the expression is true it evaluates a statement.

Syntax

`IF` *(expression)* `THEN` *statements* [`ELSE` *statements*] `END IF`

expression	Boolean expression that evaluates to TRUE or FALSE
statements	One or more Arbortext IsoDraw Macro Language statements.

Example

```
IF (drawLine = true) THEN
   CREATE LINE 100 100 200 300
ELSE
   MESSAGE "No line created!"
END IF
```

You can include more than one statement in an if statement:

```
IF (selectLine = false) THEN
   CREATE LINE 100 200 300 400
   CREATE ELLIPSE 100 500 200 35 45
   MESSAGE "ready"
END IF
```

The `IF` statement can also be nested:

```
IF (expression) THEN
   true_statement_1
   IF (expression) THEN
      true_statement
   ELSE
      false_statement
   END IF
   true_statement_2
ELSE
   IF (expression) THEN
      true_statement
   END IF
END IF
```

While

In a `WHILE` loop, if the expression evaluates as true, the statement is executed and then the expression is reevaluated. If it is true again the body of the loop is executed and so on. The loop exits when the expression is evaluated as false.

Syntax

`WHILE` *(expression) statements* `END WHILE`

expression	Boolean expression that evaluates to TRUE or FALSE
statements	One or more Arbortext IsoDraw Macro Language statements.

Example

Following is a simple example that prints 10 parallel vertical lines on the screen:

```
DEFINE i AS integer
i = 1
WHILE (i <= 10)
   CREATE LINE 5*i 0 5*i 50
   i = i + 1
END WHILE
```

Like the `IF` statement you can also nest the `WHLE` statement:

```
WHILE (expression)
   #do something several times
   WHILE (expression)
      #do something different
   END WHILE
END WHILE
```

For

The `FOR` statement is similar to the `WHILE` statement, except that it adds counter initialization and counter manipulation expressions.

Syntax

`FOR` *counter=from-expression* `TO` *to-expression* `STEP` *step-counter statements* `END FOR`

counter	Counter variable.
from-expression	Start of valid range of counter values.
to-expression	End of valid range of counter values.
step-counter	Amount to increment counter for each loop.
statements	One or more Arbortext IsoDraw Macro Language statements.

Example

```
DEFINE i AS integer
FOR i = 0 TO 90
   CREATE ELLIPSE 300 200 100 180 i
   CREATE ELLIPSE 300 200 100 90 i
```

```
END FOR

DEFINE i AS integer
DEFINE j AS integer
j=90
FOR i = 0 TO j STEP 5
   CREATE ELLIPSE 300 200 100 180 i
   CREATE ELLIPSE 300 200 100 90 i
END FOR
```

Although the FOR statement is similar to the WHILE statement it is often shorter and easier to read than the equivalent WHILE loop. For example a WHILE loop that counts from 1 to 10 and prints the numbers in a message window:

```
DEFINE number AS integer
number = 1
WHILE (number <= 10)
   MESSAGE "The number is now " + number
   number = number + 1
END WHILE
```

Here's the corresponding for loop:

```
DEFINE number AS integer
FOR number = 1 TO 10
   MESSAGE "The number is now " + number
END FOR
```

Break

The BREAK statement breaks the running FOR or WHILE loop and continues the macro after the next END FOR or END WHILE.

Syntax

```
BREAK
```

Example

```
MACRO myOutbreak
   DEFINE x AS integer
   DEFINE y AS integer
   DEFINE ea AS integer
   x = 1
   WHILE (x <= 12)
      y = 1
      WHILE (y <= 12)
          ea = 11 * (x - 1) + 5 * y + 20
         IF (ea > 90)
            BREAK
         END IF
         CREATE ELLIPSE 35*x 30*y 10 ea 30
         y = y + 1
       END WHILE
       x = x + 1
   END WHILE
END MACRO
```

Run

Using the `RUN` statement inside a macro or subMacro executes another loaded macro or subMacro.

Syntax

`RUN` *name*

name	Name of macro or subMacro.

Example

```
SUBMACRO routine
   MESSAGE "in routine"
END SUBMACRO

###########################################

MACRO main
   MESSAGE "in main"
   RUN routine
   MESSAGE "back in main"
END MACRO
```

Return

Using the `RETURN` statement inside a subMacro returns the given value as a result to the calling macro.

Syntax

`RETURN` *expression*

expression	Expression to return to calling macro.

Example

```
SUBMACRO FindPageFormat
   DEFINE s AS string
   DEFINE p AS Point
   p.x = activeDoc.window.pageX
   p.y = activeDoc.window.pageY
   IF ( p.x > p.y )
   IF (p.x > p.y )
      s - "landscape"
   ELSE
      s = "portrait"
   END IF
   RETURN s
END SUBMACRO

MACRO GetPageFormat
   DEFINE format AS string
   format = RUN FindPageFormat
   MESSAGE "The page is in the " + format + " format"
```

```
END MACRO
```

Error Handling

Applies to Arbortext IsoDraw 7.0 F000 and later.

As soon as an error occurs in a macro or subMacro, execution of that macro or subMacro stops by default—even if the error is treatable. If the error was detected in a subMacro, the subMacro stops and the error is reported to the calling macro—which stops that calling macro. The error reporting continues to "bubble up" until all calling macros are stopped.

You can use the `ON ERROR` commands below to change the default error handling behavior. For example, you can prevent a macro from stopping when an error occurs using the command.

Example

```
SUBMACRO DeepError_0
   MESSAGE "Starting DeepError_0"
   ERROR -1
   MESSAGE "Ending DeepError_0"
END SUBMACRO
SUBMACRO DeepError_1
   MESSAGE "Starting DeepError_1"
   RUN DeepError_0
   MESSAGE "Ending DeepError_1"
END SUBMACRO
SUBMACRO BubbleUpErrors
   MESSAGE "Starting BubbleUpErrors"
   RUN DeepError_1
   MESSAGE "Ending BubbleUpErrors"
END SUBMACRO
```

On Error Goto

Applies to Arbortext IsoDraw 7.0 F000 and later.

This command defines a local error handling routine that is called as soon as an error is detected. The error handling routine only applies to the subMacro or macro that contains the `ON ERROR GOTO` command; the routine does not extend to the calling macro.

An error handling routine is not active until the `ON ERROR GOTO` command line executes.

Syntax

`ON ERROR GOTO` *macrocall*

macrocall	A macro or subMacro name. It is possible to add a valid list of parameters in brackets. Any specified variables and expressions will be evaluated at runtime when the error handling routine starts.

Example

```
SUBMACRO IgnoreErrorHandler
# Use an empty SubMacro as ErrorHandler
# to ignore all errors.
END SUBMACRO

SUBMACRO StandardErrorHandler( string sLine )
# This SubMacro simulates the standard error behaviour.
 ERROR app.lastMacroError sLine
END SUBMACRO

SUBMACRO GermanErrorMessageBox( string sLine )
# Example for custumized error handling
 DEFINE s AS string
 s = "Fehlernummer: " + app.lastMacroError
 s = s + $newline + "Zeile: " + sLine
 MESSAGE s
END SUBMACRO

SUBMACRO DumpErrToFile( string sOut )
# Write Error Messages to own file
 DEFINE nErr AS integer
 nErr = app.lastMacroError
 FWRITE sOut nErr +" "+ errorString( nErr, "..." )
END SUBMACRO

SUBMACRO MyErrorHandler( string sLine )
 DEFINE nErr AS integer
 nErr = app.lastMacroError
 # Handle specific error situation:
 IF ( nErr= 400 ) THEN
  MESSAGE "Error number 400: caught it!"
  Return
 END IF
 # Force standard error message as default:
 ERROR nErr sLine
END SUBMACRO

#-------------------------------------------------------
SUBMACRO IML_CheckErrorCode
 DEFINE nErr AS integer
 DEFINE sMsg AS string
 ON ERROR GOTO MyErrorHandler( \
     app.currentMacro.activeLine )
 sMsg = "Please enter an error number: [400]"
 nErr = GET integer sMsg
 ERROR nErr
END SUBMACRO

SUBMACRO IML_DumpErrorMessages
# Dump some system and macro error messages to a file.
 DEFINE i AS integer
 DEFINE sOut AS string
 sOut = "C:\err_msgs.txt"
 FNEW sOut 8_bit
 ON ERROR GOTO DumpErrToFile( sOut )
 FOR i=-1 TO -20 STEP -1
  ERROR i
 END FOR
```

```
 FWRITE sOut "---------------"
 FOR i=400 TO 420
  ERROR i
 END FOR
END SUBMACRO

Macro IML_ErrorHandling_Example

# Trigger error without installed handler:
# (If you delete the comment sign from the following line the macro will
# not be executed beyond this line!)

# ERROR -1 "Just kidding!"

# Ignore the triggered errors
 ON ERROR GOTO IgnoreErrorHandler
 ERROR -1
 ERROR 400 "Can you see me?"
 MESSAGE "No, you can't!"

# Pass a parameter to the error handler
# (The parameter values will not be evaluated until the error occurs!)
 ON ERROR GOTO GermanErrorMessageBox( \
     app.currentMacro.activeLine )
 MESSAGE 100/0

# Installed message handler only affects the local file
 Run IML_CheckErrorCode

# Error handler "GermanErrorMessageBox" is still
# active here
 ERROR "User triggered error"

# A less trival example
 Run IML_DumpErrorMessages

# Break macro execution on errors again...
 ON ERROR GOTO StandardErrorHandler( \
     app.currentMacro.activeLine )
 ERROR -1 "Just kidding!"
END Macro
```

Error

Applies to Arbortext IsoDraw 7.0 F000 and later.

The `ERROR` command triggers an error.

Syntax

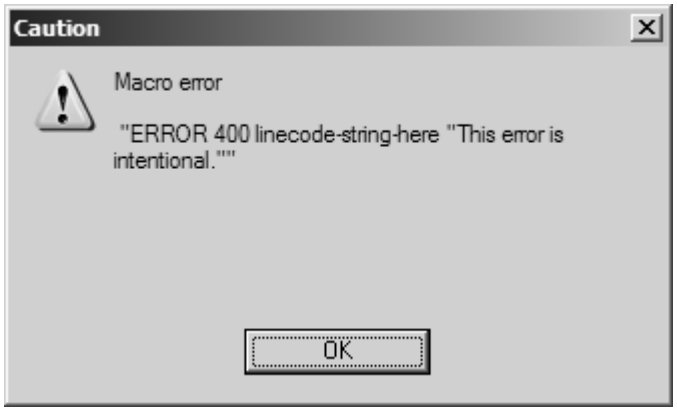
`ERROR` *errornumber* [*linecode*] [*"messagetext"*]

errornumber	(`integer`) Any positive or negative integer. • Negative value indicates system error. • Positive value in the range (400–599) indicates an application-specific error. • Zero value indicates no error—and setting *errornumber* to zero will not trigger an error.
linecode	(optional; `string`) Any valid character string. If the *errornumber* value is set to a postive integer in the range (400-599), the *errornumber* value will appear before the *linecode* string in the error message. ***Note*** *Do not enclose the linecode string in quotation marks.*
messagetext	(optional) Error message text

Example

This example includes the *linecode* string:

```
ERROR 400 linecode-string-here "This error is intentional."
```



Example

This example does not include the *linecode* string:

```
ERROR 400 "This error is intentional."
```

II

Menu Commands

3 File Menu

With this group of commands you can directly access most of the features from the Arbortext IsoDraw **File** menu.

New

The NEW command creates a new document. After being created this new document will be active.

Syntax

NEW [*"template"*]

template	Optional parameter used to load a template file. Without the parameter an empty file is created. You can define a path to a template-file or you can just give a file-name. In this case the template folder is searched for the template file.

The NEW command returns a reference on the created document. The properties of that document could be queried and set either through that reference or through the activeDoc object. (See Document Object on page 175.)

Example

```
#creates a new and empty document
NEW

#tries to load a template file located
#in the standard template folder
NEW "myTemplate.iso"

#load a template from a server
NEW "\\server\isoTemplates\ci.iso

#load a template from a mac HD
NEW "Macintosh HD:My Files:abc.iso

##############################################

DEFINE template AS string
template = "c:\templates\iso0815.iso"
NEW template
```

Open

The OPEN command opens a document as long as it is in a format that can be interpreted by Arbortext IsoDraw. After being opened that document is active.

Syntax

OPEN [*"path"*]

path	Optional parameter to define the path to the document that should be loaded. You should always define relative paths. If the parameter path is not given, the user can choose a file through the standard open-file dialog. If there is only a path without a file name in the path-parameter this path is the default path in the open -file dialog.

Example

```
#open, without a path, opens the open-file dialog
OPEN

#open, with a path, but without a file-name opens
#the open-file dialog with the folder specified in the path
OPEN "D:\IsoDraw-Data"

#open a file on a Windows-PC
OPEN "C:\Documentation\IsoDraw\engine.iso"

#open a file on a Mac
OPEN "Macintosh HD:IsoDraw Illus:frontWheel.iso"

########################################

DEFINE path AS string
path = "\\server\Documentation\Illustrations\
"open path + "test.iso"
OPEN path + "example.iso"
```

Close

The `CLOSE` command closes the active document or all open documents.

Syntax

`CLOSE ALL_WINDOWS [CONFIRM_YES | CONFIRM_NO | AS_CONFIRM]`

ALL_WINDOWS	If this keyword is provided the `CLOSE` command closes all open documents.
CONFIRM_YES	All unsaved documents will be saved before closing. If a document was not saved before, a file-save dialog will appear asking the user for a name and path. If the user cancels this dialog the macro itself will terminate.
CONFIRM_NO	All documents will be closed whether they were saved before or not.
ASK_CONFIRM	Default if no parameter is specified. Before closing any unsaved document the user will be asked if he would like to save that document. If the user cancels the dialog the macro itself will be terminated.

Example

```
CLOSE

CLOSE ALL_WINDOWS CONFIRM_NO

CLOSE CONFIRM_YES
```

Save

The SAVE command saves the active document in the Arbortext IsoDraw or CGM format.

Syntax

SAVE [*"path"*] [*"version"*] [PACKED]

path	Optional parameter defines where, and under what name, the document should be saved. If the parameter is not given the actual path of the active document is taken. If the document was not saved before the path will be requested through a dialog.
version	Optional parameter for saving the document as a version other than the default ISO or CGM format (if CGM is set as the native format). Possible version-strings:
PACKED	Optional parameter to create an Arbortext IsoDraw file containing all the data from placed files. Only the data required for displaying the placed file is saved. The link to the original file is broken. This option should only be given for version 5 or higher.

default	Actual Arbortext IsoDraw version or CGM-format if set as native in the preferences.
version_6	Arbortext IsoDraw 6 format
version_5	Arbortext IsoDraw 5 format
version_4	Arbortext IsoDraw 4 format
version_3	Arbortext IsoDraw 3 format
CGM	CGM format

Example

```
#save on a Mac
SAVE "Macintosh HD: My Files: xy.iso"

#save on a PC in CGM format
SAVE "c:\temp\illustration.cgm" "CGM"

#save on a server - "packed"
SAVE "\\server\documentation\illus\V10_engine.iso" PACKED
```

Import Layers

The `IMPORT LAYERS` command imports the defined layers from the defined document.

Syntax

`IMPORT LAYERS` *"path" "layer_name"*

path	Defines the source document's location.
layer_name	Defines the name of the layer being imported.

Example

```
#importing 3 layers
IMPORT LAYERS "c:\work\engine.iso" \
              "engine_mount" \
              "motor_starter" "motor_control_unit"
```

Save Layers

The `SAVE LAYERS` command saves all defined layers from the current document as an standard Arbortext IsoDraw file.

Syntax

`SAVE LAYERS` *"path" "layer-name"*

path	Defines where and under what name the document will be saved.
layer-name	Defines the layers to be saved. At least one layer name has to be defined.

Example

```
#save some layers
SAVE LAYERS "c:\temp\layers.iso" "Main" "Illustration"
```

Export

The `EXPORT` command exports the current document in the given format.

Syntax

`EXPORT` *"path" "exportformat"* [*percentage*]

path	Defines where, and under what name, the document should be exported.
exportformat	File format for saved file. The table below lists the available formats with their corresponding keys.
percentage	Optional parameter that defines scaling of the image in percent. Since it is defined as float you can enter a number with decimal digits.

exportformat Keys	exporformat Keys Description
AI	Adobe Illustrator
BMP	Bitmap
CALS_Raster	Continuous Acquisition and Lifecycle Support
CGM	Computer Graphics Metafile
DWG	File format of AutoCAD
DXF	Drawing Interchange Format
EPSF	Encapsulated PostScript File
HPGL	Hewlett Packard Graphics Language
IGES	Initial Graphics Exchange Standard
Interleaf	Publishing system
JPEG	Raster format
MIF	Maker Interchange Format
Objects_Text	All object informations as text file
PICX	Raster format
PICT	PICTure format from Apple Macintosh
PNG	Portable Network Graphic
SVG	Scalable Vector Graphics
SVGZ	Compressed SVG
Text-Excerpt	Arbortext IsoDraw's own formats
TIFF	Tagged Image File Format
WMF	Windows Meta File

Example

```
#export as CGM in the original size
EXPORT "a:\plan.cgm" "CGM"

#export as TIFF scaled down 66%
EXPORT "c:\picts\results.tif" "TIFF" 66

#export as JPEG scaled down 33.33%
EXPORT "Macintosh HD:My Files:results.jpg" 33.33
```

Place

The `PLACE` command places a document on the current ilustration.

Syntax

`PLACE` *"path"* [*x1 y1 x2 y2*]

path	Defines the placed file's path. You should always define relative paths. If the parameter path is not given, the user can choose a file through the standard open-file dialog.
x1 y1 x2 y2	(optional) Defines the size and location of the target rectangle for the placed file using the coordinates of the rectangle's lower left corner and upper right corner. (*x1 y1*) specifies the lower left corner; (*x2 y2*) specifies the upper right corner. ***Note*** *For 3D files, the target rectangle is not relevant and will be ignored.*

Example

```
#no parameters
PLACE

#just the filename
PLACE "c:\screw.iso"

#the filename and the coordinates of the placement
PLACE "C:\screw.iso" 120 150 180 210
```

Printer Setup

Applies to Arbortext IsoDraw 7.0 F000 and later.

The PRINTER SETUP command changes print settings in Arbortext IsoDraw.

If you include parameters in the command line, PRINTER SETUP changes the print settings corresponding to those parameters. If you do not include parameters in the command line, PRINTER SETUP opens the **Print Setup** dialog box so the user can change print settings manually.

If you run PRINTER SETUP with one or more documents open, print setting changes apply to the current document only. If you run PRINTER SETUP with no documents open, any print setting changes become the default settings in the **Print Setup** dialog box.

Syntax

`PRINTER SETUP` [`OVERLAP` *x y*] [*scaling*] [`ORIENTATION` {`PORTRAIT` | `LANDSCAPE`}] [`PAPER_SIZE` *width height*]

No parameters	Displays the **Print Setup** dialog box.
OVERLAP *x y*	(optional) Specifies the horizontal (*x*) and vertical (*y*) overlap in millmeters. (Overlap enables printed pages to be stitched together.) If `OVERLAP` values are omitted, the printer's default settings are used.

<table>
<tr><td>scaling</td><td>(optional) Specifies a scale factor in percent. Any value less than <code>100</code> will scale the print down from its original size. Any value higher than <code>100</code> will enlarge the printout.</td></tr>
<tr><td>ORIENTATION</td><td>(optional) Sets the printed output orientation to <code>PORTRAIT</code> or <code>LANDSCAPE</code>.

Note
Currently, the <code>ORIENTATION</code> parameter is ignored when a macro executes PRINTER SETUP. However, the macro recorder will record the <code>ORIENTATION</code> parameter setting.

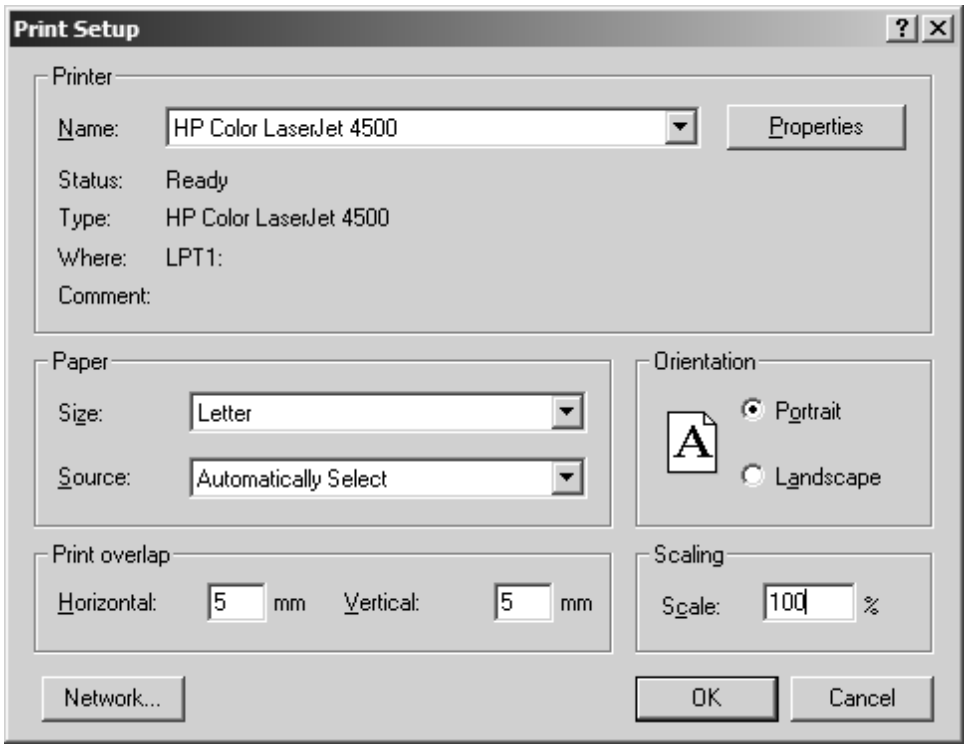
<table><tr><td>PORTRAIT</td><td>Specifies portrait orientation.</td></tr><tr><td>LANDSCAPE</td><td>Specifies landscape orientation.</td></tr></table></td></tr>
<tr><td>PAPER_SIZE</td><td>(optional) Specifies the size of the paper in millimeters. If the width value is greater than the height value, the orientation is landscape by default. The <code>PAPER_SIZE</code> specified must match one of the available paper sizes. If it does not, the command will not change the paper size.

Note
Currently, the <code>PAPER_SIZE</code> parameter is ignored when a macro executes PRINTER SETUP. However, the macro recorder still records <code>PAPER_SIZE</code> settings.

<table><tr><td>width</td><td>The paper width.</td></tr><tr><td>height</td><td>The paper height.</td></tr></table></td></tr>
</table>

Example

```
PRINTER SETUP
```




```
PRINTER SETUP OVERLAP 0 0
PRINTER SETUP SCALING 75
PRINTER SETUP ORIENTATION LANDSCAPE
PRINTER SETUP ORIENTATION PORTRAIT SCALING 110
PRINTER SETUP SCALING 250 OVERLAP 10 20

# Set 'Letter' Format:
PRINTER SETUP PAPER_SIZE 215.90 279.399
```

Print

Applies to Arbortext IsoDraw 7.0 F000 and later.

Prints the current document. If there are no parameters defined, the **Print** dialog will open.

Note

Currently, `PRINT` parameter settings are ignored when a macro executes. However, the macro recorder still records `PRINT` parameter settings.

Syntax

`PRINT` [*pageselect*] [`COPIES` *c* `[SORTED]`] [`OVERLAP` *x y*] [`TO_FILE` [*"path"*]]

<table>
<tr><td>No parameters</td><td>Displays the Print dialog box.</td></tr>
<tr><td>pageselect</td><td>(optional) Prints the document pages specified by one of the pageselect keyword phrases below:

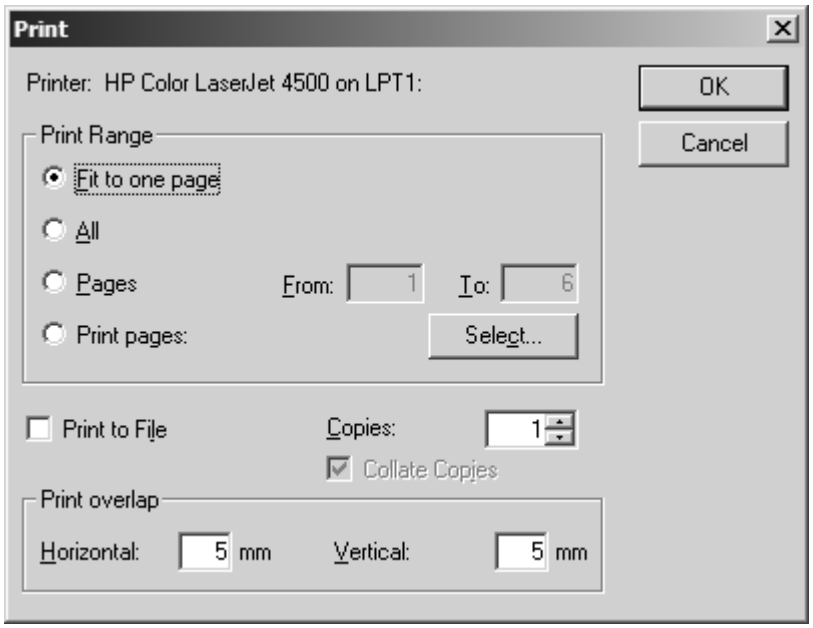
<table>
<tr><td>ALL_PAGES</td><td>Prints all pages in the document.</td></tr>
<tr><td>FIT_TO_ON-E_PAGE</td><td>Reduces the illustration so it fits on a single page when printed.</td></tr>
<tr><td>PAGE n</td><td>Prints one page; page number n.</td></tr>
<tr><td>PAGES FROM n1 TO n2</td><td>Prints a range of pages or multiple pages in the document.</td></tr>
<tr><td>PAGES ni</td><td>Prints multiple pages. The page numbers, ni, do not have to be sequential.</td></tr>
</table></td></tr>
<tr><td>COPIES c</td><td>(optional) Prints the specified number of document copies, c.

<table>
<tr><td>SORTED</td><td>(optional) Prints document copies in page number order.</td></tr>
</table></td></tr>
<tr><td>OVERLAP x y</td><td>(optional) Specifies the horizontal (x) and vertical (y) overlap in millmeters. (Overlap enables printed pages to be stitched together.) If <code>OVERLAP</code> values are omitted, the printer’s default settings are used.</td></tr>
<tr><td>TO_FILE</td><td>(optional) Prints the document to a file instead of a printer.

<table>
<tr><td>path</td><td>(optional) The location and filename of the file to be printed to. If no path is specified, a Save As… dialog box appears when <code>PRINT TO_FILE</code> executes.</td></tr>
</table></td></tr>
</table>

Example

```
PRINT
```



```
PRINT ALL_PAGES
PRINT PAGE 1
PRINT PAGES FROM 3 TO 4
PRINT PAGES 2 4 6
PRINT ALL_PAGES COPIES 3
PRINT PAGES FROM 1 TO 3 COPIES 10 SORTED
PRINT ALL_PAGES OVERLAP 5 5
PRINT TO_FILE "C:tmp\output"
```

Save

Saves the current document in Arbortext IsoDraw file format or CGM format (if CGM is set as the native format).

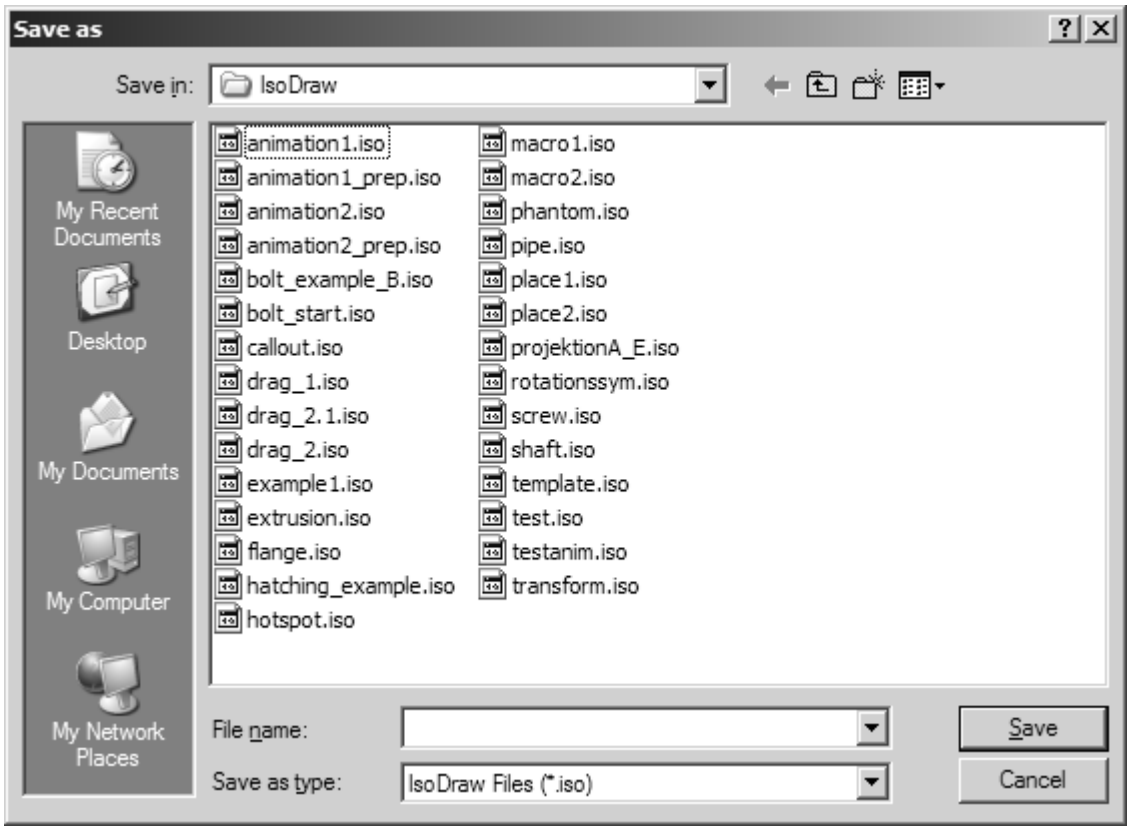
Syntax

`SAVE` [*"path"*] [*"version"* `[PACKED]`]

No parameters	Displays the **Save As** dialog box.
path	(optional) The location and filename of the file to be saved. If no *path* is specified, a **Save As...** dialog box appears when `SAVE` executes.
version	(optional; `string`) Any one of the file format specifiers below: **Default** — Current Arbortext IsoDraw version (or CGM if CGM is set as the native format) **Version_71** — Arbortext IsoDraw 7.1 format **Version_7** — Arbortext IsoDraw 7 format **Version_6** — Arbortext IsoDraw 6 format **Version_5** — Arbortext IsoDraw 5 format **Version_4** — Arbortext IsoDraw 4 format **Version_3** — Arbortext IsoDraw 3 format **CGM** — CGM format
PACKED	(optional) Packs all information into one file. External files placed in the document are included in the packed file. ***Note*** *This keyword has been supported since Arbortext IsoDraw 5.0. However, if you are running Arbortext IsoDraw 7.0 or later you can only save Arbortext IsoDraw 7.0 or later files in packed format.*

Example

```
SAVE "C:\sarah\cluch.cgm" "CGM"
SAVE "C:\sarah\cluch4.iso" "Version_4"
SAVE "C:\sarah\cluch5p.iso" "Version_5" packed
SAVE
```



Quit

The `QUIT` command quits the application.

Syntax

`QUIT [CONFIRM_YES | CONFIRM_NO | ASK_CONFIRM]`

CONFIRM_YES	All unsaved documents will be saved before closing. If a document was not saved before, a file-save dialog will appear asking the user for a name and path. If the user cancels this dialog the macro itself will terminate.
CONFIRM_NO	All documents will be closed whether they were saved before or not.
ASK_CONFIRM	Default if no parameter specified. Before closing any unsaved document the user will be asked if he would like to have that document saved. If the user cancels this dialog the macro itself will be terminated.

Example

```
#quit
QUIT CONFIRM_YES
```

4 Edit Menu

With this group of commands you can directly access most of the features from the Arbortext IsoDraw **Edit** menu. The commands `UNDO` and `REDO` will not be recorded through the Arbortext IsoDraw macro recorder.

See Selecting Elements on page 96 for additional commands to select elements.

Cut

With the CUT command you can remove selected elements. The elements are deleted and saved in the program's clipboard, replacing the former contents of the clipboard. The cut elements can be retrieved from the clipboard at any time through the command paste.

Syntax

```
CUT
```

Copy

With the COPY command you can copy selected elements into the program's clipboard, replacing the former contents of the clipboard. You can retrieve the copied elements from the clipboard at any time through the command paste.

Syntax

```
COPY
```

Duplicate

With the DUPLICATE command you can paste a copy of the selected elements into the document. To a certain degree the command is a combination of the commands COPY and PASTE SAME_POSITION. However the selected elements are not saved into the clipboard. This has the advantage that the old contents of the clipboard are retained.

Syntax

```
DUPLICATE
```

Syntax

```
DUPLICATE
```

Delete Selection

The DELETE SELECTION command deletes the current selection.

Syntax

```
DELETE SELECTION
```

Paste

The PASTE command pastes the contents of the clipboard into the middle of the screen window. Even after you have activated another Arbortext IsoDraw file, the elements are still pasted into the middle of the screen.

Syntax

```
PASTE[SAME_POSITION][MAINTAIN_ATTRIBUTES]
```

SAME_POSITION	Optional parameter defines that the elements are inserted at the same position from where they were cut or copied. This applies even if you have changed the screen position. Even if you have activated another Arbortext IsoDraw file, the elements are still pasted at the precise position where they where cut or copied from the original document.
MAINTAIN_ATTRIBUTES	Optional parameter defines that the standard attributes of the elements (pens, styles, shadows, colors) in the original document will be retained when pasting.

Example

```
#paste the clipboard content and retain
#the standard attributes of the elements
PASTE MAINTAIN_ATTRIBUTES
```

Select All

The SELECT ALL command activates all the elements and groups in the current document. Elements located on locked layers are not activated.

Syntax

```
SELECT ALL
```

Select None

The SELECT NONE command deactivates the selection in the current document.

Syntax

```
SELECT NONE
```

Select Object

Applies to Arbortext IsoDraw 7.0 F000 and later:

Selects objects by object ID.

Note

This command will NOT trigger any error or warning in the specified object(s) do not exist.

Syntax

`SELECT OBJECT` *object_id1* [*object_id2*... [*object_idn*]]`[TOGGLE]`

object_id1	(`string`) Specifies the object ID of the object to select. (You must specify at least one object ID.)
object_id2 … object_idn	(optional; `string`) Specifies additional object IDs of objects to select. Separate *object_ID*s with a space character.
TOGGLE	Switches the specified object(s) current selection state from selected to not selected, or vice-versa.

Example

```
SELECT OBJECT "AUTOID_44400"
SELECT OBJECT "SCREW_L" "SCREW_R" "SCREW_C"
SELECT OBJECT sObj toggle
```

Select If / SubSelect If

The `SELECT IF` command lets you select elements according to specific criteria. This is particularly useful if you want to edit several elements with the same attributes.

The `SUBSELECT IF` command, like the `SELECT IF` command, lets you select elements according to specific criteria. The difference between the two commands is that `SUBSELECT IF` selects the specified elements from an already existing selection.

Syntax

{`SELECT IF | SUBSELECT IF`} *operand_1 comparison_operator*

comparison_operator	EXISTS or EXISTS NOT.
operand_1	The parameter can be assigned with these values: OBJECT_INFO OBJECT_NAME OBJECT_TIP OBJECT_ATTRIBUTE

{SELECT IF | SUBSELECT IF} *operand_1 comparison_operator "operand_2"*

comparison_operator	IS or IS NOT or IS EQUAL TO or IS NOT EQUAL TO
operand_1, operand_2	operand_2 depends on the value of operand_1 according to the following table:

operand_1	operand_2
TYPE	"line" "rectangle" "polygon" "marker" "ellipse" "inner_thread" "outer_thread" "bezier" "text" "placed_file" "image" "callout"
PEN	all standard pen names "$ISO_NOPEN"
LINESTYLE	all standard linestyle names
SHADOW	all standard shadow names "$ISO_NOSHADOW"
COLOR	all standard color names "$ISO_NOFILL" "$ISO_WHITE" "$ISO_BLACK"
HATCHING	any "string"
PATTERN	any "string"

{SELECT IF | SUBSELECT IF} *operand_1 comparison_operator "operand_2"*

comparison_operator	IS or IS EQUAL TO or IS NOT or IS NOT EQUAL TO
operand_1, operand_2	operand_2 depends on the value of operand_1 according to the following table:

operand_1	operand_2
OBJECT_ID	any "string"
OBJECT_NAME	any "string"
OBJECT_TIP	any "string"

	OBJECT_ATTRIBUTE	any "string"
	TEXT	any "string"
	TEXT_POSITION	any float
	TEXT_ALIGNMENT	"CENTERED" "LEFT" "RIGHT" "MEDIUM" "MIDDLE" "TOP" "BOTTOM" "BASE_LINE"
	FONT	any "string" e.g.: "arial"
	FONT_SIZE	any float
	FONT_FACE	"NORMAL" "BOLD" "ITALIC" "BOLDITALIC"
	LEADING	any float OR "$AUTOMATIC"
	KERNING	any float
	CALLOUT	any "string"

`{SELECT IF | SUBSELECT IF}` *operand_1 comparison_operator "operand_2"*

comparison_operator	IS or IS NOT or IS EQUAL TO or IS NOT EQUAL TO or CONTAINS
operand_1	Any string
operand_2	Any string

Example

```
SELECT IF object_info EXISTS_NOT

#both lines are identical
SUBSELECT IF type IS EQUAL TO "line"
SUBSELECT IF type IS "line"

SELECT IF OBJECT_ATTRIBUTE "positioning" CONTAINS "left"
```

Move

The `MOVE` command moves the current selection. Without any parameter given the **Move** dialog is opened.

Syntax

```
MOVE
MOVE SELECTION x y NO_ELEMENTS NO_PATTERNS
MOVE COPY x y CONNECT NO_ELEMENTS NO_PATTERNS
```

SELECTION	Move the active selection.
COPY	Copy the selection.
x, y	Define the direction of the movement.
CONNECT	Copy with connecting lines.
NO_ELEMENTS	Move patterns with reference to elements.
NO_PATTERNS	Move only the elements

Example

```
#just moving the selection
MOVE SELECTION 123.22 (-11.98) NO_PATTERNS

#moving and copying
MOVE COPY 77.99 22 CONNECT
```

Align

The ALIGN command aligns selected 2D elements in a specified direction.

Syntax

ALIGN *direction*

direction	May be one of: LEFT, MEDIUM, RIGHT, TOP, MIDDLE or BOTTOM

Distribute

The DISTRIBUTE command equally distributes selected 2D elements in a specified direction.

Syntax

DISTRIBUTE *direction*

direction	May be one of: LEFT, MEDIUM, RIGHT, TOP, MIDDLE or BOTTOM

Example

```
SELECT IF Type IS EQUAL TO "text"
DISTRIBUTE LEFT
```

Preferences

The preferences can be accessed through the application object. (See Application Object - User Interface Preferences on page 223 and Application Object - Data Exchange Preferences on page 249.)

5 Element Menu

With this group of you can directly access most of the features from the Arbortext IsoDraw **Element** menu. See Edit Menu on page 59 and Selecting Elements on page 96 for commands to select elements.

Arrange

With the `ARRANGE` command you can reorder elements, bringing them above or below others.

Syntax

`ARRANGE` *direction*

direction	Defines direction that selected elements will be re-arranged. Allowed values are:

top	Element will be above all others.
below	Element will be below all others.
up	Order of arrangement will be adjusted by one position towards the top.
down	Order of arrangement will be adjusted by one position towards the bottom.

Example

```
ARRANGE top
```

Convert Selection into Elements

With the `CONVERT SELECTON INTO ELEMENTS` command you can convert selected elements into simpler types of elements based on the table below:

Elements	Converts Into
Polyline	Lines
Rectangle	Lines
Inner Thread	Ellipses (segments)
External Thread	Lines and Ellipses (segments)
Polygon	Lines
Dimension	Already a group of basic elements
Callout	Polylines, Text and Rectangle or Polyline and Ellipses

Syntax

```
CONVERT SELECTON INTO ELEMENTS
```

Convert Selection into Bezier Parts

With the `CONVERT SELECTION INTO BEZIER PARTS` command you can convert selected elements into Beziers.

Syntax

```
CONVERT SELECTION INTO BEZIER PARTS
```

Convert Selection into Polylines

With the convert selection into Polylines command you can convert selected elements into Polylines.

Syntax

```
CONVERT SELECTON INTO POLYLINES[NO_HANDLES]
```

NO_HANDLES	Optional parameter defines that elements with handles, such as Beziers will not be converted.

Join Polylines

With the `JOIN POLYLINES` command you can join selected lines and polylines into a single polyline.

Syntax

```
JOIN POLYLINES
```

Join Beziers

With the `JOIN BEZIERS` command you can join selected elements into a single Bezier.

Syntax

```
JOIN BEZIERS[CONVERT]
```

CONVERT	Optional parameter defines that all elements will be converted into Beziers and joined. Without this option only Beziers will be joined.

Generate Contour

With the `GENERATE CONTOUR` command you can create a continous, closed contour from the selected elements.

Syntax

```
GENERATE CONTOUR
```

Create Compound Path

With the `CREATE COMPOUND` path command you can create a connected, compound path from the selected elements.

Syntax

```
CREATE COMPOUND
```

Groups

Group Selection

With the `GROUP SELECTION` command you can create a group from all selected elements.

Syntax

```
GROUP SELECTION
```

Example

```
GROUP SELECTION

MACRO all_lines_in_one_group
   SELECT IF type IS "line"
   GROUP SELECTION
END MACRO
```

Ungroup Selection

The `UNGROUP SELECTION` command lets you ungroup the group you have selected.

Syntax

```
UNGROUP SELECTION[DEEP]
```

DEEP	Optional parameter defines that the command should ungroup all nested groups in the selection. This is similar to using the shift key when ungrouping manually.

Start Group

The `START GROUP` command starts a group. After executing this command every new element created through a macro command will be added to that group unless the group is closed through an end group command or a new group is opened with start group.

Syntax

```
START GROUP
```

End Group

The `END GROUP` command closes the last grouping level that was started with start group.

Syntax

```
END GROUP
```

Example

```
MACRO nested_groups
   START GROUP
      START GROUP
         CREATE ELLIPSE 254.557 195.959 30 240 35.264
         CREATE ELLIPSE 197.99 195.959 30 300 35.264
         CREATE ELLIPSE 226.274 244.949 30 180 35.264
      END GROUP
      CREATE RECTANGLE  226.274 146.968 282.843 179.628 282.843\
                        244.949 226.274 212.288
      CREATE RECTANGLE 282.843 244.949 226.274 277.608 169.705\
                        244.949 226.274 212.288
      CREATE RECTANGLE 169.705 244.949 169.705 179.628 226.274\
                        146.968 226.274 212.288
   END GROUP
END MACRO
```

Mask At

With the mask at command you can mask the selected element with the element at the location specified.

Syntax

`MASK AT` *x y*

x, y	Location of the element which will be used to mask the selected element.

Example

```
MASK AT 120 340
```

Release Mask

With the RELEASE MASK command you can remove the selected mask.

Syntax

```
RELEASE MASK
```

Lock

With the LOCK command you can lock selected elements.

Syntax

```
LOCK
```

Unlock

With the UNLOCK command you can unlock all elements.

Syntax

```
UNLOCK
```

Transformation 3D

With the TRANSFORMATION 3D command you can switch to the 3D transformation mode. A placed 3D element needs to be selected.

Syntax

```
TRANSFORMATION 3D
```

Set Image Transparency

With the SET IMAGE TRANSPARENCY command you can toggle the transparency setting for raster images.

Syntax

SET IMAGE TRANSPARENCY *switch*

switch	Allows you to change the setting. Allowed values are ON and OFF.

Example

```
SET IMAGE TRANSPARENCY ON
```

Import Selection

With the `IMPORT SELECTON` command you can import a selected, placed element. This will break the link to the placed file.

Syntax

```
IMPORT SELECTON
```

6 Objects Menu

With this group of commands you can directly access most of the features from the Arbortext IsoDraw **Objects** menu.

Create Object_info

The CREATE OBJECT_INFO command creates an object info for an element.

Syntax

`CREATE OBJECT_INFO` *element*

element	Element that should get an object info. For further information seeDocument Object on page 175 and Element Object on page 193.

Example

```
DEFINE el AS element
el = activeDoc.firstSelectedElement
CREATE OBJECT_INFO el
```

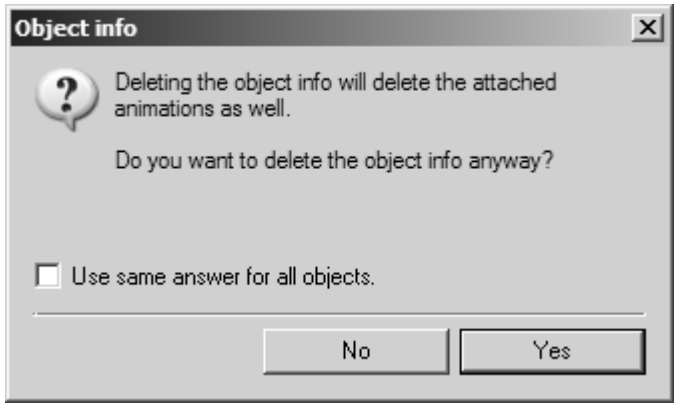
Delete Object_info

The DELETE OBJECT_INFO command removes object info from all selected elements and groups.

Syntax

`DELETE OBJECT_INFO[DEEP][NO_ANIMATED ]`

DEEP	(optional) Additionally removes object info from all elements and nested groups in the selected groups. Using `DELETE OBJECT_INFO DEEP` corresponds to choosing **Objects ▸ Delete object info** while holding the SHIFT key down.
NO_ANIMATED	(optional) Prevents animations attached to objects and/or groups from being deleted when you delete their object info. This command corresponds to choosing **Objects ▸ Delete object info** with animated objects and/or groups selected—then clicking **No** when the dialog box below asks you to verify that you want to delete animations along with the object info:

Create Object_attribute

CREATE OBJECT_ATTRIBUTE creates a new attribute for the object info of a given element.

Syntax

`CREATE OBJECT_ATTRIBUTE` *"name" "type" element*

name	Name of the new attribute as string.
type	Type of the new attribute. Possible data types are `INTEGER`, `FLOAT`, `STRING` and `LINK` where the `LINK` is just a special string.
element	Element which should get a new attribute to its object info. The element must already have an object info (see Create Object_info on page 76).

Example

```
MACRO Create Object Info

DEFINE el AS element
el = CREATE ELLIPS 100 100 100 90 90
CREATE OBJECT_IONFO  el

CREATE OBJECT_ATTRIBUTE "Title" "string" el
el.info.attributes["Title"].value = "IsoDraw"

CREATE OBJECT_ATTRIBUTE "Stock" "integer" el
el.info.attributes["Stock"].value = 10

CREATE OBJECT_ATTRIBUTE "URL" "link" el
el.info.attributes["URL"].value = "http://www.itedo.com"

END MACRO
```

Delete Object_attribute

The DELETE OBJECT_ATTRIBUTE command removes an object attribute.

Syntax

DELETE OBJECT_ATTRIBUTE *"name" element*

name	Defines the name of the attribute that has to be removed.
element	Defines the element.

Example

```
DELETE OBJECT_ATTRIBUTE "Stock" el
```

Create Hotspots

The CREATE HOTSPOTS command generates hotspots for selected text elements.

Syntax

CREATE HOTSPOTS *element n* NUMBERS_ONLY CONVERT_SPACES

n	defines the number of characters from the text element that will be assigned as the Object Name.
numbers_only	Optional parameter defines that only elements comprised entirely of numbers will have hotspots generated.
CONVERT_SPACES	Optional parameter defines that any spaces in the object name will be converted into an underscore (_) when used as the Object Name.

This function returns an integer value. This value represents the number of hotspots created.

Example

```
CREATE HOTSPOTS 64 CONVERT_SPACES
```

Show in IsoView

Applies to Arbortext IsoDraw 7.0 F000 and later.

Shows the current document in Arbortext IsoView. If there is no browser window open yet, the command opens a Microsoft Internet Explorer browser window first. Optional save confirmation parameters determine how to deal with documents which have not been saved yet.

Syntax

```
SHOW IN ISOVIEW[CONFIRM_YES | CONFIRM_NO | ASK_CONFIRM]
```

No parameter	Arbortext IsoDraw will try to save the document if it has not been saved yet (same behavior as `ASK_CONFIRM` below).
CONFIRM_YES	Any document which has not been saved yet will be saved before being viewed in Arbortext IsoView. If there is no name for the document available yet, the **Save As** dialog box appears. If this dialog box is cancelled, the macro will abort.
CONFIRM_NO	If the document has not been saved yet, this command is aborted. In any other case the command will continue and display the document in Arbortext IsoView.
ASK_CONFIRM	Arbortext IsoDraw will try to save the document if it has not been saved yet.

Example

```
# This command prevents the document from being shown
# in Arbortext IsoView if the document has not been saved yet:
SHOW IN ISOVIEW CONFIRM_NO
```

Delete Uncompliant Attributes

Applies to Arbortext IsoDraw 7.0 F000 and later.

Deletes attributes from selected objects that do not comply with the currently selected DTD.

Syntax

```
DELETE UNCOMPLIANT ATTRIBUTES
```

Returns the number of objects affected/changed.

Example

```
MACRO Delete_All_Non_DTD_Attributes
 DEFINE count AS integer
 SELECT ALL
 count = DELETE UNCOMPLIANT ATTRIBUTES
 MESSAGE "Changed " + count + " object(s)"
  END MACRO
```

7 Text Menu

With this group of commands you can directly access most of the features from the Arbortext IsoDraw **Text** menu.

Text

The `TEXT` command activates one text property and assigns it to all selected text elements. All text elements created after the execution of this command will also have these properties.

Syntax

`TEXT` *attribute value*

attribute	**value**
FONT	Any installed font name as a quoted string.
SIZE	Any float number as a point value.
FACE	One of: NORMAL, BOLD, ITALIC, BOLDITALIC.
ALIGN	One of: As a global alignment: CENTERED As a horizontal alignment: LEFT, MEDIUM, RIGHT As a vertical alignment: TOP, MIDDLE, BOTTOM, BASE_LINE
LEADING	Any float number as a point value.
KERNING	Any float number.
POSITION	Any float number as a point value.

Example

```
TEXT FONT "arial"
TEXT SIZE 12.8
TEXT FACE BOLD
TEXT ALIGN RIGHT
TEXT LEADING 3.7
TEXT KERNING 7
TEXT POSITION 12.7
```

Convert Text to Paths

The `CONVERT TEXT TO PATHS` command will convert selected text elements in to filled Bézier paths.

Syntax

```
CONVERT TEXT TO PATHS
```

8 Window Menu

With this group of commands you can directly access most of the features from the Arbortext IsoDraw **Windows** menu.

Activate Window

The ACTIVATE WINDOW command activates an already opened document.

Syntax

ACTIVATE WINDOW *"file"*

file	Name of a document window that should be activated. The document has to be open.

Example

```
#activate a already opened document
ACTIVATE WINDOW "abc.iso"
```

Hide

The HIDE command closes an Arbortext IsoDraw window. (If the window is already hidden, this command has no effect.)

Note

The Arbortext IsoDraw macro recorder does not record the HIDE and SHOW commands.

Syntax

HIDE {PALETTE | *window*}

PALETTE	Hides the palette window (toolbox).
window	Hides the specified window: **ATTRIBUTE_WINDOW** — Hides the attribute window. Adding one the following optional parameters after ATTRIBUTE_WINDOW shows the main attribute window with the specified attribute window hidden: **PENS_WINDOW** — Hides the **Pens** attribute window. **STYLES_WINDOW** — Hides the **Styles** attribute window. **SHADOWS_WINDOW** — Hides the **Shadows** attribute window.

		GRID-S_WIN-DOW	Hides the **Grids** attribute window.
		FORMA-TS_WIN-DOW	Hides the **Formats** attribute window.
		VIEW-PORT-S_WIN-DOW	Hides the **Viewports** attribute window.
		CALL-OUT_S-TYLE-S_WIN-DOW	Hides the **Callouts** attribute window.
	LAYER_WI-NDOW	Hides the **Layers** window.	
	FILL_WIN-DOW	Hides the **Fills** window.	
	OB-JECT_WIN-DOW	Hides the **Objects** window.	
	LIBRARY_ WINDOW	Hides the **Library** window.	
	BROWSER_ WINDOW	Hides the **Browser Window**.	
	ALL	Hides all windows listed on the **Windows** menu.	

Example

```
HIDE PALETTE
HIDE ATTRIBUTE_WINDOW
HIDE LAYER_WINDOW
```

Show

The `SHOW` command opens an Arbortext IsoDraw window. (If the window is already showing, this command has no effect.)

Note

The Arbortext IsoDraw macro recorder does not record the `SHOW` and `HIDE` commands.

Syntax

`SHOW` {`PALETTE` | *window*}

<table>
<tr><td>PALETTE</td><td>Shows the palette window (toolbox).</td></tr>
<tr><td>window</td><td>Shows the specified window.
<table>
<tr><td>ATTRIBUTE_WINDOW</td><td>Shows the attribute window. Adding one the following optional parameters after <code>ATTRIBUTE_WINDOW</code> shows the specified attribute window in the main attribute window.
<table>
<tr><td>PENS_WINDOW</td><td>Shows the Pens attribute window.</td></tr>
<tr><td>STYLES_WINDOW</td><td>Shows the Styles attribute window.</td></tr>
<tr><td>SHADOWS_WINDOW</td><td>Shows the Shadows attribute window.</td></tr>
<tr><td>GRIDS_WINDOW</td><td>Shows the Grids attribute window.</td></tr>
<tr><td>FORMATS_WINDOW</td><td>Shows the Formats attribute window.</td></tr>
<tr><td>VIEWPORTS_WINDOW</td><td>Shows the Viewports attribute window.</td></tr>
<tr><td>CALLOUT_STYLES_WINDOW</td><td>Shows the Callouts attribute window.</td></tr>
</table></td></tr>
<tr><td>LAYER_WINDOW</td><td>Shows the Layers window.</td></tr>
<tr><td>FILL_WINDOW</td><td>Shows the Fills window.</td></tr>
<tr><td>OBJECT_WINDOW</td><td>Shows the Objects window.</td></tr>
<tr><td>LIBRARY_WINDOW</td><td>Shows the Library window.</td></tr>
<tr><td>BROWSER_WINDOW</td><td>Shows the Browser Window.</td></tr>
<tr><td>ALL</td><td>Shows all windows listed on the Windows menu.</td></tr>
</table></td></tr>
</table>

Example

```
SHOW PALETTE
SHOW ATTRIBUTE_WINDOW
SHOW ATTRIBUTE_WINDOW VIEWPORTS_WINDOW
SHOW LAYER_WINDOW
SHOW ALL
```

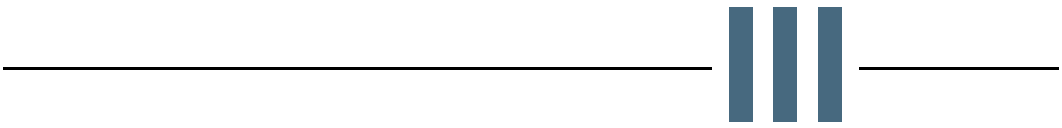
Zoom

The ZOOM command sets the view size of the active Arbortext IsoDraw document.

Syntax

ZOOM *size*

size	View size for the active document as string or float. Possible values are PAGE, EXTENT or any integer from 0.01 to 6400 to specify a scale %.



Window Commands

9 Layer Window

With this group of commands you can directly access most of the features from the Arbortext IsoDraw **Layer** menu.

Add Layer

The `ADD LAYER` command creates a new layer in the active document. This layer automatically becomes the active layer, i.e. all elements which you now generate will be assigned to this layer. The command will also return a reference on the layer it has just created (see Layer Object on page 217).

Syntax

`ADD LAYER` *"name" r g b*

name	Name of the layer.
r, g, b	Identifying color of the layer as RGB. All three color-numbers have to defined separated by a blank to execute the macro command correctly.

Example

```
#no color given
DEFINE lyr AS layer
lyr = ADD LAYER "myNewLayer"

#a new layer with an identification color
ADD LAYER "anotherLayer" 188 0 29

#a more complex example
MACRO AddLayers
   DEFINE i AS integer
   DEFINE k AS integer
   FOR i = 1 TO 8
      k = 32*(i-1)
      ADD LAYER "Layer "+i 255-k k 0
   END FOR
END MACRO
```

Cut Layer

The `CUT LAYER` command cuts the current layer and all its elements from the currently active document and saves them in the program's clipboard. The cut layer can be retrieved from the clipboard at any time through the command paste layer.

Syntax

```
CUT LAYER
```

Copy Layer

With the COPY LAYER command you can copy the active layer into the program's clipboard. The copied layer replaces the former content of the clipboard. The copied layer can be retrieved from the clipboard at any time through the command paste layer.

Syntax

```
COPY LAYER
```

Paste Layer

The PASTE LAYER command pastes the content of the clipboard (provided that the content of the clipboard is a layer) into the currently active document. Even if you have activated another Arbortext IsoDraw file.

Syntax

```
PASTE LAYER
```

Duplicate Layer

With the DUPLICATE LAYER command you can paste a copy of the selected layer into the document. To a certain degree the command is a combination of the commands COPY LAYER and PASTE LAYER. However the selected layer is not saved to the clipboard. This has the advantage that the old content of the clipboard is retained.

Syntax

```
DUPLICATE LAYER
```

Delete Layer

The DELETE LAYER command deletes the currently active layer.

Syntax

```
DELETE LAYER
```

Activate Layer

The ACTIVATE LAYER command will activate an inactive layer. All elements which you now create will be assigned to the newly activated layer.

Syntax

```
ACTIVATE LAYER "name"
```

name	Name of the layer you wish to activate. The name is case-sensitive.

Example

```
ACTIVATE LAYER "Standard layer"
```

Delete All Empty Layers

With the `DELETE ALL EMPTY LAYERS` command you can automatically delete all layers that contain no elements.

Syntax

```
DELETE ALL EMPTY LAYERS
```

Selected Elements to Active Layer

The `SELECTED ELEMENTS TO ACTIVE LAYER` command will move all selected elements to the active layer.

Syntax

```
SELECTED ELEMENTS TO ACTIVE LAYER
```

Selected Text to Active Layer

The `SELECTED TEXT TO ACTIVE LAYER` command will move all selected text elements to the active layer.

Syntax

```
SELECTED TEXT TO ACTIVE LAYER
```

10 Palette Window Toolbox

With this group of commands you can directly access most of the features from the Arbortext IsoDraw **Palette** window toolbox.

Selecting Elements

These commands allow you to select elements with the Arbortext IsoDraw Macro Language: See Edit Menu on page 59 or Arrange on page 68 for additional commands for selecting elements.

Select Rectangle

The `SELECT RECTANGLE` command selects all elements within the given rectangle.

Syntax

`SELECT RECTANGLE` *x1 y1 x2 y2* `[WITH_PARTIAL][DIRECT][ADD]`

x1, y1	Start point of the diagonal of the rectangle.
x2, y2	End point of the diagonal of the rectangle.
WITH_PARTIAL	Optional parameter defines that elements partially within the rectangle will also be selected. If this parameter is not given only elements completely inside rectangle will be selected.
DIRECT	Optional parameter defines that elements within a group will be selected even if the entire group is not within the selection rectangle.
ADD	Optional parameter add defines that this new selection will be added to an existing selection, rather than replacing it. This is similar to using the SHIFT key when making multiple selections manually.

Example

```
SELECT RECTANGLE 12.45 22 (-10.23) 289
```

Select Polygon Start

The `SELECT POLYGON START` begins the process of creating a selection polygon.

Syntax

```
SELECT POLYGON START
```

Select Polygon Points

The SELECT POLYGON POINTScommand allows points for the selection polygon to be specified.

Syntax

`SELECT POLYGON POINTS` *p1* [*p2*] [...] [*pn*]

p1	Start point for the polygon as point.
p2 ... pn	Optional parameters define additional points for the polygon selection.

Select Polygon End

The SELECT POLYGON END command end the selection polygon command.

Syntax

```
SELECT POLYGON END[WITH_PARTIAL][DIRECT][ADD]
```

WITH_PARTIAL	Optional parameter defines if partially selected elements should be selected.
DIRECT	Optional parameter defines that elements within a group will be selected even if the entire group is not within the selection polygon.
ADD	Optional parameter defines that this new selection will be added to an existing selection, rather than replacing it. This is similar to using the SHIFT key when making multiple selections manually.

Example

```
# Select everything in the lower right
DEFINE px AS Integer
DEFINE py AS Integer
px = activeDoc.window.PageX
py = activeDoc.window.PageY
SELECT POLYGON START
SELECT POLYGON POINTS 0 0 px 0
SELECT POLYGON POINTS px py
SELECT POLYGON DIRECT ADD
```

Select At

The `SELECT AT` command selects an element or just a part of an element in the current document at the given coordinates.

Syntax

`SELECT AT` *x y* `[TOGGLE | PART | DIRECT] GROUP`

x, y	Define the click-point.
TOGGLE	Used for the multiple selection of different elements. If one element was already selected the selection is canceled. Corresponds to selecting with the SHIFT key.
PART	Used for adding a line-segment or a bézier-segment to a selection. Corresponds to selecting with the ALT key pressed.

DIRECT	Corresponds to selecting with the **Direct Selection** arrow cursor.
GROUP	Used for electing one group of a nested group containing this element. Corresponds to selecting with the CTRL key pressed if the cursor is the **Direct Selection** arrow cursor.

Transform Selection

Scale Selection

Use the `SCALE SELECTION` command to scale the elements that you have selected.

Syntax

`SCALE SELECTION` *x y m_x m_y* `[COPY][NO_ELEMENTS][NO_PATTERNS]`

x, y	Corner, and therefore, in what direction you want to scale the elements.
m_x, m_y	Scaling of the x- and y-axis. These parameters will multiply the axis. i.e. every number below one is for shrinking the selection, one does no scaling at all and everything over one is for enlarging the selection.
COPY	(optional) Scales and copies the object instead of just scaling it.
NO_ELEMENTS	(optional) Scales patterns with reference to elements.
NO_PATTERNS	(optional) Scales only the elements.

Example

```
SCALE SELECTION 432.2 223.33 1.5 1.5 COPY
```

Shear Selection

Applies to Arbortext IsoDraw 7.0 F000 and later:

Use the `SHEAR SELECTION` command to shear the selected elements and/or patterns.

Syntax

`SHEAR SELECTION` *ref_x ref_y shear_h shear_v* `[COPY][NO_ELEMENTS][NO_PATTERNS]`

ref_x, ref_y	Reference point for shearing.
shear_h	(`float`) Shear factor in the horizontal direction.
shear_v	(`float`) Shear factor in the vertical direction.
COPY	(optional) Copies selected elements before shearing.
NO_ELEMENTS	(optional) Excludes elements from shearing.
NO_PATTERNS	(optional) Excludes patterns from shearing.

Example

```
SHEAR SELECTION 225.2 533.45 15 30 COPY NO_PATTERNS
```

Rotate Selection

The ROTATE SELECTION command rotates an object around a point.

Syntax

ROTATE SELECTION *x y* ANGLE[COPY][NO_ELEMENTS][NO_PATTERNS]

x,y	Center point you want to rotate the selected object around.
ANGLE	Rotation angle.
COPY	Optional parameter copy is for rotating and copying the object instead of just rotating it.
NO_ELEMENTS	Optional parameter to rotate patterns with reference to elements.
NO_PATTERNS	Optional parameter to rotate only the elements.

Example

```
rotate selection 345.99 22.2
33.77 no_patterns
```

Reflect Selection

The REFLECT SELECTION command reflects an object around a point.

Syntax

REFLECT SELECTION *x y* ANGLE[COPY][NO_ELEMENTS][NO_PATTERNS]

x,y	Point where the center of reflection is to lie.
ANGLE	Rotation angle.
COPY	Optional parameter is for reflecting and copying the object instead of just reflecting it.
NO_ELEMENTS	Optional parameter to relect patterns with reference to elements.
NO_PATTERNS	Optional parameter to reflect only the elements.

Example

```
REFLECT SELECTION 111 332.666 33.5
```

Create Parallels

The CREATE PARALLELS command will generate parallel paths at a specified distance from the original elements.

Syntax

CREATE PARALLELS *dist* BOTH_SIDES DELETE_ORIGINAL

DIST	Distance from the original element the parallels will be drawn in mm.
BOTH_SIDES	Optional parameter to draw both sides of the parallels.
DELETE_ORIGINAL	Optional parameter to delete original element after drawing the parallels.

Example

```
CREATE PARALLELS 10.75 BOTH_SIDES DELETE_ORIGINAL
```

Transforming the Illustration

Set Transform

The `SET TRANSFORM` command creates a transformation matrix. This matrix is used for all following commands until a restore command is executed. Therefore you should always restore the transformation matrix to normal with `RESTORE TRANSFORM` restore transform after using `SET TRANSFORM`.

Syntax

`SET TRANSFORM` *x y angle scale*
`SET TRANSFORM MATRIX` *2D matrix*

x,y	Define the offset of the displacement.
ANGLE	Defines the angle of the displacement.
SCALE	Defines the scaling of the displacement.

Example

```
MACRO trans

   DEFINE i AS integer
   DEFINE n AS integer
   DEFINE x AS integer
   DEFINE y AS integer

   SET TRANSFORM 300 200 0 0.5
      FOR n = 0 TO 75 STEP 15
         SET TRANSFORM 0 0 n 1
            FOR i = 90 TO 360 STEP  90
               IF (i = 90)
                  x = 0
                  y = 200
               END IF
               IF (i = 180)
                  x = -200
                  y = 0
               END IF
               IF (i = 270)
                  x = 0
                  y = -200
```

```
            END IF
            IF (i = 360)
               x = 200
               y = 0
            END IF
            SET TRANSFORM x y i 1
               CREATE LINE 0 0 100 0
            RESTORE TRANSFORM
         END FOR
      RESTORE TRANSFORM
   END FOR
RESTORE TRANSFORM
END MACRO
```

Restore Transform

The `RESTORE TRANSFORM` command removes the last transformation matrix from the transformation chain.

Syntax

```
RESTORE TRANSFORM
```

Absolute

If the `ABSOLUTE` mode is set all transformation matrices are ignored. All coordinates are interpreted as absolute coordinates until the absolute mode is turned off.

Syntax

```
ABSOLUTE {ON | OFF}
```

Creating Elements

All CREATE commands (ellipse, line, rectangle ...) return a reference on the created element. The properties of this element object can be queried and set (see Element Object on page 193).

Create Line

The `CREATE LINE` command plots a line in your Arbortext IsoDraw Document.

Syntax

`CREATE LINE` *x1 y1 x2 y2*

`x1 y1 x2 y2`

x1, y1	Starting point of the line.
x2, y2	End point of the line.

Example

```
CREATE LINE 189 10 20.23 200.99
```

Append Line Segment

The `APPEND LINE SEGMENT` command appends a line to the last line created by the `CREATE LINE` command or by an `APPEND LINE` command. To append a line for the first time there has to be a line created by the `CREATE LINE` command,

The start point for the appended line is the end point of the line plotted by the `CREATE LINE` command or . the `APPEND LINE SEGMENT` command.

Syntax

`APPEND LINE SEGMENT` *x y*

x, y	End point of the line segment.

Example

```
MACRO house_of_nicholas

#first create a line by a macro command
CREATE LINE 100 100 200 100

#second append line segments
APPEND LINE SEGMENT 200 200
APPEND LINE SEGMENT 100 100
APPEND LINE SEGMENT 100 200
APPEND LINE SEGMENT 200 200
APPEND LINE SEGMENT 150 250
APPEND LINE SEGMENT 100 200
APPEND LINE SEGMENT 200 100

END MACRO
```

Create Ellipse

The `CREATE ELLIPSE` command plots an ellipse.

Syntax

`CREATE ELLIPSE` *x y radius angle value*

x, y	Center of the ellipse.
radius	Radius of the ellipse.
angle	Ellipse angle.
value	Ellipse value.

Example

```
CREATE ELLIPSE 113.34 112.99 99 23.5 27
```

Create Inner Thread

The `CREATE INNER THREAD` command creates an inner thread.

Syntax

`CREATE INNER THREAD` *x y radius angle value* [*depth*]

x, y	Center of the ellipse.
radius	Radius of the ellipse.
angle	Ellipse angle.
value	Ellipse value.
depty	Optional parameter describes the depth of the inner thread. If the parameter is set to -1 or missing the inner thread will be completely filled.

Example

```
CREATE INNER THREAD 342.947 222.495 50 240 35.264
```

Create Outer Thread

The `CREATE OUTER THREAD` command creates an outer thread.

Syntax

`CREATE OUTER THREAD` *x y radius angle value* [*depth*]

x, y	Center of the ellipse.
radius	Radius of the ellipse.
angle	Ellipse angle.
value	Ellipse value.
depty	Optional parameter describes the depth of the outer thread. If the parameter is missing the depth of the outer thread will be set to 0.

Create Callout

The `CREATE CALLOUT` command inserts a callout element into the currently active document.

Syntax

`CREATE CALLOUT` *"style" x1 y1 x2 y2* [*“notation”*]

style	Style is used for the callout. The given style must be defined before using it for the command. The name of the style is case sensitive.
x1, y1	Starting point of the callout. The callout number will appear at the coordinates of the end point. To create a callout without a line the start and end coordinates have to be identical.

x2, y2	End point of the callout.
notation	Optional parameter is needed for callout styles with the scheme set to `NO SCHEME`. In these cases the notation will be used as the callout text. If the scheme of the callout style is not set to `NO SCHEME`, the notation will be ignored.

Example

```
#a callout with automatic numbering
CREATE CALLOUT "Normal" 112.39 110 290.99 100

#creating an element for later modification
DEFINE ele_07 AS element
ele_07 = CREATE CALLOUT "Normal" 112.39 110 290.99 100

#a callout with a given name
#remember: the "No scheme" style must be used
CREATE CALLOUT "myStyle" 15 127.25 15 115 "AX_22"
```

Create Rectangle

The `CREATE RECTANGLE` command plots a rectangle in active document.

Syntax

`CREATE RECTANGLE` *p_x s sp_y lr_x lr_y ep_x ep_y ul_x ul_y* `[rounded]` [*angle*] [*value*]

sp_x. sp_y	Starting point of the rectangle.
lr_x, lr_y	Lower right corner of the rectangle.
ep_x, ep_y	End point of the rectangle.
ul_x, ul_y	Upper left corner of the rectangle.
ROUNDED	Optional parameter used for defining the rounded corners of the rectangle. Where `ROUNDED` is for initiating, *value* for the ellipse value and *angle* is for the ellipse angle of the rounding.
angle	Optional parameter used for defining the rounded corners of the rectangle. Where `ROUNDED` is for initiating, *angle* is for the ellipse angle of the rounding.
value	Optional parameter used for defining the rounded corners of the rectangle. Where `ROUNDED` is for initiating, *value* is for the ellipse value of the rounding.

Example

```
#a normal rectangle
CREATE RECTANGLE 236.881 222.495 236 132.681 314 87 314 177

#a rectangle with rounded corners
CREATE RECTANGLE 236 222 236.881 132.681 314 87.772 314 177.587
ROUNDED 90 90
```

Create Polygon

The `CREATE POLYGON` command plots a polygon.

Syntax

`CREATE POLYGON` *x y corners radius angle value*

x,y	Define the center of the polygon.
corners	Defines how many corners (or sides) the polygon is going to have.
radius	Because a polygon behaves like an ellipse while being drawn, you also have to define a radius (see Create Ellipse on page 102).
angle	Because a polygon behaves like an ellipse while being drawn, you also have to define an angle as the ellipse angle (see Create Ellipse on page 102).
value	Because a polygon behaves like an ellipse while being drawn, you also have to define a value as a ellipse value (see Create Ellipse on page 102).

Example

```
#drawing an octagon
CREATE POLYGON 381.838 212.288 8 65 300 35.264
```

Create Bezier Curve

The CREATE BEZIER CURBE command draws a bezier curve.

Syntax

`CREATE BEZIER CURVE` *point1 handle1 handle2 point2*

point1	x- and y-coordinates of the starting point of the bézier curve and the starting point of the first handle.
handle1	x- and y-coordinates of the endpoint of the first handle. As a result, this also defines the length and the heading of the first handle.
handle2	x- and y-coordinates of the endpoint of the second handle. As a result, this also defines the length and the heading of the second handle.
point2	x- and y-coordinates of the end point of the bézier curve.

Example

```
CREATE BEZIER CURVE 100 100 150 100 150 200 200 200
```

Append Bezier Segment

The `APPEND BEZIER SEGMENT` command appends a bézier curve to the last bézier curve created by the `CREATE BEZIER CURVE` command or by an `APPEND BEZIER SEGMENT` command. To append a bézier curve for the first time there has to be a bézier curve created by the `CREATE BEZIER CURVE` command.

The start point for the appended bézier curve is the end point of the bézier curve created by the `CREATE BEZIER CURVE` command or. the `APPEND BEZIER SEGMENT` command.

Syntax

`APPEND BEZIER SEGMENT` *handle1 handle2 point*

handle1	x- and y-coordinates of the endpoint of the first handle. As a result, this also defines the length and the heading of the first handle.
handle2	x- and y-coordinates of the endpoint of the second handle. As a result, this also defines the length and the heading of the second handle.
point	x- and y-coordinates of the end point of the bézier curve.

Example

```
CREATE BEZIER CURVE 100 100 150 100 150 200 200 200
APPEND BEZIER SEGMENT 250 200 250 100 300 100
APPEND BEZIER SEGMENT 350 100 350 200 400 200
APPEND BEZIER SEGMENT 450 200 450 100 500 100
```

Create Text

The CREATE TEXT command creates a text element in the active document.

Syntax

`CREATE TEXT` *x y "text"*

x,y	Lower left corner of the text box.
text	String defines the text that is created.

Example

```
CREATE TEXT 100 100 "The Arbortext IsoDraw Macro Language"
```

Change Text At

The `CHANGE TEXT AT` command changes a text element at a specific position on the current document.

Syntax

`CHANGE TEXT AT` *x y"text"*

x, y	Position of the text element that will be changed as float.
text	New text as string.

Example

```
CHANGE TEXT AT 100 100 "Arbortext IsoDraw 6"
```

Set Ellipsevalues

The `SET ELLIPSEVALUES` command sets the ellipse value or *angle* used during the creation of new ellipses.

Syntax

`SET ELLIPSEVALUES` *angle*

angle	Ellipse angle between 1 and 90 as float.

Example

```
SET ELLIPSEVALUES 45
```

11 Attribute Window

With this group of commands you can directly access most of the features in the Arbortext IsoDraw attribute window.

Pens

Add Pen

With the `ADD PEN` command you can create a new pen. The new pen will be added to the currently active document and inherit the attributes of the currently active pen. The new pen will be set to active.

Note

If there is no document open, the created pen will be a new standard pen available in all new documents.

Syntax

`ADD PEN` *"name"* [*width*]

name	Name of the pen after creation. If there is already a pen with that name the macro command can not be executed. You cannot use the following characters within a pen name: Forward slash (/) Circumflex accent (^) Exclamation point (!) Less than (<) Left parenthesis “(“ Colon (:)
width	Optional parameter defines the width of the pen in mm.

Example

```
ADD PEN "highlight_this"
ADD PEN "BIG" 5.0
```

Delete Pen

The `DELETE PEN` command deletes an existing pen from the currently active document.

Note

With no document open a standard pen will be deleted.

Syntax

`DELETE PEN` *"name" "substName"*

name	Name of the pen that is going to be deleted. If no pen with that name exists the command can not be executed.
substName	Name of the pen that is substituted. If no pen with that name exists the command can not be executed.

Example

```
DELETE PEN "BIG" "$ISO_THICK"
```

Set Active Pen

The `ACTIVE PEN` command sets the active pen to the one specified.

Syntax

`SET ACTIVE PEN` *"name"*

name	Name of the pen that is going to be set as active.

Example

```
SET ACTIVE PEN "$ISO_THICK"
```

Set Lineoptions

The `SET LINEOPTIONS` command lets you change the print options for particular elements. If elements are selected when executing the command the options are changed for these elements. If no elements are selected when executing this command the print options for the whole active document are changed. Every element created after executing the command will then have these options.

Syntax

`SET LINEOPTIONS` *ends corners overprint miter_limit*

ends	Type of line ends.
corners	Type of line corners. This option is only significant for Bézier paths with corner points.
overprint	Type of overprinting you want to have. The values for this option are 0 for no overprint, 1 for overprinting the stroke, 2 for overprinting the fill and 3 for overprinting stroke and fill.
miter_limit	Limit the miter corner for sharp angles.

Example

```
SET LINEOPTIONS 2 0 0 4
```

Toggle Pens

Applies to Arbortext IsoDraw 7.0 F000 and later:

The `TOGGLE PENS` command switches the thick and thin attributes of any selected element; for example, any thick line will be switched to a thin line and vice-versa. (Double-clicking on a contour in the Arbortext IsoDraw drawing window produces the same result.)

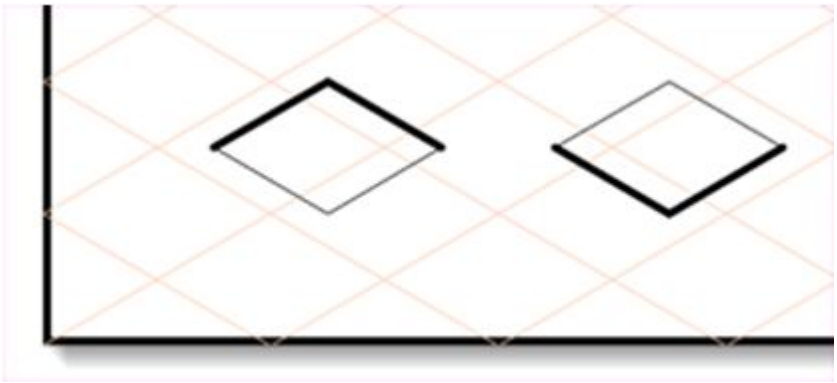
Syntax

```
TOGGLE PENS
```

Example

```
MACRO Switch_All_Thick_Thin
 SELECT ALL
 TOGGLE PENS
END MACRO
```

`TOGGLE PENS` Example Result



Styles

Add Style

The `ADD STYLE` command creates a new style. This new style will be added to the currently active document. If there is already an active style the added style will inherit the attributes of the active style. After the new style is added to the document this style will be the new active style, used for all following elements.

Note

If there is no document open, the created style will be a new standard style available in all new documents.

Syntax

`ADD STYLE` *"name" pattern*

name	Name of the new style. If there is already a style with that name the command can not be executed. The following characters are forbidden within a style name: Forward slash (/) Circumflex accent (^) Exclamation point (!) Less than (<) Left parenthesis “(“ Colon (:)
pattern	Optional parameter consists of 6 floats defining the different strokes and gaps of the patterns: *stroke1 gap1 stroke2 gap2 stroke3 gap3*

Example

```
ADD STYLE "myStyle"

#adding a style with the optional patterns
ADD STYLE "myNewStyle" 0.5 2.5 0.5 3.5 1.5 1.5
```

Delete Style

The `DELETE STYLE` command deletes an existing style out of the style list of the currently active document.

Note

With no document open a standard style will be deleted.

Syntax

`DELETE STYLE` *"name" "substName"*

name	Name of the style that is going to be deleted. If no style with that name exists the command can not be executed.
substName	Name of the style that is substituted. If no style with that name exists the command can not be executed.

Example

```
DELETE STYLE "myNewStyle" "Dashed"
```

Set Active Style

The `ACTIVE STYLE` command sets the active style to the one specified.

Syntax

`SET ACTIVE STYLE` *"name"*

name	Name of the style that is going to be set as active.

Example

```
SET ACTIVE STYLE "Dashed"
```

Shadows

Add Shadow

With the `ADD SHADOW` command you can create a new shadow. The new shadow will be added into the currently active document and inherit the attributes of the currently active shadow. The new shadow will be set to active.

Note

If there is no document open, the created shadow will be a new standard shadow available in all new documents.

Syntax

`ADD SHADOW` *"name"*

name	Name of the shadow after creation. If there is already a shadow with that name the macro command can not be executed. The following characters are not allowed within in the name: Forward slash (/) Circumflex accent (^) Exclamation point (!) Less than (<) Left parenthesis “(“ Colon (:)

Example

```
ADD SHADOW "myShadow"
```

Delete Shadow

The `DELETE SHADOW` command deletes an existing shadow out of the shadow list of the currently active document.

Note

With no document open a standard shadow will be deleted.

Syntax

`DELETE SHADOW` *"name" "substName"*

name	Name of the shadow that is going to be deleted. If no shadow with that name exists the command can not be executed.
substName	Name of the shadow that is substituted. If no shadow with that name exists the command can not be executed.

Example

```
DELETE SHADOW "myShadow" "Autom. Long"
```

Set Active Shadow

The `SET ACTIVE SHADOW` command sets the active shadow to the one specified.

Syntax

`SET ACTIVE SHADOW` *"name"*

name	Name of the shadow that is going to be set as active.

Example

```
SET ACTIVE SHADOW "Autom. Long"
```

Grids

Add Grid

The `ADD GRID` command creates a new grid and adds it to the currently active document.

Note

If there is no document open, the new grid will be a new standard grid available in all new documents.

Syntax

`ADD GRID` *"name"* [*z_angle*] [*x_angle*]

name	Name of the grid after creation. If there is already a grid with that name the macro command can not be executed. You cannot use the following characters within a macro name: Forward slash (/) Circumflex accent (^) Exclamation point (!) Less than (<) Left parenthesis “(“ Colon (:)
x_angle, z_angle,	Optional parameter defines the x and z axis. The angles can range from 0° to 90° but totaled they should not exceed 90° (included). The y-axis is not specified as it will always be vertical. If the angle parameters are missing the plane angle of the first two selected lines is taken to build the grid. If only one line is selected the grid will be build as isometric-grid (30° - 30°).

Example

```
MACRO CreateGridArray
   DEFINE a AS Integer
   DEFINE b AS Integer
   FOR a=10 To 70 STEP 10
      FOR b=10 TO 80-a STEP 10
         ADD GRID "[ "+a+" - "+b+ " ]" a b
      END FOR
   END FOR
END MACRO
```

Delete Grid

Thr `DELETE GRID` command deletes an existing grid out of the grid list of the currently active document.

Note

With no document open a standard grid will be deleted.

Syntax

`DELETE GRID` *"name" "substName"*

name	Name of the grid that is going to be deleted. If no grid with that name exists the command can not be executed.
substName	Name of the grid that is substituted. If no grid with that name exists the command can not be executed.

Example

```
DELETE GRID "[ 10 - 20 ]" "Plane"
```

Formats

Add Format

With the `ADD FORMAT` command you can create a new format. The new format will be added into the currently active document and inherit the attributes of the currently active format. The new format will be set to active.

Note

If there is no document open, the created format will be a new standard format available in all new documents.

Syntax

`ADD FORMAT` *"name"*

name	Name of the format after creation. If there is already a format with that name the macro command can not be executed. You cannot use the following characters within a format name: Forward slash (/) Circumflex accent (^) Exclamation point (!) Less than (<) Left parenthesis “(“ Colon (:)

Example

```
ADD FORMAT "myFormat"
```

Delete Format

The `DELETE FORMAT` command deletes an existing format out of the format list of the currently active document.

Note

With no document open a standard format will be deleted.

Syntax

`DELETE FORMAT` *"name" "substName"*

name	Name of the format that is going to be deleted. If no format with that name exists the command can not be executed.
substName	Name of the format that is substituted. If no format with that name exists the command can not be executed.

Example

```
DELETE FORMAT "myFormat" "Normal"
```

Set Active Format

The `SET ACTIVE FORMAT` command sets the active format to the one specified.

Syntax

`SET ACTIVE FORMAT` *"name"*

name	Name of the format that is going to be set as active.

Example

```
SET ACTIVE FORMAT "Normal"
```

Viewports

Add Viewport

With the `ADD VIEWPORT` command you can create a new viewport. The new viewport will be added into the currently active document and inherit the attributes of the currently active viewport. The new viewport will be set to active.

Note

If there is no document open, the created viewport will be a new standard viewport available in all new documents.

Syntax

`ADD VIEWPORT` *"name"*

name	Name of the viewport after creation. If there is already a viewport with that name the macro command can not be executed. You cannot use the following characters within a viewport name: Forward slash (/) Circumflex accent (^) Exclamation point (!) Less than (<) Left parenthesis “(“ Colon (:)

Example

```
ADD VIEWPORT "myViewport"
```

Delete Viewport

The `DELETE VIEWPORT` command deletes an existing viewport out of the viewport list of the currently active document.

Note

With no document open a standard viewport will be deleted.

Syntax

`DELETE VIEWPORT` *"name"*

name	Name of the viewport that is going to be deleted. If no viewport with that name exists the command can not be executed.

Example

```
DELETE VIEWPORT "myViewport"
```

Execute Viewport

With the `EXECUTE VIEWPORT` command you can jump to a given viewport.

Syntax

`EXECUTE VIEWPORT` *"path" "name"*

path	Path and name of an existing Arbortext IsoDraw or CGM File. To activate the currently active file the path parameter can be an empty string.
name	Name of an existing viewport of the file given in the path parameter or the currently active file. The name parameter is case sensitive.

Example

```
EXECUTE VIEWPORT "" "detail_1"
EXECUTE VIEWPORT  "c:\data\illustration\Mz445_ttr.iso" "seat"
```

Add Layerstatus

With the `ADD LAYERSTATUS` command a layer can be added to a viewport.

Syntax

`ADD LAYERSTATUS` *"layer_name"* `TO` *"viewport_name"*

layer_name	Name of the layer to be added.
viewport_name	Name of the viewport the layer will be added to.

Example

```
ADD LAYERSTATUS "My_Layer" TO "My_Viewport"
```

Remove Layerstatus

With the `REMOVE LAYERSTATUS` command a layer can be removed from a viewport.

Syntax

`REMOVE LAYERSTATUS` *"layer_name"* `FROM` *"viewport_name"*

layer_name	Name of the layer to be removed.
viewport_name	Name of the viewport the layer will be removed from.

Example

```
REMOVE LAYERSTATUS "My_Layer" FROM "My_Viewport"
```

Callouts

Add Callout_Style

With the `ADD CALLOUT_STYLE` command you can create a new callout style. The new callout style will be added into the currently active document and inherit the attributes of the currently active callout style. The new callout style will be set to active.

Note

If there is no document open, the created callout style will be a new standard callout style available in all new documents.

Syntax

`ADD CALLOUT_STYLE` *"name"*

name	Name of the callout style after creation. If there is already a callout style with that name the macro command can not be executed. You cannot use the following characters within a callout style name: Forward slash (/) Circumflex accent (^) Exclamation point (!) Less than (<) Left parenthesis “(“ Colon (:)

Example

```
ADD CALLOUT_STYLE "myCallout"
```

Delete Callout_Style

The `DELETE CALLOUT_STYLE` command deletes an existing callout style out of the callouts list of the currently active document.

Syntax

`DELETE CALLOUT_STYLE` *"name"* [*"substName"*]

name	Name of the format that is going to be deleted. If no callout style with that name exists the command can not be executed.
substName	Optional parameter defines the name of the callout style that is substituted. If no callout style with that name exists the command can not be executed.

Example

```
DELETE CALLOUT_STYLE "myCallout" "Normal"
```

Renumber Callouts

The `RENUMBER CALLOUTS` command renumbers existing callouts, regardless of the setting specified under Update for the callout style.

Syntax

```
RENUMBER CALLOUTS
```

12 Fill Window

With this group of commands you can directly access most of the features from the Arbortext IsoDraw **Fill** window.

Colors

Add Color

With the `ADD COLOR` command you can create a new color. The new color will be added to the currently open document. The new color will be set to active.

Note

If no document is open the created color will be a new standard color available in all new documents.

Syntax

`ADD COLOR` *"name" color*

name	Name of the color after creation. If there is already a color with that name the macro command can not be executed. Furthermore, it is forbidden to use the following characters within a color name: Forward slash (/) Circumflex accent (^) Exclamation point (!) Less than (<) Left parenthesis "(" Colon (:)
color	Optional parameter defines the color as a colorSpec.

Example

```
MACRO Add Color

DEFINE red AS RGBColor
red.red = 188
red.green = 0
red.blue = 29

DEFINE color_red AS colorSpec
color_red.type = "rgbValues"
color_red.rgb = red

ADD COLOR "RED_188_0_29" color_red

END MACRO
```

Delete Color

The `DELETE COLOR` command deletes an existing color out of the color list of the currently active document.

Note

With no document open a standard color will be deleted.

Syntax

`DELETE COLOR` *"name" "substName"*

name	Name of the color that is going to be deleted. If no color with that name exists the command can not be executed.
substName	Name of the color that is used as substitution. If no color with that name exists the command can not be executed.

Example

```
DELETE COLOR "RED_188_0_29" "Black"
```

IV

3D and User Interaction Commands

13 3D Commands

Applies to Arbortext IsoDraw CADprocess only.

The 3D commands can only be executed if the current active document is opened in 3D mode, otherwise an error message will appear.

3D View

The `3D VIEW` command changes the camera angle of the 3D scene.

Syntax

`3D VIEW` *view*

view	Camera angle as integer. Allowed values are:

X	View X
Y	View Y
Z	View Z
ISOMETRIC_TOP	Isometric view top
ISOMETRIC_BOTTOM	Isometric view bottom
DIMETRIC_1	Dimetric view 1
DIMETRIC_2	Dimetric view 2
DIMETRIC_3	Dimetric view 3
DIMETRIC_4	Dimetric view 4
TRIMETRIC	Trimetric view
PERSPECTIVE	Perspective view

Example

```
3D VIEW DIMETRIC_2
```

3D SetView

The `3D SETVIEW` command is included for compatibility reasons only. It is not recommended that you use this command for new development of macros as it may be discontinued in the future. All features from this command have been incorporated into the new command `3D VIEW`.

The `3D SETVIEW` command changes the camera angle of the 3D scene.

Syntax

`3D SETVIEW` *view*

view	Camera angle as integer. Allowed values are: **900** View X **901** View Y **902** View Z **903** Isometric view top **904** Isometric view bottom **905** Dimetric View 1 **906** Dimetric View 2 **907** Dimetric View 3 **908** Dimetric View 4 **909** Trimetric View **910** Perspective View

Example

```
3D SETVIEW 903
```

3D Project

The `3D PROJECT` command converts the current 3D mode view to a 2D illustration (similar to clicking the **Convert to 2D Illustration** (Camera) button on the **3D Tools** toolbar). The projection is controlled by the properties of the `app.project3D` object. (See app.project3D on page 233.)

Syntax

`3D PROJECT` [`0 | 1` | *same*]

0 (or no parameter)	Converts the 3D view in the active window to a 2D illustration in a new, untitled window. After conversion, the 3D view window remains open and active. The 2D illustration window also remains open, but is not saved.
1	Converts the 3D view in the active window to a 2D illustration in the same window and switches the active window from 3D mode to 2D mode. ***Caution*** *3D data is not kept after conversion to 2D, therefore, saving the new 2D illustration with the original name could overwrite the original 3D file and result in a loss of 3D data.*
same	This parameter is being ignored for 2D projections of placed 3D files.

Example

```
3D PROJECT 1
```

3D Center

The 3D CENTER command allows you to align all the assemblies of the drawing such that the coordinate origin is at the center point of all assemblies.

Syntax

```
3D CENTER
```

3D ZoomExtent

The 3D ZOOMEXTENT command is included for compatibility reasons only. It is not recommended that you use this command for new development of macros as it may be discontinued in the future. All features from this command have been incorporated into the command ZOOM (see Zoom on page 87).

The 3D ZOOMEXTENT command displays the entire drawing on the screen.

Syntax

```
3D ZOOMEXTENT
```

3D HLRMode

The 3D HLRMODE command is included for compatibility reasons only. It is not recommended that you use this command for new development of macros as it may be discontinued in the future. All features from this command have been incorporated into the command 3D MODE.

This command toggles the view between hidden lines shown and hidden lines removed.

Syntax

3D HLRMODE *mode*

mode	Defines whether hidden lines are shown. Allowed values are: **15** Hidden lines shown. **16** Hidden lines removed.

Example

```
3D HLRMODE 15
```

3D Mode

The `3D MODE` command sets the view mode in the 3D window.

Syntax

`3D MODE` *mode*

mode	View mode displayed. Allowed values are: WIREFRAME HLR FACED SHADED

Example

```
3D HLRMODE WIREFRAM
```

3D Explosion

The `3D EXPLOSION` command gives you the means to automatically explode the components of a larger assembly unit along a specific axis.

Syntax

`3D EXPLOSION` *direction 3d-point*

direction	Axis and the direction you wish to explode the assemblies. Allowed values are:

1	Direction: positive, axis: x.
2	Direction: both, axis: x.
3	Direction: negative, axis: x.
4	Direction: positive, axis.
5	Direction: both, axis: y.
6	Direction: negative, axis.
7	Direction: positive, axis.
8	Direction: both, axis: z.
9	Direction: negative, axis: free.
10	Direction: positive, axis.
11	Direction: both, axis: z.
12	Direction: negative, axis: free.

3d-point	Defines the free-axis as point3 if the user has selected a free axis for explosion

Example

```
#exploding along the x-axis (both directions)
3D EXPLOSION 2 0 0 0

#exploding along a free axis (positive direction)
3D EXPLOSION 10 0.005 0.997 0.077
```

3D Move

The `3D MOVE` command moves all selected assemblies in a given direction.

Syntax

`3D MOVE` *3d-point*

3d-point	Axis the drawing should be moved as point3.

Example

```
#movement along the x-axis
3D MOVE 200 0 0
```

3D Axis

The `3D AXIS` command sets the 3D Select axis; the axis used for panning and rotating selected elements in 3D mode.

Use this command to define the X, Y, or Z axis as the 3D Select axis—or to turn off the 3D Select axis. Alternatively, you can define a free axis as the 3D Select axis. The free axis can intersect the origin and a point or any two points.

Note

A free axis is in addition to the X, Y, and Z coordinate system axes.

Syntax

`3D AXIS` [*x y z* [*x1 y1 z1*] `|` `OFF` `|` `X` `|` `Y` `|` `Z`]

x y z	(optional; `float`) Sets the 3D Select axis to a free axis that intersects the origin and the point (*x*, *y*, *z*). (Separate coordinate values with a space character.)
x y z x1 y1 z1	(optional; `float`) Sets the 3D Select axis to a free axis that intersects the points (*x*, *y*, *z*) and (*x1*, *y1*, *z1*). (Separate coordinate values with a space character.)

OFF	(optional) Removes the 3D Select axis if it exists.
X | Y | Z	(optional) Sets the 3D Select axis to the X, Y, or Z axis. Specifying `X` or `Z` sets both the X and Z axes as 3D Select axes.

Example

```
# No 3D Select axis set:
3D AXIS OFF

# Specifying X axis sets 3D Select axis
# to both X and Z axes
3D AXIS X

# 3D Select axis set to free axis that
# intersects the origin and (1,0,1):
3D AXIS 1 0 1

# 3D Select axis set to free axis that
# intersects points (100,120,70) and (200,120,70)
3D AXIS 100 120 70  200 120 70
```

3D Transform

The `3D TRANSFORM` command creates a 3D transformation on the selected assemblies.

Syntax

`3D TRANSFORM` *3d-matrix*

3d-matrix	Defines the transformation.

Example

```
#stretching the drawing to double size
#along the y-axis
3D TRANSFORM 1 0 0 0  0 2 0 0  0 0 1 0  0 0 0 0
```

3D Reset

The `3D RESET` command removes all 3D transformations from the selected assemblies.

Syntax

`3D RESET`

3D SetDist

The `3D SETDIST` command sets the focal distance measured in millimeters (only used in the perspective view).

Syntax

`3D SETDIST` *value*

value	Focal distance in mm as float.

Example

```
3D SETDIST 25.5
```

3D Hole rectangle

The `3D HOLE RECTANGLE` command creates a 3D cut or 3D transparent rectangle.

Syntax

`3D HOLE RECTANGLE` *point point* `[TRANSPARENT]`

point	Start and end points of the rectangle as float.
TRANSPARENT	Optional parameter defines the rectangle as transparent rather than a cut rectangle.

Example

```
3D HOLE RECTANGLE 0 0 200 250 TRANSPARENT
```

3D Hole polygon start

The `3D HOLE POLYGON START` command begins the definition of a 3D cut or transparent free shape.

Syntax

`3D HOLE POLYGON START[TRANSPARENT]`

TRANSPARENT	Optional parameter defines the free shape as transparent rather than a cut free shape.

Example

```
MACRO Create 3D Cut
   # Cuts a rhombus
   3D HOLE POLYGON START
```

```
   3D HOLE POLYGON POINTS (-50) 0 0 50
   3D HOLE POLYGON POINTS 50 0 0 (-50)
   3D HOLE POLYGON END
END MACRO
```

3D Hole polygon points

The `3D HOLE POLYGON POINTS` command defines a point in a 3D cut or transparent free shape.

Syntax

`3D HOLE POLYGON POINTS` *point* [...]

point	Points on the free path as float. Many points can be listed for a single 3D Hole polygon points command.

Example

```
3D HOLE POLYGON POINTS (-50) 0 0 50
```

3D Hole polygon end

The `3D HOLE POLYGON END` command ends the definition of a 3D cut or transparent free shape.

Syntax

```
3D HOLE POLYGON END
```

14 Further Macro Commands

FWrite

The FWRITE command writes a string with a new line character to the end of a text file. If the file doesn't exist it will be created.

Syntax

FWRITE *"path" "message"*

message	Text of message.
path	Path where message is written.

Example

```
DEFINE txtOut AS string
txtOut = "D:\work\output.txt"
FWRITE txtOut "Results:"
```

FNew

The FNEW command creates a new text file with the name a location specified. If the file already exists it will be overwritten.

Syntax

FNEW *"path"*[8_BIT]

path	File path.
8_BIT	Optional parameter defines that the file will be created as a 8-Bit ASCII text file. By default the file is created as a UNICODE text file.

Example

```
DEFINE txtOut AS string
txtout = "C:\Temp\test.txt"
FNEW txtout
FWRITE txtout "Hello World!"
```

Log

With the command LOG you can write your own messages into the log file.Within the Arbortext IsoDraw working folder you can find a file named macro.log. All errors and warnings which occur during the execution of a macro are written into that file by the Arbortext IsoDraw Macro Language.

Syntax

LOG *"message"*

message	Message written into the macro.log file.

Example

```
LOG "Demo finished"

#this also works
FOR i=0 TO 10
   LOG i
END FOR
```

Menu

The MENU command simulates the selection of an Arbortext IsoDrawmenu item. If a dialog is opened by the selection it will appear and pause the macro until closed by the user.

Syntax

MENU *"menupath"* [WITH_SHIFT | WITH_ALT]

menupath	The command searches for the first menu command beginning with the complete search-string. Because menu commands differ from language to language all macros using the menu command are language specific. The MENU command can be used to start a plugin (like Arbortext IsoCompose) and can also reach all commands from the popup menus. You can not use the MENU command to open any item from the **Help**-menu and you only have access to menu commands that are available at the time the command is executed.
WITH_SHIFT, WITH_ALT	MENU comand can be extended by either one of the 2 optional specifiers WITH_SHIFT or WITH_ALT to simulate a pressed SHIFT- or ALT-key.

Example

```
MENU "element info"

MENU "Start IsoCompose"
```

Debugging Commands

The Arbortext IsoDraw Macro Language includes special commands that are useful when debugging. Remove these debugging commands when your macro is working properly.

Debug Step

The `DEBUG STEP` command turns the step-by-step execution of a macro command on or off.

Syntax

```
DEBUG STEP {ON | OFF}
```

ON	Turns step-by-step execution of a macro command on.
OFF	Turns step-by-step execution of a macro command off.

Example

```
MACRO debug
   DEBUG STEP ON
   #first create a line by a macro command
   CREATE LINE 100 100 200 100
   #second append line segments
   APPEND LINE SEGMENT 200 200
   APPEND LINE SEGMENT 100 100
   APPEND LINE SEGMENT 100 200
   APPEND LINE SEGMENT 200 200
   APPEND LINE SEGMENT 150 250
   APPEND LINE SEGMENT 100 200
   APPEND LINE SEGMENT 200 100
   DEBUG STEP OFF
END MACRO
```

Debug Commands

This command displays the **Macro Language** dialog box with a text box you can use to enter and run one macro command at a time. After you enter a macro command, click **OK** to run it.

After the macro command successfully executes, the text box will be cleared and is ready for the next input. Click **Cancel** to close the dialog box.

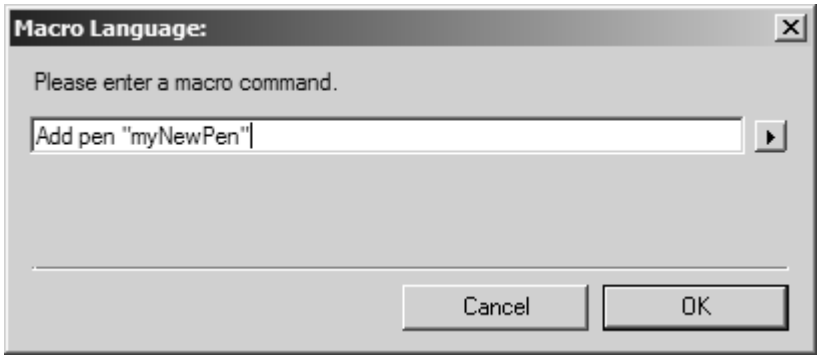
Applies to Arbortext IsoDraw 7.0 F000 and later:

Click the arrow to the right of the text box to select one of the last ten sucessfully executed macro commands from a list.

Syntax

```
DEBUG COMMANDS
```

Example



Debug Reset

The `DEBUG RESET` command termintates the execution of the current macro, discards all variables and reloads all available macros.

Syntax

```
DEBUG RESET
```

Debug Stack

The `DEBUG STACK` command opens a window listing the call stack of .nested sub-macros.

Syntax

```
DEBUG STACK
```

Debug Locals

The `DEBUG LOCALS` command opens a window listing all defined local variables.

Syntax

```
DEBUG LOCALS
```

Debug Globals

The `DEBUG GLOBALS` command opens a window listing all defined global variables.

Syntax

```
DEBUG GLOBALS
```

Dump

The DUMP command writes internal macro processor information to a text file for debugging purposes.

Syntax

DUMP *key filename*

key	Specifies the type of internal macro processor information to dump. Allowed values are: **COMMANDS** — List of all available macro commands (the first keyword of each). **MACROS** — List of all macros currently loaded in memory. **STACK** — List of all macros and subMacros currently on the stack. **GLOBALS** — List of all variables defined with a global scope. **LOCALS** — Currently defined local variables of the macro in which the DUMP command has been used. **FUNCTIONS** — List of all defined IML functions.
filename	(string) Location and filename of the DUMP text file. If you specify a new *filename*, a new file will be created. If you specify an existing *filename*, the new DUMP information will be appended to the end of the existing file. (Existing DUMP files in the specified location are not overwritten.)

Example

```
MACRO Dump_all
 DEFINE sOut AS string
 sOut = "C:\IML_dump.txt"
 FNEW sOut
 FWRITE sOut "----- Loaded macros: -----"
 DUMP MACROS sOut
 FWRITE sOut "----- Call stack: -----"
 DUMP STACK sOut
 FWRITE sOut "----- Global variables: -----"
 DUMP GLOBALS sOut
 FWRITE sOut "----- Local variables: -----"
 DUMP LOCALS sOut
END MACRO
```

After this example Dump_all macro executes, the file IML_dump.txt could contain the following information:

```
----- Loaded macros: -----
Macro Dump_all
----- Call stack: -----
----- Global variables: -----
----- Local variables: -----
String: sOut = "C:\IML_dump.txt"
```

Wait Timer

With the `WAIT TIMER` command you can stop the application for a given time period. The current window will be redrawn beforehand.

Syntax

`WAIT TIMER` *ticks*

ticks	Length of time the application will be stopped as an integer. One tick is 1/60 of a second.

Example

```
MACRO movement
   DEFINE i AS integer
   DEFINE n AS integer
   FOR i = 0 TO 90 STEP 15
      CREATE ELLIPSE 100 200 20 180 i
      CREATE ELLIPSE  100 200 20 90 i
   END FOR
   SELECT ALL
   GROUP SELECTION
   FOR n=2 TO 400 STEP 5
      WAIT TIMER 1
      MOVE SELECTION 5 1
   END FOR
END MACRO
```

Sleep

Applies to Arbortext IsoDraw 7.0 F000 and later:

Puts the application asleep for the defined period of time. The current window, menu and toolbars will be refreshed before. This command releases the CPU for other applications.

Syntax

`SLEEP` *milliseconds*

milliseconds	(`integer`) Time in milliseconds.

Example

```
#Sleep for one-half second:
SLEEP 500
```

Launch

Applies to Arbortext IsoDraw 7.0 F000 and later:

Starts an external application.

Syntax

`LAUNCH` *"app" "cmd"*

app	(`string`) Filename and path to the external application to be started.
cmd	(`string`) Parameters to be passed to the application on start-up.

If the application starts, the LAUNCH command returns the application's process ID. If the application fails to start, the `LAUNCH` command returns zero.

Example

```
# Open Macro-Logfile with Notepad
MACRO Show_Logfile
 LAUNCH "C:\WINDOWS\notepad.exe" "C:\ ...  \macro.log"
END MACRO
```

Terminate

Applies to Arbortext IsoDraw 7.0 F000 and later:

Stops an external application by its process ID (PID).

Syntax

`TERMINATE` *pid*

pid	(`integer`) Process ID of the external application to be stopped.

The TERMINATE command returns (`boolean`) `true` if the external application stopped successfully.

Caution

Stopping an external application with the `TERMINATE` command can cause loss of data and system instability. The application cannot save its state or data before it is terminated. You should only stop external applications that were started using the `LAUNCH` command.

Example

```
#Stop the application with process ID 7714:
TERMINATE 7714
```

Edit

Applies to Arbortext IsoDraw 7.0 F000 and later:

The `EDIT` command starts a text editing application, such as Windows Notepad (`notepad.exe`). If the EDIT command line includes a macro name, the macro file containing that macro opens in the editing application.

Syntax

`EDIT` *"app"* [*"macroname"*] [*"opt"*]

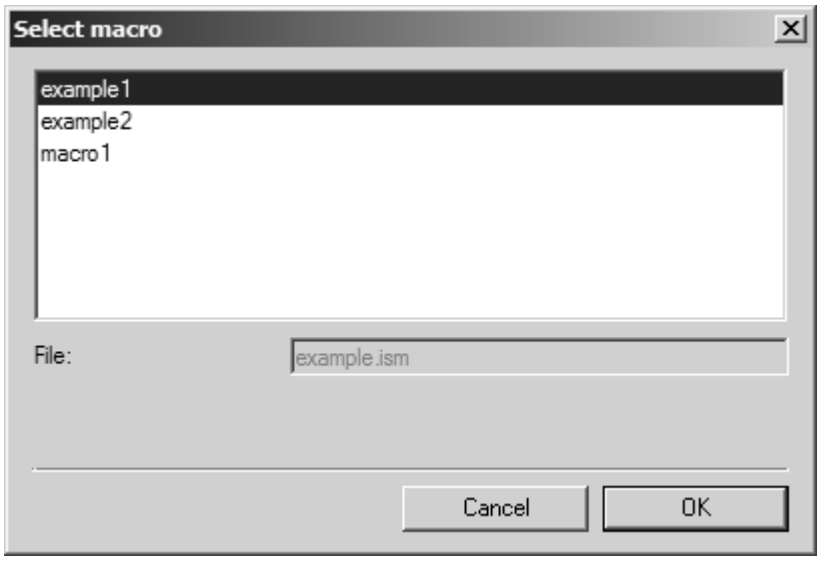
app	(`string`) The path and file name for the text editing application.
macroname	(optional; `string`) The name of the macro to edit. If *macroname* is omitted—or if an empty string (" ") is passed—the **Select macro** dialog box opens so the macro name can be selected manually.
opt	(optional; `string`) A text editor start-up command option, such as search string. This option enables some text editing applications to open a file to a line containing a specific string, such as a command or parameter value. ***Note*** *Not all editing applications support this feature—and those that do have different start-up command line syntax rules. See the editing application documentation for more information.*

The `EDIT` command returns the process ID of the text editing application if it starts successfully, or zero if the application fails to start.

Example

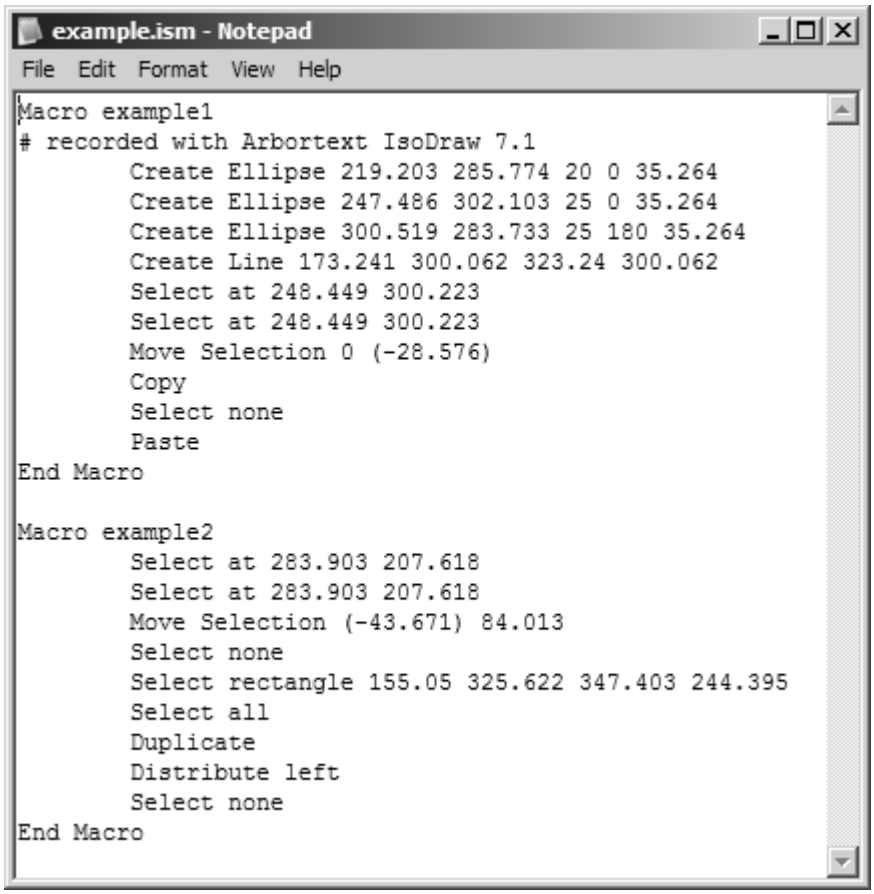
```
# This Edit command opens the Select macro dialog box.
# Manually select the macro to edit:

EDIT "C:\WINNT\system32\notepad.exe" ""
```




```
# This Edit command runs Notepad and opens
# the macro file example.ism which contains
# the macro example1:

EDIT "C:\WINNT\system32\notepad.exe" "example1"
```



Batch

Applies to Arbortext IsoDraw 7.0 F000 and later:

The BATCH command reads a text file line by line and calls a macro with the line's content individually.

Syntax

BATCH *file* [*macro*]

file	(string) The path and file name of the text file to read.
macro	(optional; string) The name of the macro that will process the text file content. If this parameter is omitted, the **Select macro** dialog box opens for manual selection. The BATCH command calls the specified macro for each line of the text file. Each line's string content is passed to the macro as a single parameter value. BATCH only works with macros that expect exactly one parameter. Macros of any other kind will cause BATCH to display an error message.

Example

```
SUBMACRO Delete_Pen( string sPenName )
 DELETE PEN sPenName "$ISO_NOPEN"
END SUBMACRO

MACRO Remove_Invalid_Pens
 BATCH "C:\InvalidPens.txt" "Delete_Pen"
END MACRO
```

Extension

With the EXTENSION command you can execute a plug-in extension with a command line.

Syntax

EXTENSION *"name" "param"*

name	Filename of the extension plug-in as string. Most likely you will use the file name extension ISP here.
param	Command line parameters for the plugin as string.

Example

Please find below a sample with double-quote syntax. The target format is SVG in this case. Note that the source and the destination folders must exist and check the name of the batch tool in the extensions folder.

In this sample the backslash character is used for the line continuation (see Line Continuation on page 31).

```
EXTENSION "batch6_e.isp" "-f22 " + \
"-s""C:\tmp\batch_in"" "+ \
"-d""C:\tmp\batch_out"""
```

Next, we have a sample with mixed quote characters. The target format is CGM . We call a macro ‘rotate90’ on each illustration.

```
EXTENSION "batch6.isp" '-s"D:\TEMP\batch\in"' \
+ ' -d"D:\TEMP\batch\out" -f10 -m"rotata90"'
```

In the next sample we have defined the parameters for the batch tool as string variables. This helps to avoid confusion with the quotes.

```
DEFINE sPlugin AS String
DEFINE sSourceDir AS String
DEFINE sDestDir AS String
DEFINE sParam AS String
DEFINE sFormat AS String

# set the name of the BatchTool
sPlugin = 'batch6_e.isp'

# set the parameter for the source folder
sSource = ' -s"d:\temp\batchin"'

# set the parameter for the destination folder
sDest = ' -d"d:\temp\batchout"'

# set the parameter for the export format
# (10 : CGM)
sFormat = '-f10'

# build the parameter string for the BatchTool
sParam = sFormat + sSource + sDest

# call the BatchTool with the paramter string
EXTENSION sPlugIn sParam
```

If no file format code is specified Arbortext IsoDraw will use its standard format.

```
EXTENSION sPlugIn "-s" + sSourceDir + " -d" \
+ sTargetDir + " - m" + sMacroName
```

For your convenience you will find a list of the format identifiers

```
# Allowed format identifiers are:
#
#  0 Standard IsoDraw format, currently IsoDraw 6
#  1 IsoDraw 4 format
#  2 IsoDraw 3 format
#  3 IsoDraw 2.6 format(not supported On Windows)
#  4 Encapsulated PostScript File format
#  5 Adobe Illustrator 1.1 format
```

```
#  6 Adobe Illustrator 88 format
#  7 Initial Graphics Exchange Standard format
#  8 Drawing Interchange Format
#  9 Hewlett Packard Graphics Language
# 10 Computer Graphics Metafile
# 11 PICTure format
# 12 Tagged Image File Format
# 13 Bitmap format
# 14 PCX format# 15 CALS Raster
# 16 Text Excerpt
# 17 Interleaf format
# 18 Maker Interchange Format
# 19 IsoDraw 5 format
# 20 IsoDraw 5 format, packed
# 21 DWG (AutoCAD format)
# 22 Scalable Vector Graphics
# 23 JPEG format
# 24 Portable Network Graphic format
# 25 Windows Metafile format
# 26 IsoDraw 6 format
# 27 IsoDraw 6 format, packed
```

Increase Text Elements

With the `INCREASE TEXT ELEMENTS` command you can increase the numeric values within selected text elements and callouts by the value specified.

Syntax

`INCREASE TEXT ELEMENTS` *value*

value	Numeric increase as integer.

Example

```
INCREASE TEXT ELEMENTS 5
```

Decrease Text Elements

With the `DECREASE TEXT ELEMENTS` command you can decrease the numeric values within selected text elements and callouts by the value specified.

Syntax

`DECREASE TEXT ELEMENTS` *value*

value	Numeric decrease as integer.

Example

```
DECREASE TEXT ELEMENTS 1
```

15 Interacting with the User

These commands enable the macro to interact with a user.

Message

The `MESSAGE` command prints a message box on the screen.

Syntax

`MESSAGE` *"text"*

text	Message which is shown in the message box.

Example

```
MESSAGE "Ready!"
```

Get

The `GET` command opens a dialog window to get a value from the user.

Syntax

`GET` *input_type "text"*

input_type	Input value type. The parameter may have the following values assigned: `INTEGER` — for an integer value `FLOAT` — for a float value `STRING` — for a string value `BOOLEAN` — for a boolean value
text	String to display to user.

Example

```
username = GET STRING "Please enter your name!"
```

Wait Mouseclick

The `WAIT MOUSECLICK` command stops the execution of a macro until the user hits a mouse button.

Syntax

`WAIT MOUSECLICK`

The `WAIT MOUSECLICK` command returns a MouseEvent.The MouseEvent contains information which can be accessed through the following properties. (It is assumed that myME has already been defined as a MouseEvent for the examples that follow.

Example

```
WAIT MOUSECLICK

DEFINE myME AS MouseEvent
myME = WAIT MOUSECLICK
```

mouseEvent.click

The click property returns which mouse button has been clicked.

0	No button pressed
1	Left mouse button pressed
2	Right mouse button pressed

```
MESSAGE myME.click
```

mouseEvent.ptPix

The ptPix property returns the mouse position as pixel coordinates relative to the current active window. The point of origin is the upper left corner of the active window.

```
MESSAGE myME.ptPix.x

MESSAGE myME.ptPix.y
```

mouseEvent.ptPixGrid

The ptPixgrid property returns the mouse position as pixel coordinates depending on element and grid magnetism relative to the current active window. The point of origin is the upper left corner of the active window.

```
MESSAGE myME.ptPixGrid.x

MESSAGE myME.ptPixGrid.y
```

mouseEvent.ptMM

The ptMM property returns the mouse position of the page coordinates in millimeters. The point of origin is the lower left corner of the active page.

```
MESSAGE myME.ptMM.x

MESSAGE myME.ptMM.y
```

mouseEvent.ptMMGrid

The ptMM property returns the mouse position, in millimeters of the page coordinates depending on element and grid magnetism. The point of origin is the lower left corner of the active page.

```
MESSAGE myME.ptMMGrid.x

MESSAGE myME.ptMMGrid.y
```

mouseEvent.modifiers

The modifiers property returns which special keys are simultaneously pressed with a mouse key.

256	CRTL key
512	SHIFT key
2048	ALT key

```
MESSAGE myME.modifiers
```

```
MACRO MouseDemo

   DEFINE me AS MouseEvent
   DEFINE pt AS Point
   DEFINE ms AS String

   me = WAIT MOUSECLICK
   IF (me.click = 0) THEN
      MESSAGE "Aborted"
   ELSE
      pt = me.ptPix
      ms = " at pixel coordinates"
      ms = ms + pt.x + "," + pt.y
      IF (me.click = 1) THEN
         ms = "Left click" + ms
      Else
         ms = "Right click" + ms
      END IF
      IF (me.modifiers = 512) THEN
         ms = ms + " with Shift-Key"
      END IF
      IF (me.modifiers = 256) THEN
         ms = ms + " with Ctrl-Key"
      END IF
      IF (me.modifiers = 2048) THEN
         ms = ms + " with Alt-Key"
      END IF
      pt = me.ptMMGrid
      Create Text pt.x pt.y ms
   END IF

END MACRO
```

Beep

The `BEEP` command returns an audible beep.

Syntax

```
BEEP
```

V

Functions and Data Types

16 Functions

Functions are handled as expressions and may contain (in brackets) additional expressions to evaluate as a single value.

Trigonometric Functions

All trigonometric functions in Arbortext IsoDraw are degree based.

The following trigonometric functions are defined:

- `sin()`
- `cos()`
- `tan()`
- `arcsin()`
- `arccos()`
- `arctan()`

Example

```
MACRO waveplot
   DEFINE i AS integer
   CREATE LINE 0 100 0 100
   FOR i=1 TO 720
      APPEND LINE SEGMENT i*0.82 sin(i)*100+100
   END FOR
   CREATE LINE 0 200 0 200
   FOR i=1 TO 720
      APPEND LINE SEGMENT i*0.82 cos(i)*100+100
   END FOR
END MACRO
```

Other Mathematical Functions

The following mathematical functions are defined:

`sqrt` (*source*)	square root
`ln` (*source*)	natural logarithm
`exp` (*source*)	exponential function
`abs` (*source*)	absolute value

Example

```
MACRO logplot
   DEFINE i AS integer
   FOR i=1 to 300
      CREATE LINE i 0 i ln(i)*20
   END FOR
END MACRO
```

Random Function

The `rand()` function will return a random number between 0 and the given argument.

The random generator must be initialized with the command randomize. With no parameter given the current time initiates the function; with a parameter given for a repeatable randomized sequence.

Example

```
MACRO modernArt
   DEFINE i AS integer
   RANDOMIZE
   FOR i=0 to 100
      CREATE LINE rand(600) rand(420) rand(600) rand(420)
   END FOR
END MACRO
```

String Functions

There are some special string functions in the Arbortext IsoDraw Macro Language:

`eval` (*source*)	Evaluates a sting as a mathematical function.
`lower` (*source*)	Converts the string to lowercase.
`upper` (*source*)	Converts the string to uppercase.
`len` (*source*)	Returns the length of a string.
`left` (*source* , *count*)	Returns the string's left most characters equal to the count.
`right` (*source*, *count*)	Returns the string's right most characters equal to the count.
`mid` (*source*, *start* ,*count*)	Returns the string's characters from the given start to the right, equal to the count.

Example

```
MACRO String Characters
DEFINE str AS string
str = "123456789"
   MESSAGE "the string is: " + str
   MESSAGE "the first three: " + left(str, 3)
   MESSAGE "the last four: " + right(str, 4)
   MESSAGE "the 4th to 6th: " + mid(str, 4, 3)
END MACRO
```

`find` (*source*, *search*, *start*)	Returns the character position of the search term. Only searches to the right of the start term.
`stripExt` (*source*)	Returns the string without an extension.
`getExt` (*source*)	Returns the string's extension.
`stripFileName` (*source*)	Returns the string without the file name (i.e. returns the path).
`getFileName` (*source*)	Returns the string's file name and extension (i.e. without the path)

Example

```
MACRO Path, Name and Extension Info
   NEW
```

```
   SAVE "C:\Program Files\ITEDO Software\NameSample.iso"
   MESSAGE "the file's full path is: " + $Newline + activeDoc.path
   MESSAGE "the file's full name is: " + $Newline + activeDoc.name
   MESSAGE "the file's name minus extension is: " + $Newline +
   stripExt(activeDoc.name)
   MESSAGE "the file's extension is: " + $Newline + getExt(activeDoc.name)
   MESSAGE "the file's path without its name is: " + $Newline +
   stripFileName(activeDoc.path)
END MACRO
```

`isAlpha` (*source*)	Returns if the string is composed entirely of alpha characters as boolean.
`isDigit` (*source*)	Returns if the string is composed entirely of digits as boolean.
`isLower` (*source*)	Returns if the string is composed entirely of lower case alpha characters as boolean.
`isUpper` (*source*)	Returns if the string is composed entirely of upper case alpha characters as boolean.
`isSpace` (*source*)	Returns if the string is composed entirely of spaces as boolean.
`isAscii` (*source*)	Returns if the string consists entirely of characters found in the first 128 from the ASCII character map as boolean.
`isControl` (*source*)	Returns is the string consists entirely of control characters (such as TAB) as boolean.
`isPrintable` (*source*)	Returns is the string consists entirely of printable characters (i.e. without control characters) as boolean.
`isNumerical` (*source*)	Returns if the string is numerical as boolean. This differs from isDigit (e.g. -99.9 is not true as isDigit, but is true as isNumberical).
`code` (*source*)	Returns the first character's code from the Ascii character map as integer.
`char` (*source*)	Returns the character for the character from the Ascii character map for the given integer value.

Example

```
MACRO stringFunctions
   DEFINE txt AS string
   txt = "Arbortext IsoDraw Macro Language"
   MESSAGE txt
   MESSAGE len(txt)
   MESSAGE upper(txt)
   MESSAGE lower(txt)
END MACRO
```

Time Functions

There are some special functions in the Arbortext IsoDraw Macro Language to determine the time:

`date` ()	Returns the date as string
`time` (*seconds*)	Returns the time as string. Use the optional “seconds” parameter to determine if seconds are given. Enter true to include seconds and false to exclude them.
`ticks` ()	Returns the time in ticks as string. There are 60 ticks per second.

Example

```
MESSAGE "today is " + date()

MESSAGE "and the time is " + time(true)
```

Negation

The `not`() function is for the logical negation. The result is either true or false.

Example

```
IF ( not(a < b) ) THEN
   #if a is greater than b
ELSE
   #if a is smaller than b
END IF
```

Exists

`exists`(*expression*) returns TRUE if the given expression is true within the context of the macro and the defined document, layer or element does exist.

Example

```
# some examples:

IF (exists (activeDoc) = false) THEN
   MESSAGE "No document opened!"

IF (exists (activeDoc.firstSelectedElement) = false) THEN
   MESSAGE "No element selected!"

IF (exists (activeDoc.layers[3]) = false) THEN
   MESSAGE "Less than three layers exist!"

IF (exists (activeDoc.layers["Background"].firstChild) = false) THEN
   MESSAGE "Background layer is empty!"

IF (exists (activeDoc.firstSlectedElement.info) = false) THEN
   MESSAGE "The first selected element has no object info!"

IF (exists (app.pens["Thick"]) = false) THEN
   MESSAGE "Pen ""Thick"" does not exist!"
```

```
MACRO AddCustomPen
   DEFINE penName AS String
   penName = Get String "Name of your new Pen?"
   IF (exists (activeDoc.Pens[penName]) = false) THEN
      Add Pen penName
   ELSE
      MESSAGE "Pen " + penName + " already exists!"
   END IF
END MACRO
```

Return

Parameters can be passed down to macros or subMacros using a `return` function.

Example

```
SUBMACRO Fact( Integer n )
#Calculates the factorial "n!"
DEFINE prod AS integer
DEFINE i AS integer
   prod = 1
   FOR i = 1 TO n
      prod = prod*i
   END FOR
   RETURN prod
END SUBMACRO

MACRO Calculate Factorial
DEFINE n AS integer
DEFINE result AS integer
   n = Get integer "Please enter a small integer value"
   n = abs(n)
   result = Run Fact(n)
MESSAGE n+"! = " + result
END MACRO
```

Call

Applies to Arbortext IsoDraw 7.0 F000 and later.

The `call()` function calls a subMacro as a function. The subMacro has to have a `RETURN` directive to close and pass back a value to the calling macro.

call()

`call(`*macrocall*`)`

macrocall	(`string`) Name of the macro or subMacro to be called. The parameter list can be added in `()` parentheses. The name must correspond to a macro or subMacro in memory.
RETURN value	The type is determined by the calling macro or subMacro.

Example

```
SUBMACRO Foobar( integer p1, integer p2 )
 DEFINE r AS integer
# ...
 RETURN r
END SUBMACRO

MACRO MyMainMacro
 DEFINE result AS integer
 # ...
 # SubMacro-call with "Run" command:
 result = RUN Foobar(a,b)
 # ...
 # The same, using the "call"-Function:
 result = call("Foobar(a,b)")
 # ...
 # The following is not possible with "Run":
 s = GET STRING "Please enter Macro name"
 result = call( s+"(a,b)" )
 # ...
END MACRO
```

17 Simple Data Types

IML provides four simple data types commonly found in many computer languages. These data types can return or set a single number, string, or boolean literal value.

Integers

Integers are whole numbers, for example, 2, 4 and 357. The range of an integer extends from (-2,147,483,648) through 2,147,483,647. Negative values have to be put in parentheses, for example (-1).

Floating-Point Numbers

Floating point numbers represent numeric values with decimal digits. Negative values have to be put in parentheses, for example, (-1.3).

Strings

String literals are delimited either with single(’) or double (") quotes in Arbortext IsoDraw macros:

```
'Mo Szyslak'"Julius Hibbert"
```

Within a string other type quotes can be used without limitation:

```
MESSAGE "hello world: 'part one'"
MESSAGE 'hello world: "part two"'
```

Quotes of the same type have to be doubled for not terminating the string:

```
string1 = "double quote: "" "
string2 = 'single quote: '' '
```

Adding one string to another with the ’+’ operator:

```
DEFINE text AS string
text = "con" + 'cat'
MESSAGE text + "enated"
```

Booleans

A boolean represents a "truth value" - it says whether something is true or not. Truth and falseness determine the outcome of conditional code such as:

```
DEFINE test AS boolean
test = true

IF (test = true) THEN
   #do something
ELSE
   #do something ELSE
END IF
```

18 Complex Data Types

IML provides complex data types which are unique to the language. Unlike simple data types, complex data types contain multiple values which can be returned or set separately using IML property statements.

Point

A point owns two float properties which define the x- and y-coordinates of the point.

```
DEFINE pt_Start AS point
pt_Start.x = (-22.99)
pt_Start.y = 123.123
```

Point3

The point3 type defines a three-dimensional point with its x-, y- and z-coordinates. These properties are float literals.

```
DEFINE pt3_corner AS point3
pt3_corner.x = 332.76
pt3_corner.y = 239
pt3_corner.z = (-45.88)
```

Rectangle

The rectangle type defines a rectangle with its top-left- and bottom-right-corners. The properties defining the single coordinates are float literals.

```
DEFINE rect_Select AS rectangle
rect_Select.top = 252.55
rect_Select.left =  299.297
rect_Select.bottom =  498.348
rect_Select.right =  125.73
```

RGBColor

An RGBColor owns three properties which are integer literals. These properties define the values of **r**ed, **g**reen and **b**lue. In contrast to "normal" integer literals the RGB-integers can only range from 0 to 255.

```
DEFINE RGB_red AS RGBColor
RGB_red.red = 188
RGB_red.green = 0
RGB_red.blue = 139
```

CMYKColor

A CMYKColor owns four properties which are floats literals. These properties define the values of **c**yan, **m**agenta, **y**ellow and **k**ey (or blac**k**). In contrast to "normal" float literals the CMYK-float range is only from 0 to 1.

```
DEFINE CMYK_blue AS CMYKColor
CMYK_blue.cyan = 0.99
CMYK_blue.magenta = 0.7
CMYK_blue.yellow = 0.13
CMYK_blue.black = 0.08
```

ColorSpec

The ColorSpec data type is an either...or data type. This means that the data type could assume three different states. Depending on the state of the data type, it owns two or three properties.

The first and only common to all three different states of the ColorSpec data type is the type property. This property contains the string value "cmykValues", "rgbValues" or "colorRef".

If the value type is "cmykValues" the ColorSpec data type owns a second property called cmyk which is a CMYKColor data type.

```
#defining a colspec with another variable
DEFINE CMYK_blue AS CMYKColor
CMYK_blue.cyan = 0.99
CMYK_blue.magenta = 0.7
CMYK_blue.yellow = 0.13
CMYK_blue.black = 0.08

DEFINE color_blue AS ColorSpec
color_blue.type = "cmykValues"
color_blue.cmyk = CMYK_blue

Add color "NewBlue" color_blue
```

If the value type is "rgbValues" the ColorSpec data type owns a second property called rgb which is an RGBColor data type.

```
#defining a colorSpec directly
DEFINE color_red AS ColorSpec
color_red.type = "rgbValues"
color_red.rgb.red = 188
color_red.rgb.green = 0
color_red.rgb.blue = 29
```

If the value type is "colorRef" the ColorSpec data type owns another two properties. One is a string and called color and one is a float and called tone. The color property contains the name of the color the data type is referencing. Every color defined in Arbortext IsoDraw is allowed. The tone property defines the modification of intensity as a float (0 to 1) of the color defined by the color property.

```
DEFINE grey25 AS ColorSpec
grey25.type = "colorRef"
grey25.color = "$ISO_BLACK"
grey25.tone = 0.25
```

Fill

The fill data type owns four properties. The first one is a string which defines the type of the fill. The allowed values for type are "no_fill", "hatching", "pattern" and "color".

```
DEFINE myFill AS fill
```

```
myFill.type = "pattern"
```

The second property is a ColorSpec called colSpec (see ColorSpec on page 171).

```
myFill.colSpec.rgb.blue = 29
```

The third property defines the hatching used. Values allowed for this property are the names of all existing hatchings as string.

```
myFill.hatching = "Aluminium"
```

The fourth property is used for defining the type of pattern. Allowed values for that property are the names of all existing patterns as string.

```
myFill.pattern = "25% Grey"
```

Mouse Event

The mouse event returns information about the mouse behavior of the user, e.g. which mouse button has been clicked (see Wait Mouseclick on page 154).

```
DEFINE myME AS MouseEvent
myME = WAIT MOUSECLICK
```

VI

Object Data Types

About Object Data Types in IML

IML provides object data types—complex, multi-property data types that can also contain methods that define how the object behaves when invoked.

There are four IML objects: `document`, `element`, `layer`, and `application`. (The `application` object returns and sets user interface and data exchange preferences.)

IML also includes sub data types that, unlike other data types, you do not `DEFINE`. Rather, you use them to set or return the attribute preferences for the current `document` or `application` object.

19 Document Object

The Document object defines a document in Arbortext IsoDraw.

```
DEFINE myDoc AS document
myDoc = new
```

activeDoc

The ActiveDoc object does the following:

- Sets a defined Document object (such as `myDoc` above) as the currently active document
- Enables you to return and set properties of the active document

```
#both examples change the name of the active
#document

DEFINE doc_01 AS document
doc_01 = activeDoc
doc_01.name = "a new document"

#or

activeDoc.name = "a new document"
```

It is assumed that myDoc has already been defined as a document and set as the activeDoc for the following samples.

document.name

Returns and sets the name of the document as string.

```
myDoc.name = "myNewDocument"
```

document.path

Returns and sets the path of the document as string.

```
myDoc.path = "c:\temp\doc_01.iso"
```

document.penCount

Returns the number of the pens as integer. This property is read only.

```
MESSAGE myDoc.penCount
```

document.active_Pen

Returns and sets the name of the active pen as string.

```
myDoc.active_pen = "Thick"
```

document.styleCount

Returns the number of the styles as integer. This property is read only.

```
MESSAGE myDoc.styleCount
```

document.active_style

Returns and sets the name of the active styles as string.

```
myDoc.active_style = "Solid"
```

document.shadowCount

Returns the number of shadows as integer. This property is read only.

```
MESSAGE myDoc.shadowCount
```

document.active_shadow

Returns and sets the name of the active shadow as string.

```
myDoc.active_shadow = "Autom. Short"
```

document.gridCount

Returns the number of grids as integer. This property is read only.

```
MESSAGE myDoc.gridCount
```

document.active_grid

Returns and sets the name of the active grid as string.

```
myDoc.active_grid = "Isometric"
```

document.formatCount

Returns the number of text formats as integer. This property is read only.

```
MESSAGE myDoc.formatCount
```

document.active_textFormat

Returns and sets the name of the active text format as string.

```
myDoc.active_textFormat = "Normal"
```

document.viewportCount

Returns the number of viewports as integer. This property is read only.

```
MESSAGE myDoc.viewportCount
```

document.viewports[ ]

This property gives access to all viewport attributes but it can not be used directly. It is assumed that myDoc has already been defined as a document and set as the activeDoc for the following samples.

document.viewports[].name

Returns and sets the viewport name as string.

```
myDoc.viewports[1].name = "test"
```

document.viewports[].Id

Returns the viewport ID as string. This property is read only.

```
MESSAGE myDoc.viewports[1].Id
```

document.viewports[].Rectangle

Returns and sets the viewport rectangle as rectangle.

```
myDoc.viewports[1].rectangle.bottom = 500
myDoc.viewports[1].rectangle.left = 135
myDoc.viewports[1].rectangle.right = 200
myDoc.viewports[1].rectangle.top = 650
```

document.viewports[].LayerCount

Returns the number of layers in a viewport as integer. This property is read only.

```
MESSAGE myDoc.viewports[1].LayerCount
```

document.viewports[ ].Layers[]

This property gives access to layer attributes within a viewport but it can not be used directly. It is assumed that myDoc has already been defined as a document and set as the activeDoc for the following samples.

document.viewports[].Layer[].Name

Returns and sets the name of a layer within a viewport as string.

```
myDoc.viewports[1].Layers[1].Name = "Lyr1"
```

document.viewports[].Layers[].locked

Returns and sets the locked state of a layer within a viewport as boolean.

```
myDoc.viewports[1].Layers[1].locked = true
```

document.viewports[].Layers[].protected

Returns and sets the protected state of a layer within a viewport as boolean.

document.viewports[].Layers[].active

```
myDoc.viewports[1].Layers[1].protected = true
```

Returns and sets the active state of a layer within a viewport as boolean.

```
myDoc.viewports[1].Layers[1].active = true
```

document.viewports[].Layers[].printable

Returns and sets the printable state of a layer within a viewport as boolean.

```
myDoc.viewports[1].Layers[1].printable = true
```

document.viewports[].Layers[].exportable

Returns and sets the exportable state of a layer within a viewport as boolean.

```
myDoc.viewports[1].Layers[1].exportable = true
```

document.viewports[].Layers[].visible

Returns and sets the visible state of a layer within a viewport as boolean.

```
myDoc.viewports[1].Layers[1].visible = true
```

document.viewports[].Layers[].useColor

Returns and sets the use of color property of a layer within a viewport as boolean.

```
myDoc.viewports[1].Layers[1].useColor = true
```

This property gives access to layer attributes within a viewport but it can not be used directly. It is assumed that myDoc has already been defined as a document and set as the activeDoc for the following samples.

document.calloutCount

Returns the number of callout styles as integer. This property is read only.

```
MESSAGE myDoc.calloutCount
```

document.active_callout

Returns and sets the name of the active callout style as string.

```
myDoc.active_calloutstyle = "Normal"
```

document.layerCount

Returns the number of layers of the document as integer.

```
MESSAGE myDoc.layerCount
```

document.layers[ ]

Gives access to all layer attributes but can not be reached directly (see Layer — The Layer Object on page).

```
myDoc.layers[1].name = "myLayer"
```

document.selectedElements

Returns the number of selected elements of the document as integer.

```
MESSAGE myDoc.selectedElements
```

document.firstSelectedElement

This property returns the first selected element of a selection as element. To get the next element of a selection use element.nextSelectedElement (see element.nextSelectedElement on page 197).

```
MACRO ListElementSelection
   DEFINE el AS Element
   DEFINE m AS String
   el = activeDoc.firstSelectedElement
   IF (exists(el ) = false) THEN
      MESSAGE "Select some Elements, please!"
   Else
      m = "Type(s) of selected Elements: " +  el.type
```

```
      el = el.nextSelectedElement
      WHILE (exists(el ) = true)
         m = m + ", " + el.type
         el = el.nextSelectedElement
      END WHILE
      MESSAGE m + "."
   END If
END MACRO
```

document.selectedParts

Returns the number of selected parts of the document as integer.

```
MESSAGE myDoc.selectedParts
```

document.modified

Returns if the document was modified as boolean. (returns 0 if the document is not modified, 1 if it has been modified)

```
MESSAGE myDoc.modified
```

document.grid

This property gives access to all grid attributes but it can not be used directly. It is assumed that myDoc has already been defined as a document and set as the activeDoc for the following samples.

document.grid.gridSize

Returns and sets the grid size property as float.

```
myDoc.grid.gridSize = 20
```

document.grid.radius

Returns and sets the magnetic radius property as float.

```
myDoc.grid.radius = 5
```

document.grid.elementMagnet

Returns and sets the magnetic elements property as boolean.

```
myDoc.grid.elementMagnet = true
```

document.grid.gridMagnet

Returns and sets the magnetic grid points property as boolean.

```
myDoc.grid.gridMagnet = true
```

document.grid.showGrid

Returns and sets the show grid property as boolean.

```
myDoc.grid.showGrid = true
```

document.grid.showDim

Returns and sets the show dimensions property as boolean.

```
myDoc.grid.showDim = true
```

document.grid.constrain

Returns and sets the align to grid property as boolean.

```
myDoc.grid.constrain = true
```

document.grid.gridInFront

Returns and sets the grid in front property as boolean.

```
myDoc.grid.gridInFront = true
```

document.grid.noFShortInIso

Returns and sets the no isometric foreshortening property as boolean.

```
myDoc.grid.noFShortInIso = true
```

document.grid.gridColor

Returns and sets the color property as RGBColor.

```
myDoc.grid.gridColor.red = 50
myDoc.grid.gridColor.green = 150
myDoc.grid.gridColor.blue = 50
```

document.grid.firstSelColor

Returns and sets the first color property as RGBColor.

```
myDoc.grid.firstSelColor.red = 10
myDoc.grid.firstSelColor.green = 160
myDoc.grid.firstSelColor.blue = 140
```

document.grid.secondSelColor

Returns and sets the second color property as RGBColor.

```
myDoc.grid.secondSelColor.red = 75
myDoc.grid.secondSelColor.green = 30
myDoc.grid.secondSelColor.blue = 85
```

document.window

This property gives access to all window attributes but it can not be used directly. It is assumed that myDoc has already been defined as a document and set as the activeDoc for the following samples.

document.window.pageX

Returns and sets the window x-cords property as float.

```
myDoc.window.pageX = 200
```

document.window.pageY

Returns and sets the window y-cords property as float.

```
myDoc.window.pageY = 300
```

document.window.overlapX

Returns and sets the print overlap x-cords property as float.

```
myDoc.window.overlapX = 10
```

document.window.overlapY

Returns and sets the print overlap y-cords property as float.

```
myDoc.window.overlapY = 10
```

document.window.scale

This command is included for compatibility reasons only. It is not recommended that you use this command for new development of macros as it may be discontinued in the future. All features from this command have been incorporated into the new command `ZOOM` (see Zoom on page 87).

Returns and sets the window view size (zoom) property as float.

```
myDoc.window.scale = 10
```

document.window.dimScale

Returns and sets the dimension scaling property as float.

```
myDoc.window.dimScale = 3
```

document.window.preview

Returns and sets the objects preview property as integer. The allowed values are:

0	“no preview”
1	“a preview”
2	“visible hotspots”
4	“visible objects”

Add values to combine properties.

```
myDoc.window.preview = 6
```

document.window.is3D

Returns the 3D property as boolean. This property is read only.

```
MESSAGE myDoc.window.is3D
```

document.window.curSystem

Returns and sets the current unit system property as string. Allowed values are "mm", "in" and "pt".

```
myDoc.window.curSystem = "mm"
```

document.shadow

This property gives access to all shadow attributes but it can not be used directly. It is assumed that myDoc has already been defined as a document and set as the activeDoc for the following samples.

document.shadow.shadowWidth

Returns and sets the shadow width property as float.

```
myDoc.shadow.shadowWidth = 5
```

document.shadow.shadowFactor

Returns and sets the shadow factor property as float.

```
myDoc.shadow.shadowFactor = 2
```

document.thread

This property gives access to all thread attributes but it can not be used directly. It is assumed that myDoc has already been defined as a document and set as the activeDoc for the following samples.

document.thread.inner.upTo1

Returns and sets the first "inner thread up to" property as float.

```
myDoc.thread.inner.upTo1 = 20
```

document.thread.inner.upTo2

Returns and sets the second "inner thread up to" property as float.

```
myDoc.thread.inner.upTo2 = 40
```

document.thread.inner.size1

Returns and sets the first inner thread distance property as float.

```
myDoc.thread.inner.size1 = 2.5
```

document.thread.inner.size2

Returns and sets the second inner thread distance property as float.

```
myDoc.thread.inner.size2 = 4
```

document.thread.inner.size3

Returns and sets the third inner thread distance property as float.

```
myDoc.thread.inner.size3 = 5.5
```

document.thread.outer.upTo1

Returns and sets the first "outer thread up to" property as float.

```
myDoc.thread.outer.upTo1 = 20
```

document.thread.outer.upTo2

Returns and sets the second "outer thread up to" property as float.

```
myDoc.thread.outer.upTo2 = 40
```

document.thread.outer.size1

Returns and sets the first outer thread distance property as float.

```
myDoc.thread.outer.size1 = 2.5
```

document.thread.outer.size2

Returns and sets the second outer thread distance property as float.

```
myDoc.thread.outer.size2 = 4
```

document.thread.outer.size3

Returns and sets the third outer thread distance property as float.

```
myDoc.thread.outer.size3 = 5.5
```

document.thickthin

This property gives access to all thick/thin attributes but it can not be used directly. It is assumed that myDoc has already been defined as a document and set as the activeDoc for the following samples.

document.thickthin.useThickThin

Returns and sets the "use thick thin technique" property as boolean.

```
myDoc.thickthin.useThickThin = true
```

document.thickthin.thickPen

Returns and sets the thick pen property as string. The name of every exiting pen is an allowed value.

```
myDoc.thickthin.thickPen = "Thick"
```

document.thickthin.thinPen

Returns and sets the thin pen property as string. The name of every exiting pen is an allowed value.

```
myDoc.thickthin.thinPen = "Thin"
```

document.thickthin.useEllipsePens

Returns and sets the "specify pens for ellipses and threads" property as boolean.

```
myDoc.thickthin.useEllipsePens = true
```

document.thickthin.thick1

Returns and sets the first thick pen property as string. The name of every exiting pen is an allowed value.

```
myDoc.thickthin.thick1 = "Thick"
```

document.thickthin.thin1

Returns and sets the first thin pen property as string. The name of every exiting pen is an allowed value.

```
myDoc.thickthin.thin1 = "Thin"
```

document.thickthin.thick2

Returns and sets the second thick pen property as string. The name of every exiting pen is an allowed value.

```
myDoc.thickthin.thick2 = "Thick"
```

document.thickthin.thin2

Returns and sets the second thin pen property as string. The name of every exiting pen is an allowed value.

```
myDoc.thickthin.thin2 = "Thin"
```

document.thickthin.thick3

Returns and sets the third thick pen property as string. The name of every exiting pen is an allowed value.

```
myDoc.thickthin.thick3 = "Thick"
```

document.thickthin.thin3

Returns and sets the third thin pen property as string. The name of every exiting pen is an allowed value.

```
myDoc.thickthin.thin3 = "Thin"
```

document.thickthin.upto1

Returns and sets the first up to property as float.

```
myDoc.thickthin.upto1 = 20
```

document.thickthin.upto2

Returns and sets the second up to property as float.

```
myDoc.thickthin.upto2 = 40
```

document.background

This property gives access to all background attributes but it can not be used directly. It is assumed that myDoc has already been defined as a document and set as the activeDoc for the following samples.

document.background.inColor

Returns and sets the “in color” property as boolean.

```
myDoc.background.inColor = true
```

document.background.intensity

Returns and sets the intensity property as float. The property range is from 0 to 100.

```
myDoc.background.intensity = 50
```

document.background.color

Returns and sets the screen color property as RGBColor.

```
myDoc.background.color = "{RGB 50 80 120}"
```

document.lineOptions

This property gives access to all line option attributes but it can not be used directly. It is assumed that myDoc has already been defined as a document and set as the activeDoc for the following samples.

document.lineOptions.lineCap

Returns and sets the line cap ends property as integer. Allowed values are 0 for “flat”, 1 for “round” and 2 for “square”.

```
myDoc.lineOptions.lineCap = 1
```

document.lineOptions.lineJoin

Returns and sets the line option corners property as integer. Allowed values are 0 for “mitered”, 1 for “rounded” and 2 for “bevel”.

```
myDoc.lineOptions.lineJoin = 2
```

document.lineOptions.miterLimit

Returns and sets the miter limit property as integer.

```
myDoc.lineOptions.miterLimit = 4
```

document.lineOptions.overPrint

Returns and sets the overprint property as integer. Allowed values are 0 for “none”, 1 for “stroke”, 2 for “fill” and 3 for “stroke and fill”.

```
myDoc.lineOptions.overPrint = 2
```

document.simpleEllipsePrinting

This property sets and returns the simple ellipse printing as boolean.

```
myDoc.simpleEllipsePrinting = true
```

document.colorCount

Returns the number of colors as integer. This property is read only.

```
MESSAGE myDoc.colorCount
```

document.hatchingCount

Returns the number of hatchings as integer. This property is read only.

```
MESSAGE myDoc.hatchingCount
```

document.patternCount

Returns the number of patterns as integer. This property is read only.

```
MESSAGE myDoc.patternCount
```

document.Objects[*object_ID*]

Applies to Arbortext IsoDraw 7.0 F000 and later:

Enables access to an object within the document by object ID string. This property is read only.

object_ID	(String) Object ID of the object to access.

```
MESSAGE myDoc.patternCount
```

Example

```
MACRO ObjectIndexDemo
 DEFINE sID AS string
 DEFINE sName AS string
 sID = "AUTOID_3833"
 IF ( exists(activeDoc.Objects[sID]) )
  sName = activeDoc.Objects[sID].info.NAME
  MESSAGE "Name: " + sName
 ELSE
  MESSAGE "Not Found!"
 END IF
END MACRO
```

document.lockedHidden

Applies to Arbortext IsoDraw 7.0 F000 and later:

Returns or sets the show/hide state of locked elements in a document as boolean. This property corresponds to the **Hide** command on the **Element** menu.

true	Hide locked elements.
false	Show locked elements.

- If a macro changes the state of `document.lockedHidden`, the changed show/hide state of locked elements is not saved in the document.
- Recording a macro that selects or clears **Hide** on the **Element** menu adds a `document.lockedHidden = {true|false}` statement to the macro.

Note

*Like **Hide**, this document property setting only affects locked elements on unlocked layers. Locked elements on locked layers are not affected.*

```
document.lockedHidden = true
```

Example

```
MACRO ObjectIndexDemo
 DEFINE sID AS string
 DEFINE sName AS string
 sID = "AUTOID_3833"
 IF ( exists(activeDoc.Objects[sID]) )
  sName = activeDoc.Objects[sID].info.NAME
  MESSAGE "Name: " + sName
 ELSE
  MESSAGE "Not Found!"
 END IF
END MACRO
```

document.lock3Dinteraction

Applies to Arbortext IsoDraw 7.0 F000 and later:

Returns or sets the show/hide state of the 3D object **Rotate** tool in Arbortext IsoView as boolean. This property corresponds to the **Lock 3D interaction** command on the **Objects** menu. It only applies when the current document contains 3D objects and the user (or macro) selects **Show in IsoView** on the **Objects** menu. (See Show in IsoView on page 78.)

true	Hides the 3D object **Rotate** tool in Arbortext IsoView when Arbortext IsoView is opened with the **Show in IsoView** command. 3D objects cannot be rotated in Arbortext IsoView.
false	Shows the 3D object **Rotate** tool in Arbortext IsoView when Arbortext IsoView is opened with the **Show in IsoView** command. 3D objects can be rotated in Arbortext IsoView.

- The state of `document.lock3Dinteraction` can be changed by a macro.
- Recording a macro that selects or clears **Lock 3D interaction** on the **Objects** menu adds a `document.lock3Dinteraction = {true|false}` statement to the macro. (The **Lock 3D interaction** command is only available when there are 3D objects in the current document.)

```
document.lock3Dinteraction = true
```

20

Element Object

An element is something you have created in Arbortext IsoDraw, for example an ellipse or a line.

```
DEFINE myElem AS element
myElem = CREATE LINE 10 10 100 10
```

All create commands (ellipse, line, rectangle ...) return a reference on the created element. The properties of this element object can be queried and set.

```
MACRO Change Segment Pens

DEFINE myElem AS element
myElem = CREATE LINE 10 10 100 100
APPEND LINE SEGMENT 100 10
APPEND LINE SEGMENT 10 10

myElem.line.segments[1].pen = "Thick"
myElem.line.segments[2].pen = "Medium"
myElem.line.segments[3].pen = "Thin"

END MACRO
```

It is assumed that myElem has already been defined as an element for the following samples.

element.element_id

Returns and sets the element's internal id as integer.

```
MESSAGE myElem.element_id
```

element.type

Returns the element type as string. Allowed values are:

```
"Line"
"Rectangle"
"Polygon"
"Marker"
"Ellipse"
"Inner_thread"
"Outer_thread"
"Bezier"
"Text"
"Placed_file"
"Image"
"Callout"
```

This property is read only.

```
MESSAGE myElem.type
```

element.locked

Returns and sets the locked state of the element as integer. Allowed values are 0 for "unlocked" and 1 for "locked".

```
myElem.locked = 0
```

element.mask

Returns the masked state of the element as boolean. This property is read only.

```
MESSAGE myElem.mask
```

element.box

Returns the element box as rectangle. This property is read only.

```
MESSAGE myElem.box.top
```

```
MESSAGE myElem.box.left
```

```
MESSAGE myElem.box.bottom
```

```
MESSAGE myElem.box.right
```

element.firstChild

This will return the first element located within a group. The returned data type is element. This property is read only.

element.lastChild

This will return the last element located within a group. The returned data type is element. This property is read only.

element.previousSibling

This will return the previous element located within a group or on a layer. The returned data type is element. This property is read only.

element.nextSibling

This will return the next element located within a group or on a layer. The returned data type is element. This property is read only.

```
MACRO ElementChildAndSiblingDefine i AS integer
   DEFINE n AS integer
   DEFINE k AS integer
   DEFINE m AS integer
   DEFINE ele1 AS element
   DEFINE ele2 AS element
   DEFINE ele3 AS element
   DEFINE lay AS layer

   FOR i = activeDoc.layerCount to 1 step -1
      MESSAGE "processing layer " + activeDoc.layers[i].name
      ele1 = activeDoc.layers[i].firstChild
      n = 1
      WHILE (exists(ele1) = true)
         MESSAGE "The " + n + " element of this layer is a " + ele1.type
         n = n + 1
         ele2 = ele1.firstChild
         k = 1
         WHILE (exists(ele2) = true)
            MESSAGE "The " + k + " element of this group is a " + ele2.type
            ele3 = ele2.firstChild
            m = 1
            WHILE (exists(ele3) = true)
               MESSAGE "The " + m + " element of this subgroup is a " + ele3.type
               ele3 = ele3.nextSibling
               m = m + 1
```

```
            END WHILE
            ele2 = ele2.nextSibling
            k = k + 1
         END WHILE
         ele1 = ele1.nextSibling
      END WHILE
   END FOR
END MACRO
```

element.parent

Returns the parent of an element as element.

element.layer

Returns the layer which the element is located on as layer. This property is read only.

```
MACRO getLayerName
   DEFINE el AS element
   select all
   el = activeDoc.firstSelectedElement
   MESSAGE el.layer.name
END MACRO
```

element.selected

Returns the selected state of the element as integer. This property is read only. 0 means “not selected” and 1 means “selected”.

```
MESSAGE myElem.selected
```

element.nextSelectedElement

This property returns the next selected element of a selection as element. To get the first element of a selection see document.firstSelectedElement on page 181.

```
MACRO MakeHotspots
   DEFINE el AS Element
   Select IF Type is equal to "Text"
   el = activeDoc.firstSelectedElement
   WHILE (Exists (el) = true )
      Create Object_Info el
      el.info.hotspot = "region"
      el = el.nextSelectedElement
   END while
END MACRO
```

element.lineCap

Returns and sets the ends property of the element as integer. Allowed values are 0 for “flat”, 1 for “round” and 2 for “square”.

```
myElem.lineCap = 1
```

element.lineJoin

Returns and sets the corners property of the element as integer. Allowed values are 0 for “mitered”, 1 for “rounded” and 2 for “bevel”.

```
myElem.lineJoin = 2
```

element.miterLimit

Returns and sets the miter limit property of the element as integer.

```
myElem.miterLimit = 4
```

element.overPrint

Returns and sets the overprint property of the element as integer. Allowed values are 0 for “none”, 1 for “stroke”, 2 for “fill” and 3 for “stroke and fill”.

```
myElem.overPrint = 2
```

element.segmentCount

Returns the number of the segments of the element as integer. This property is read only.

```
MESSAGE myElem.segmentcount
```

element.fill

Returns and sets the filling of the element as Fill.

```
myElem.fill.colSpec.rgb.red = 200
myElem.fill.colSpec.rgb.green = 60
myElem.fill.colSpec.rgb.blue = 120
```

element.group.childCount

Returns the number of elements within a group as integer. This property is read only.

```
MESSAGE myElem.group.childCount
```

element.document

With element.document you can modify the properties of a placed element.

```
DEFINE myElem AS elementmy
Elem = place

MESSAGE myElem.document.destinationRect.top
```

element.document.startPoint

Note

This property is deprecated and is only supported for backward compatibility.

Returns and sets the start point parameter as point.

```
myElem.document.startPoint.x = 58.84
```

element.document.destinationRect

Returns the destination rectangle parameter as rect. This property is read only.

```
MESSAGE myElem.document.destinationRect.bottom
```

element.document.file_name

Returns the file name of the placed document as string. This property is read only.

```
MESSAGE myElem.document.file_name
```

element.document.path

Returns the path of the placed document as string. This property is read only.

```
MESSAGE myElem.document.path
```

element.document.update_mode

Returns and sets the update mode of the placed document as integer. Allowed values are 1 for “auto”, 2 for “notify” and 3 for “none”.

```
myElem.document.update_mode = 1
```

element.document.include_file

Note

This property is deprecated and is only supported for backward compatibility.

Returns and sets if the placed file should be included in the document as integer. Allowed values are 0 for "do not include" and 2 for "include file"

```
myElem.document.include_file = 0
```

element.info

With element.info you can modify the properties of the object info of the element.

```
MESSAGE myElem.info.id
```

element.info.type

Returns and sets the type of the object info as string.

```
MESSAGE myElem.info.type
```

element.info.id

Returns and sets the object id as string.

```
myElem.info.id = "AQ01"
```

element.info.name

Returns and sets the object name as string.

```
myElem.info.name = "TI"
```

element.info.tip

Returns and sets the object tip as string.

```
myElem.info.tip = "hi"
```

element.info.hotspot

Returns and sets the object hotspot as string. Allowed string values are “none”, “line” and “region”

```
myElem.info.hotspot = "region"
```

element.info.attrCount

Returns the number of attributes of the object as integer. This property is read only.

```
MESSAGE myElem.info.attrCount
```

element.info.attributes[ ]

This property gives access to object attributes but can not be used directly. It is assumed that myElem has already been defined as an element for the following samples

element.info.attributes[ ].type

Returns and sets the type of one object attribute as string. Allowed string values are "String", "Float", "Integer" and "Link".

```
myElem.info.attributes[1].type
= "Integer"
```

element.info.attributes[ ].name

Returns and sets the name of one object attribute as string.

```
myElem.info.attributes[1].name
= "IsoDraw"
```

element.info.attributes[ ].value

Returns and sets the value of one object attribute. Depending on what type the attribute is defined the values of this property could be integer, float or string (see element.info.attributes[ ] on page 200).

```
myElem.info.attributes[1].value = "Integer"
```

element.info.view_context

This property gives access to all view context attributes but can not be used directly. It is assumed that myElem has already been defined as an element for the following samples

element.info.view_context.type

Returns and sets the type of the view context as string. Possible values are "none", "extent" and "size".

```
myElem.info.view_context.type = "none"
```

element.info.view_context.rectangle

If the view context “size” is used, this property must also be used to return and set the size of the view context as rectangle.

```
myElem.info.view_context.rectangle.top = 500.89
myElem.info.view_context.rectangle.left = 200
myElem.info.view_context.rectangle.bottom = 300.54
myElem.info.view_context.rectangle.right = 350.34
```

element.line

This property gives access to all line attributes but it can not be used directly. It is assumed that myElem has already been defined as an element for the following samples

element.line.startPoint

Returns and sets the start point of the line as point.

```
myElem.line.startPoint.x = 107.55
myElem.line.startPoint.y = 157.39
```

element.line.segments[ ].selected

Returns the selected state of one line segment as integer. This property is read only. 0 means “not selected” and 1 means “selected”.

```
MESSAGE myElem.line.segments[1].selected
```

element.line.segments[ ].endPoint

Returns and sets the end point of one line segment as point.

```
myElem.line.segments[1].endPoint.x = 188.66
myElem.line.segments[1].endPoint.y = 256.66
```

element.line.segments[ ].pen

Returns and sets the pen of one line segment as string. Any existing pen is allowed.

```
myElem.line.segments[1].pen = "Thick"
```

element.line.segments[ ].style

Returns and sets the style of one line segment as string. Any existing style is allowed.

```
myElem.line.segments[1].style = "Solid"
```

element.line.segments[ ].shadow

Returns and sets the shadow of one line segment as string. Any existing shadow is allowed.

```
myElem.line.segments[1].shadow = "Autom. Long"
```

element.ellipse

This property gives access to all ellipse attributes but it can not be used directly. It is assumed that myElem has already been defined as an element for the following samples

element.ellipse.centerPoint

Returns and sets the center point of the ellipse as point.

```
myElem.ellipse.centerPoint.x = 58.76
myElem.ellipse.centerPoint.y = 12.45
```

element.ellipse.angle

Returns and sets the angle of the ellipse as float.

```
myElem.ellipse.angle = 45
```

element.ellipse.value

Returns and sets the value of the ellipse as float.

```
myElem.ellipse.value = 90
```

element.ellipse.radius

Returns and sets the radius of the ellipse as float.

```
myElem.ellipse.radius = 500
```

element.ellipse.segments[ ].selected

Returns the selected state of one ellipse segment as integer. This property is read only. 0 means “not selected” and 1 means “selected”.

```
MESSAGE myElem.ellipse.segments[1].selected
```

element.ellipse.segments[ ].startAngle

Returns and sets the start angle of one ellipse segment as float.

```
myElem.ellipse.segments[1].startAngle = 35.7
```

element.ellipse.segments[ ].endAngle

Returns and sets the end angle of one ellipse segment as float.

```
myElem.ellipse.segments[1].endAngle = 35.7
```

element.ellipse.segments[ ].pen

Returns and sets the pen of one ellipse segment as string. Any existing pen is allowed.

```
myElem.ellipse.segments[1].pen = "Thick"
```

element.ellipse.segments[ ].style

Returns and sets the style of the ellipse segment as string. Any existing style is allowed.

```
myElem.ellipse.segments[1].style = "Solid"
```

element.ellipse.segments[ ].shadow

Returns and sets the shadow of one ellipse segment as string. Any existing shadow is allowed.

```
myElem.ellipse.segments[1].shadow = "Autom. Long"
```

element.innerthread

This property gives access to all inner threads attributes but it can not be used directly. It is assumed that myElem has already been defined as an element for the following samples

element.innerthread.centerPoint

Returns and sets the center point of the inner thread as point.

```
myElem.innerthread.centerPoint.x = 134.56
myElem.innerthread.centerPoint.y = 12.45
```

element.innerthread.angle

Returns and sets the angle of the inner thread as float.

```
myElem.innerthread.angle = 45
```

element.innerthread.value

Returns and sets the value of the inner thread as float.

```
myElem.innerthread.value = 90
```

element.innerthread.radius

Returns and sets the radius of the inner thread as float.

```
myElem.innerthread.radius = 500
```

element.innerthread.segments[ ].selected

Returns the selected state of one inner thread segment as integer. This property is read only. 0 means “not selected” and 1 means “selected”.

```
MESSAGE myElem.innerthread.segments[1].selected
```

element.innerthread.segments[ ].startAngle

Returns and sets the start angle of one inner thread segment as float.

```
myElem.innerthread.segments[1].startAngle = 35.7
```

element.innerthread.segments[ ].endAngle

Returns and sets the end angle of one inner thread segment as float.

```
myElem.innerthread.segments[1].endAngle = 35.7
```

element.innerthread.segments[ ].pen

Returns and sets the pen of one inner thread segment as string. Any existing pen is allowed.

```
myElem.innerthread.segments[1].pen = "Thick"
```

element.innerthread.segments[ ].style

Returns and sets the style of one inner thread segment as string. Any existing style is allowed.

```
myElem.innerthread.segments[1].style = "Solid"
```

element.innerthread.segments[ ].shadow

Returns and sets the shadow of one inner thread segment as string. Any existing shadow is allowed.

```
myElem.innerthread.segments[1].shadow = "Autom. Long"
```

element.outerthread

This property gives access to all outer threads attributes but it can not be used directly. It is assumed that myElem has already been defined as an element for the following samples

element.outerthread.centerPoint

Returns and sets the center point of the outer thread as point.

```
myElem.outerthread.centerPoint.x = 150.75
myElem.outerthread.centerPoint.y = 180
```

element.outerthread.angle

Returns and sets the angle of the outer thread as float.

```
myElem.outerthread.angle = 45
```

element.outerthread.value

Returns and sets the value of the outer thread as float.

```
myElem.outerthread.value = 90
```

element.outerthread.radius

Returns and sets the radius of the outer thread as float.

```
myElem.outerthread.radius = 500
```

element.outerthread.segments[ ].selected

Returns the selected state of one outer thread segment as integer. This property is read only. 0 means “not selected” and 1 means “selected”.

```
MESSAGE myElem.outerthread.segments[1].selected
```

element.outerthread.segments[ ].startAngle

Returns and sets the start angle of one outer thread segment as float.

```
myElem.outerthread.segments[1].startAngle = 35.7
```

element.outerthread.segments[ ].endAngle

Returns and sets the end angle of one outer thread segment as float.

```
myElem.outerthread.segments[1].endAngle = 35.7
```

element.outerthread.segments[ ].pen

Returns and sets the pen of one outer thread segment as string. Any existing pen is allowed.

```
myElem.outerthread.segments[1].pen = "Thick"
```

element.outerthread.segments[ ].style

Returns and sets the style of one outer thread segment as string. Any existing style is allowed.

```
myElem.outerthread.segments[1].style = "Solid"
```

element.outerthread.segments[ ].shadow

Returns and sets the shadow of one outer thread segment as string. Any existing shadow is allowed.

```
myElem.outerthread.segments[1].shadow = "Autom. Long"
```

element.callout

This property gives access to all callout attributes but it can not be used directly. It is assumed that myElem has already been defined as an element for the following samples

element.callout.style_name

Returns the name of the callout style as string. This property is read only.

```
MESSAGE myElem.callout.style_name
```

element.callout.line_pen

Returns and sets the type of the callout line pen as string. Allowed values are all existing pen names and "$CS_DEFAULT" for using the value of the Normal callout style.

```
MESSAGE myElem.callout.line_pen

myElem.callout.line_pen = "$CS_DEFAULT"
```

element.callout.line_style

Returns and sets the type of the callout line style as string. Allowed values are all existing style names and "$CS_DEFAULT" for using the value of the Normal callout style.

```
myElem.callout.line_style = "$CS_DEFAULT"
```

element.callout.line_shadow

Returns and sets the type of the callout line shadow as string. Allowed values are all existing shadow names and "$CS_DEFAULT" for using the value of the Normal callout style.

```
myElem.callout.line_shadow = "$CS_DEFAULT"
```

element.callout.shape_type

Returns and sets the callout shape type as integer. Use $CS_DEFAULT for the shape type value of the Normal callout style. Allowed values are:

0	"None"
1	"Circle"

2	"Triangle Up"
3	"Triangle Down"
4	"Rectangle"
5	"Pentagon"
6	"Hexagon"

```
myElem.callout.shape_type = 2
```

```
myElem.callout.shape_type = $CS_DEFAULT
```

element.callout.shape_pen

Returns and sets the callout shape pen as string. Allowed values are all existing pen names and "$CS_DEFAULT" for using the value of the Normal callout style.

```
myElem.callout.shape_pen = "$CS_DEFAULT"
```

element.callout.shape_style

Returns and sets the callout shape type as string. Allowed values are all existing line style names and "$CS_DEFAULT" for using the value of the Normal callout style.

```
myElem.callout.shape_style = "Solid"
```

element.callout.shape_shadow

Returns and sets the callout shape shadow as string. Allowed values are all existing shadow names and "$CS_DEFAULT" for using the value of the Normal callout style.

```
myElem.callout.shape_shadow = "no shadow"
```

element.callout.fill

Returns and sets the fill of the callout shape.

```
myElem.callout.fill.colSpec.type = "ColorRef"
```

```
myElem.callout.fill.colSpec.color = "Black"
```

```
myElem.callout.fill.colSpec.tone = 0.50
```

element.callout.shape_width

Returns and sets the value of the callout shape width as float. Use $CS_DEFAULT for the shape type value of the Normal callout style.

```
myElem.callout.shape_width = 2.5
```

element.callout.shape_height

Returns and sets the value of the callout shape height as float. Use $CS_DEFAULT for the shape type value of the Normal callout style.

```
myElem.callout.shape_height = 1.5
```

element.callout.text_update

Returns and sets the status of the text update width as string. The allowed values are "none" and "auto" or "$CS_DEFAULT" for the value of the Normal callout style.

```
myElem.callout.text_update = "auto"
```

element.callout.text_position

Returns and sets the type of the text alignment as string. The allowed values are "aligned" and "centered" or "$CS_DEFAULT" for the value of the Normal callout style.

```
myElem.callout.text_position = "centered"
```

element.callout.text_prefix

Returns and sets the text prefix as string. Use "$CS_DEFAULT" for the value of the Normal callout style.

```
myElem.callout.text_prefix = "partNo:"
```

element.callout.text_postfix

Returns and sets the text postfix as string. Use "$CS_DEFAULT" for the value of the Normal callout style.

```
myElem.callout.text_postfix = "_66765"
```

element.callout.string

Returns and sets the text content of a callout—without prefix and postfix— as string. This callout element property can be changed by a macro. Use "$CS_DEFAULT" for the value of the Normal callout style.

```
myElem.callout.string = "DK95002"
```

element.callout.text_gap

Returns and sets the text gap as float. Use $CS_DEFAULT for the value of the Normal callout style.

```
myElem.callout.text_gap = 1.5
```

element.callout.text_font

Returns and sets the text font of the callout as string. Any installed font can be used. Use "$CS_DEFAULT" for the value of the Normal callout style.

```
myElem.callout.text_font = "Arial"
```

element.callout.text_face

Returns and sets the text face of the callout as string. Allowed values are "normal", "bold", "italic", "bolditalic" and "$CS_DEFAULT" for the value of the Normal callout style.

```
myElem.callout.text_face = "bold"
```

element.callout.text_size

Returns and sets the text size of the callout as float. Use $CS_DEFAULT for the value of the Normal callout style.

```
myElem.callout.text_size = 24.5
```

element.callout.text_strokecolor

Returns and sets the text stroke color of the callout as ColorSpec (see ColorSpec on page 171).

```
myElem.callout.text_strokecolor.type = "rgbValues"
```

element.callout.text_stroke

Returns and sets the text stroke of the callout as float.

```
myElem.callout.text_stroke = 2.2
```

element.callout.text_fillcolor

Returns and sets the color of the fill of the text of the callout as ColorSpec (see ColorSpec on page 171).

```
myElem.callout.text_fillcolor.type = "rgbValues"
myElem.callout.text_fillcolor.rgb.red = 168
myElem.callout.text_fillcolor.rgb.green = 201
myElem.callout.text_fillcolor.rgb.blue = 98
```

element.callout.text_scheme

Returns the text scheme of the callout as string. This property is read only.

```
MESSAGE myElem.callout.text_scheme
```

element.callout.text_hotspot_flag

Returns and sets if the hotspot flag is set as integer. Allowed values are 0 for "No Hotspot", 1 for "Is Hotspot" and 256 for the value of the callout style.

```
myElem.callout.text_hotspot_flag = 256
```

element.rect

This property gives access to all rectangle attributes but it can not be used directly. It is assumed that myElem has already been defined as an element for the following samples

element.rect.startPoint

Returns and sets the start point of the rectangle as point.

```
myElem.rect.startPoint.x = 107.55
myElem.rect.startPoint.y = 200.89
```

element.rect.segments[ ].selected

Returns the selected state of one rectangle segment as integer. This property is read only. 0 means “not selected” and 1 means “selected”.

```
MESSAGE myElem.rect.segments[1].selected
```

element.rect.segments[ ].endPoint

Returns and sets the end point of one rectangle segment as point.

```
myElem.rect.segments[1].endPoint.x = 388.66
myElem.rect.segments[1].endPoint.y = 388.66
```

element.rect.segments[ ].pen

Returns and sets the pen of one rectangle segment as string. Any existing pen is allowed.

```
myElem.rect.segments[1].pen = "Thick"
```

element.rect.segments[ ].style

Returns and sets the style of one rectangle segment as string. Any existing style is allowed.

```
myElem.rect.segments[1].style = "Solid"
```

element.rect.segments[ ].shadow

Returns and sets the shadow of one rectangle segment as string. Any existing shadow is allowed.

```
myElem.rect.segments[1].shadow = "Autom. Long"
```

element.polygon

This property gives access to all polygon attributes but it can not be used directly. It is assumed that myElem has already been defined as an element for the following samples

element.polygon.startPoint

Returns and sets the start point of the polygon as point.

```
myElem.polygon.startPoint.x = 107.55
myElem.polygon.startPoint.y = 107.55
```

element.polygon.segments[ ].selected

Returns the selected state of one polygon segment as integer. This property is read only. 0 means “not selected” and 1 means “selected”.

```
MESSAGE myElem.polygon.segments[1].selected
```

element.polygon.segments[ ].endPoint

Returns and sets the end point of one polygon segment as point.

```
myElem.polygon.segments[1].endPoint.x = 188.66
myElem.polygon.segments[1].endPoint.y = 348.66
```

element.polygon.segments[ ].pen

Returns and sets the pen of one polygon segment as string. Any existing pen is allowed.

```
myElem.polygon.segments[1].pen = "Thick"
```

element.polygon.segments[ ].style

Returns and sets the style of one polygon segment as string. Any existing style is allowed.

```
myElem.polygon.segments[1].style = "Solid"
```

element.polygon.segments[ ].shadow

Returns and sets the shadow of one polygon segment as string. Any existing shadow is allowed.

```
myElem.polygon.segments[1].shadow = "Autom. Long"
```

element.marker

This property gives access to all marker attributes but it can not be used directly. It is assumed that myElem has already been defined as an element for the following samples

element.marker.startPoint

Returns and sets the start point of the marker as point.

```
myElem.marker.startPoint.x = 107.55
myElem.marker.startPoint.y = 237.55
```

element.marker.segments[ ].endPoint

Returns and sets the end point of one marker segment as point.

```
myElem.marker.segments[1].endPoint.x = 188.66
myElem.marker.segments[1].endPoint.x = 456.34
```

element.bezier

This property gives access to all bezier attributes but it can not be used directly. It is assumed that myElem has already been defined as an element for the following samples

element.bezier.startPoint

Returns and sets the start point of the bezier curve as point.

```
myElem.bezier.startPoint.x = 107.55
myElem.bezier.startPoint.x = 237.87
```

element.bezier.bezFlag

Returns and sets the type of the first bezier curve as integer. 0 means “curve” and 1 means “corner”.

```
myElem.bezier.bezFlag = 1
```

element.bezier.segments[ ].selected

Returns the selected state of one bezier segment as integer. This property is read only. 0 means “not selected” and 1 means “selected”.

```
MESSAGE myElem.bezier.segments[1].selected
```

element.bezier.segments[ ].h1Point

Returns and sets the end point of the first bezier segment handle as point.

```
myElem.bezier.segments[1].h1Point.x = 18.6
myElem.bezier.segments[1].h1Point.y = 18.6
```

element.bezier.segments[ ].h2Point

Returns and sets the end point of the second bezier segment handle as point.

```
myElem.bezier.segments[1].h2Point.x = 32.68
myElem.bezier.segments[1].h2Point.y = 232.67
```

element.bezier.segments[ ].endPoint

Returns and sets the end point of one bezier segment as point.

```
myElem.bezier.segments[1].endPoint.x = 129.56
myElem.bezier.segments[1].endPoint.y = 122.53
```

element.bezier.segment[ ].bezFlag

Returns and sets the type of one bezier segment as integer. 0 means “curve” and 1 means “corner”.

```
myElem.bezier.segments[1].bezFlag = 1
```

element.bezier.segments[ ].pen

Returns and sets the pen of one bezier segment as string. Any existing pen is allowed.

```
myElem.bezier.segments[1].pen = "Thick"
```

element.bezier.segments[ ].style

Returns and sets the style of one bezier segment as string. Any existing style is allowed.

```
myElem.bezier.segments[1].style = "Solid"
```

element.bezier.segments[ ].shadow

Returns and sets the shadow of one bezier segment as string. Any existing shadow is allowed.

```
myElem.bezier.segments[1].shadow = "Autom. Long"
```

element.text

This property gives access to all text attributes but it can not be used directly. It is assumed that myElem has already been defined as an element for the following samples

element.text.startPoint

Returns and sets the start point of a text element as point.

```
myElem.text.startPoint.x = 122.76
myElem.text.startPoint.y = 131.76
```

element.text.justHor

Returns and sets the horizontal justification of a text element as integer. Allowed values are 0 for “left”, 1 for “center” and (-1) for “right”.

```
myElem.text.justHor = 0
```

element.text.justVer

Returns and sets the vertical justification of a text element as integer. Allowed values are 1 for “top”, 2 for “middle”, 3 for “bottom” and 4 for “baseline”.

```
myElem.text.justVer = 2
```

element.text.stroke

Returns and sets the stroke width of a text element as float. You can enter any size but when you enter -256 you will deactivate the stroke.

```
myElem.text.stroke = 2
```

element.text.back_stroke

Returns and sets the stroke width of the background of a text element as float. You can enter any size but when you enter -256 you will deactivate the stroke.

```
myElem.text.back_stroke = 1.5
```

element.text.backstroke_color

Returns and sets the stroke color of the background of a text element as ColorSpec (see ColorSpec on page 171).

```
myElem.text.backstroke_color.type = "rgbValues"
myElem.text.backstroke_color.rgb.red = 188
myElem.text.backstroke_color.rgb.green = 208
myElem.text.backstroke_color.rgb.blue = 168
```

element.text.backfill_color

Returns and sets the color of the background of a text element as ColorSpec (see ColorSpec on page 171).

```
myElem.text.backfill_color.type = "rgbValues"
myElem.text.backfill_color.rgb.red = 234
myElem.text.backfill_color.rgb.green = 29
myElem.text.backfill_color.rgb.blue = 176
```

element.text.back_width

Returns and sets the width of the background of a text element as Float.

```
myElem.text.back_width = 2.2
```

element.text.back_height

Returns and sets the height of the background of a text element as Float.

```
myElem.text.back_height = 1.1
```

element.text.back_shape

Returns and sets the shape of the background of a text element as integer. Allowed values are 0 for “rectangle” and 1 for “circle”.

```
myElem.text.back_shape = 1
```

element.text.back_auto

Returns and sets if the background of a text element is adjusted to the text as boolean.

```
myElem.text.back_auto = TRUE
```

element.text.string

Returns and sets the displayed text of a text element as string.

```
myElem.text.string = "Hello World"
```

element.image

This property gives access to all image attributes but it can not be used directly. It is assumed that myElem has already been defined as an element for the following samples

element.image.startPoint

Returns and sets the start point of an image element as point. The start point is the images lower left corner.

```
myElem.image.startPoint.x = 122.76
myElem.image.startPoint.y = 131.76
```

element.image.horResolution

Returns the horizontal resolution of an image element in dots per inch (dpi) as float. This value is Read Only.

```
MESSAGE myElem.image.horResolution
```

element.image.verResolution

Returns the vertical resolution of an image element in dpi as float. This value is Read Only.

```
MESSAGE myElem.image.verResolution
```

element.image.pixelDepth

Returns the pixel depth of an image element as integer. This value is Read Only.

```
MESSAGE myElem.image.pixelDepth
```

element.image.kind

Returns the kind of image element selected as string. This value is Read Only.

```
MESSAGE myElem.image.kind
```

element.image.rowBytes

Returns the image's horizontal size in bytes as integer. This value is Read Only.

```
MESSAGE myElem.image.rowBytes
```

element.image.imageWidth

Returns the image's horizontal size in pixels as integer. This value is Read Only.

```
MESSAGE myElem.image.imageWidth
```

element.image.imageHeight

Returns the image's vertical size in pixels as integer. This value is Read Only.

```
MESSAGE myElem.image.imageHeight
```

21 Layer Object

The Layer Data Type is a reference on an existing layer in Arbortext IsoDraw.

```
DEFINE myLyr AS layer
myLyr = add layer "extra layer"
```

All layer commands (add, cut, paste ...) return a reference on the modified layer. The properties of this layer object can be queried and set.

```
MACRO Add Layer

DEFINE myLyr AS layer
myLyr = add layer "Engine" 188 0 29
MESSAGE myLyr.name

myLyr.name = "GearBox"
MESSAGE myLyr.name

END MACRO
```

It is assumed that myLyr has already been defined as a layer for the following samples

layer.name

Returns and sets the name of the layer as string.

```
myLyr.name = "GearBox"
MESSAGE myLyr.name
```

layer.screenColor

Returns and sets the screen color of the layer as RGBColor.

```
myLyr.screenColor.red = 100
myLyr.screenColor.green = 200
myLyr.screenColor.blue = 175
```

layer.locked

```
myLyr.locked = 0
```

Returns and sets the locked state of the layer as integer. Allowed values are 0 for “unlocked” and 1 for “locked”.

layer.active

Returns the active state of the layer as boolean. This property is read only.

```
MESSAGE myLyr.active
```

layer.printable

Returns and sets the printable state of the layer as boolean.

```
myLyr.printable = true
```

layer.exportable

Returns and sets the exportable state of the layer as boolean.

```
myLyr.exportable = false
```

layer.visible

Returns and sets the visible state of the layer as boolean.

```
myLyr.visible = true
```

layer.hasElements

Returns if the layer has elements as boolean. This property is read only.

```
MESSAGE myLyr.hasElements
```

layer.useColor

Returns and sets the use of color of the layer as boolean.

```
myLyr.useColor = true
```

layer.firstChild

This will return the first element which is located on the layer. The returned data type is element. This property is read only.

layer.lastChild

This will return the first element which is located on the layer. The returned data type is element. This property is read only.

layer.previousSibling

This property will return the previous layer of the layer list. The returned data type is layer. This property is read only.

```
MACRO LayerSibling
   DEFINE i AS integer
   DEFINE theLay AS layer
   i = activeDoc.layerCount
   theLay = activeDoc.layers[i]
   WHILE (exists (theLay) = true)
      MESSAGE "Processing layer " + theLay.name
      theLay = theLay.previousSibling
   END WHILE
END MACRO
```

layer.nextSibling

This property will return the next layer of the layer list. The returned data type is layer. This property is read only.

22

Application Object - User Interface Preferences

Applies to Arbortext IsoDraw CADprocess only

Arbortext IsoDraw automatically creates the Application Object at start-up. Unlike other data types, you do not `DEFINE` the Application Object. It is always available.

With the Application object you can gain access to nearly all options available in the Arbortext IsoDraw preferences.

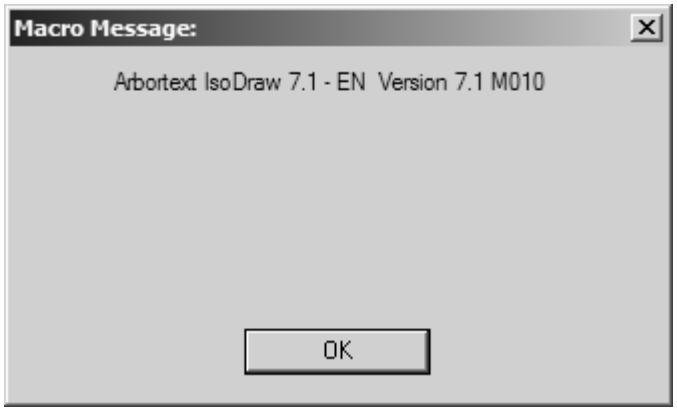
```
MESSAGE "Is Anti-Aliasing on?  ->" + app.useAntiAliasing

IF (app.useAntiAliasing = false) THEN
app.useAntiAliasing = true
MESSAGE "Now Anti-Aliasing is on"
END IF
```

app.version

These properties return software release information about the Arbortext IsoDraw application that is running the macro.

```
MESSAGE MESSAGE app.version.name + "  Version "
   + app.version.main + "." + app.version.sub
   + " M0" + app.version.xtra
#returns:
```



app.version.name

Returns the full Arbortext IsoDraw application name, the (version.revision) number, and the language. This property is read-only.

Note

The `app.version.name` property does not return the full application name for Arbortext IsoDraw releases prior to 7.0 F000.

```
MESSAGE app.version.name
```

app.version.main

Returns the version number of the currently running Arbortext IsoDraw application as integer. This property is read-only.

app.version.sub

Returns the revision number of the currently running Arbortext IsoDraw application as integer. This property is read-only.

app.version.build

Returns build number of the currently running Arbortext IsoDraw application as integer. This property is read-only.

app.version.xtra

Returns the release datecode of the currently running Arbortext IsoDraw application as integer. The datecode is of the form F*xxx* (for initial releases) or M*xxx* (for maintenance releases), where *xxx* is a 3-digit number. This property is read-only.

app.docCount

Returns the number of currently opened documents as integer. This property is read-only.

```
MESSAGE app.docCount
```

app.documents[ ].name

Returns the name of one currently opened document. This property is read-only.

```
MESSAGE app.documents[1].name
```

app.penCount

Returns the number of default pens as integer. This property is read-only.

```
MESSAGE app.penCount
```

app.styleCount

Returns the number of default styles as integer. This property is read-only.

```
MESSAGE app.styleCount
```

app.shadowCount

Returns the number of default shadows as integer. This property is read-only.

```
MESSAGE app.shadowCount
```

app.GridCount

Returns the number of default grids as integer. This property is read-only.

```
MESSAGE app.GridCount
```

app.colorCount

Returns the number of default colors as integer. This property is read-only.

```
MESSAGE app.colorCount
```

app.hatchingCount

Returns the number of hatchings as integer. This property is read-only.

```
MESSAGE app.hatchingCount
```

app.patternCount

Returns the number of patterns as integer. This property is read-only.

```
MESSAGE app.patternCount
```

app.formatCount

Returns the number of default text formats as integer. This property is read-only.

```
MESSAGE app.formatCount
```

app.calloutCount

Returns the number of default callouts as integer. This property is read-only.

```
MESSAGE app.calloutCount
```

app.password

Returns and sets the password property as string. #USE CAUTION#

```
app.password = "pumpernickel"
```

app.drawOffscreen

Returns and sets the draw Objects in background property as boolean.

```
app.drawOffscreen = false
```

app.useAntiAliasing

Returns and sets the useAntiAliasing property as boolean.

```
app.useAntiAliasing = true
```

app.showLineStyles

Returns and sets the showLineStyles attribute window property as boolean.

```
app.showLineStyles = true
```

app.showToolTips

Returns and sets the showToolTips property as boolean. This property corresponds to the **Show tooltips** option under **Tooltips** on the **Redraw** preferences page.

true	Causes tooltip text to appear when the pointer hovers over a tool button or symbol. ***Note*** *`app.showToolTips` must be `true` before the `app.showObjectTips` property can be set `true`. Also, setting `app.showObjectTips=true` while `app.showToolTips=false`will automatically set `app.showToolTips` to `true`.*

- If a macro changes the state of `app.showToolTips`, the change is saved as the new **Show tooltips** preference setting.
- Recording a macro that selects or clears the **Show tooltips** option adds an `app.showToolTips = {true|false}` statement to the macro.

```
app.showToolTips = true

#Note that:
app.showObjectTips = true
#Will also set:
app.showToolTips = true
```

app.showObjectTips

Applies to Arbortext IsoDraw 7.0 F000 and later:

Returns and sets the showObjectTips property as boolean. This property corresponds to the **Show object tips** option under **Tooltips** on the **Redraw** preferences page.

true	Object tip text appears when the pointer hovers over an object that has object tip text assigned to it. ***Note*** *`app.showToolTips` must be `true` before the `app.showObjectTips` property can be set `true`. Also, setting `app.showObjectTips=true` while `app.showToolTips=false`will automatically set `app.showToolTips` to `true`.*

- If a macro changes the state of `app.showObjectTips`, the change is saved as the new **Show object tips** preference setting.
- Recording a macro that selects or clears the **Show object tips** option adds an `app.showObjectTips = {true|false}` statement to the macro.

```
app.showObjectTips = true
#Will also set:
app.showToolTips = true
```

app.showRulers

Returns and sets the showRulers property as boolean.

```
app.showRulers = true
```

app.showCursorInfo

Returns and sets the showCursorInfo property as boolean.

```
app.showCursorInfo = true
```

app.magnetFlags

Returns and sets the magnetFlags property as integer. The allowed values are:

1	“Element points”
2	“Intersection points”
3	1 and 2
4	“Points on path”
5	1 and 4
6	2 and 4
7	all

```
app.magnetFlags = 4
```

app.selectableFills

Returns and sets the selectableFills property as boolean.

```
app.selectableFills = true
```

app.useIsoExtOnMac

Returns and sets the useIsoExtOnMac property as boolean.

```
app.useIsoExtOnMac = true
```

app.allowInternet

Returns and sets the allowInternet property as boolean, turning on and off the automatic check for product updates.

```
app.allowInternet = true
```

app.updatePeriod

Returns and sets the updatePeriod property as integer, settingthe frequency in days of automatic checks for product updates.

```
app.updatePeriod = 4
```

app.numberOfUndos

Returns and sets the numberOfUndos property as integer.

```
app.numberOfUndos = 25
```

app.autoSave

Returns and sets the autoSave property as boolean.

```
app.autoSave = true
```

app.autoSaveMinutes

Returns and sets the autoSaveMinutes property as float.

```
app.autoSaveMinutes = 5
```

app.useEllipsesIn3DTools

Returns and sets the Find elliptical arcs property as boolean. This option appears as a dialog when using the rotational surfaces or extrusion tool. It specifies whether elliptical arcs should be modeled as ellipses.

```
app.useEllipsesIn3DTools = true
```

app.preview

This property gives access to all preview properties but it can not be used directly.

app.preview.resolution

Returns and sets the resolution property as float.

```
app.preview.resolution = 72
```

app.preview.border

Returns and sets the border width property as float.

```
app.preview.border = 1
```

app.preview.width

Returns and sets the fixed width property in mm as float. The value (-1) disables the fixed width.

```
app.preview.width = 100
```

app.preview.kind

Returns and sets the image depth property as string. The allowed values are here "bitmap", "grayscale", "color8" and "color24".

```
app.preview.kind = "color24"
```

app.preview.compression

Returns and sets the compression property as integer. The allowed values are:

1	"none"
2	"PackBits"
3	"CCIT Group 3"
4	"CCIT Group 4"
5	"LZW"
6	"LZW2"

```
app.preview.compression = 2
```

app.preview.page

Returns and sets the dimension property as integer. The allowed values are 1 for "Extent" and 2 for "Drawing Size".

```
app.preview.page = 1
```

app.preview.intelOrder

Returns and sets the byte order property as boolean.

```
app.preview.intelOrder = true
```

app.preview.wantsPreview

Returns and sets the generate preview property as boolean.

```
app.preview.wantsPreview = true
```

app.compare

This property gives access to all compare attributes but it can not be used directly.

app.compare.deletedColor

Returns and sets the deleted elements property as RGBColor.

```
app.compare.deletedColor.red = 221
```

app.compare.changedColor

Returns and sets the changed elements property as RGBColor.

```
app.compare.changedColor.green = 212
```

app.compare.createdColor

Returns and sets the created elements property as RGBColor.

```
app.compare.createdColor.blue = 122
```

app.options3D

This property gives access to all 3D Options attributes but it can not be used directly.

app.options3D.showDialog

Returns and sets the Show options dialog before converting to 2D property as boolean.

```
app.options3D.showDialog = true
```

app.options3D.renderSmoothAngle

Returns and sets the Max. smoothing angle property as integer. Allowed values are from 0 to 120.

```
app.options3D.renderSmoothAngle = 65
```

app.options3D.objectAccuracy

Returns and sets the Tesselation Accuracy property as integer. Allowed values are from 0 to 100.

```
app.options3D.objectAccuracy = 32
```

app.project3D

These properties return and set 3D projection attributes—but they can not be used directly. Any changes made to `app.project3D` properties in a macro are saved in the Arbortext IsoDraw CADprocess **3D Options** preferences. (See 3D Options in the *Arbortext IsoDraw User's Reference*.)

Note

*The `app.project3D.as...` properties below correspond to mutually exclusive options in the Arbortext IsoDraw CADprocess **3D Projection** dialog box. To ensure the desired 3D projection results, one— and only one—of these properties must be `true` before the macro executes the `3D PROJECT` command.*

app.project3D.asWireframe

Returns and sets the wireframe property as boolean. This property corresponds to the **as wireframe** option in the **3D Projection** dialog box.

true	Projects a 3D view into a 2D illustration as a wireframe drawing.

```
app.project3D.asWireframe = true
```

app.project3D.asHLR

Returns and sets the remove hidden lines property as boolean. This property corresponds to the **remove hidden lines** option in the **3D Projection** dialog box.

true	Projects a 3D view into a 2D illustration with Hidden Line Removal (HLR).

```
app.project3D.asHLR = true
```

app.project3D.HLR_type

For HLR 3D projection only: Returns and sets the HLR 2D element type as string. This determines the type of 2D element—Line, Polyline, or Bézier path—that all 3D data elements are converted into during 3D projection. This property corresponds to the three HLR 3D options: **as lines**, **as polylines**, and **as Bézier paths**, under **Optimize** in the 3D **Options** dialog box.

lines	3D elements convert to 2D Line elements.
polylines	3D elements convert to 2D Polyline elements.
beziers	3D elements convert to 2D Bézier path elements.

```
app.project3D.HLR_type = "beziers"
```

app.project3D.HLR_plusEllipses

For HLR 3D projection only: Returns and sets the HLR 2D ellipse element conversion property as boolean. This property corresponds to the HLR 3D option, **generate ellipses**, under **Optimize** in the 3D **Options** dialog box.

true	Converts ellipses formed by Line, Polyline, or Bézier path elements during 3D projection into Ellipse elements in the 2D illustration.

```
app.project3D.HLR_plusEllipses = true
```

app.project3D.HLR_optSpeed

For HLR 3D projection only: Returns and sets the HLR 3D projection speed optimization property as boolean. This property corresponds to the HLR 3D option, **Optimize for speed**, under **Advanced Settings** in the 3D **Options** dialog box.

true	Accelerates the 3D projection process by performing simpler 3D-to-2D conversion calculations.

```
app.project3D.HLR_optSpeed = true
```

app.project3D.Raster_kind

For shaded (raster) 3D projection only: Returns and sets the color depth of the target 2D raster image as string. This property corresponds to the three color **Depth** options: **Lineart image (1 Bit)**, **Color image (8 Bit)**, and **Color image (32 Bit)**, in the **shaded** dialog box. (To access **Depth** settings, select **shaded** in the **3D Projection** dialog box, then click **Options**.)

bitmap	Converts 2D raster image to 1-bit black and white lineart
color8	Converts 2D raster image to 8-bit color
color32	Converts 2D raster image to 32-bit color

```
app.project3D.Raster_kind = "color32"
```

app.project3D.resolution

For shaded (raster) 3D projection only: Returns and sets the resolution in dots per inch (dpi) of the target 2D raster image as integer. Allowed values are from `1` dpi to `1139` dpi. This property corresponds to the **Resolution** option in the **shaded** dialog box. (To access the **Resolution** setting, select **shaded** in the **3D Projection** dialog box, then click **Options**.)

```
app.project3D.resolution = 300
```

app.project3D.level

For shaded (raster) 3D projection only: Returns and sets (as integer) the property that specifies:

- If object information is preserved or discarded during 3D projection.
- How many layers will be created in the 2D illustration for 3D objects that are converted to 2D objects. Each layer in the 2D illustration contains a separate raster image.

Allowed values are `1` or greater. This property corresponds to the **Layer** option in the **shaded** dialog box. (To access the **Layer** setting, select **shaded** in the **3D Projection** dialog box, then click **Options**.)

1	Object information is discarded during 3D projection. The target 2D raster image has one layer.
2 or greater	Object information is preserved during 3D projection. The target 2D raster image has the specified number of layers. Each layer contains a raster image of a converted 3D object.

```
app.project3D.level = 1
```

app.project3D.asShaded

Returns and sets the shaded property as boolean. This property corresponds to the **shaded** option in the **3D Projection** dialog box.

true	Projects a 3D view into a 2D illustration as a shaded raster image. The 2D image matches the display type in 3D mode.

```
app.project3D.asShaded = true
```

app.project3D.as3D

Returns and sets the keep 3D data property as boolean. This property corresponds to the **keep 3D data** option in the **3D Projection** dialog box.

true	Projects a 3D view into a 2D illustration and .

```
app.project3D.as3D = true
```

app.project3D.useThickThin

Returns and sets the use create thick/thin lines property as boolean.

true	Projects a 3D view into a 2D illustration as a wireframe drawing.

```
app.project3D.useThickThin = true
```

Use `app.project3D.useThickThin` to create thick/thin lines in the 2D illustration projected from a 3D view. It corresponds to the **use thick/thin lines** option in the **3D Projection** dialog box; `true` selects this option; `false` clears this selection.

app.project3D.createSurfaceBorders

Returns and sets the use create surface borders property as boolean.

```
app.project3D.createSurfaceBorders = false
```

app.project3D.smoothSurfaces

Returns and sets the Thin line threshold property as integer. Allowed values are from 0 to 100.

```
app.project3D.smoothSurfaces = 65
```

app.project3D.thickPen

Returns and sets the for thick lines property as string. Allowed values are any existing pen name.

```
app.project3D.thickPen = "$ISO_Thick"
```

app.project3D.thinPen

Returns and sets the for thin lines property as string. Allowed values are any existing pen name.

```
app.project3D.thinPen = "$ISO_Thin"
```

app.dimensions

This property gives access to all dimension attributes but it can not be used directly.

app.dimensions.baseOffset

Returns and sets the base offset property as float.

```
app.dimensions.baseOffset = 2
```

app.dimensions.extendOffset

Returns and sets the extend offset property as float.

```
app.dimensions.extendOffset = 2
```

app.dimensions.verticalOffset

Returns and sets the vertical offset property as float.

```
app.dimensions.verticalOffset = 0
```

app.dimensions.minLength

Returns and sets the minimum length property as float.

```
app.dimensions.minLength = 5
```

app.dimensions.decimalPlaces

Returns and sets the decimal places property as integer.

```
app.dimensions.decimalPlaces = 2
```

app.dimensions.leaderPen

Returns and sets the leader pen property as string. All existing pens are allowed.

```
app.dimensions.leaderPen = "Thick"
```

app.dimensions.leaderStyle

Returns and sets the leader style property as string. All existing styles are allowed.

```
app.dimensions.leaderStyle = "Solid"
```

app.dimensions.leaderShadow

Returns and sets the leader shadow property as string. All existing shadows are allowed.

```
app.dimensions.leaderShadow = "Autom. Long"
```

app.dimensions.startPen

Returns and sets the start pen property as string. All existing pens are allowed.

```
app.dimensions.startPen = "Thick"
```

app.dimensions.startStyle

Returns and sets the start style property as string. All existing styles are allowed.

```
app.dimensions.startStyle = "Solid"
```

app.dimensions.startShadow

Returns and sets the start shadow property as string. All existing shadows are allowed.

```
app.dimensions.startShadow = "Autom. Long"
```

app.dimensions.endPen

Returns and sets the end pen property as string. All existing pens are allowed.

```
app.dimensions.endPen = "Thick"
```

app.dimensions.endStyle

Returns and sets the end style property as string. All existing styles are allowed.

```
app.dimensions.endStyle = "Solid"
```

app.dimensions.endShadow

Returns and sets the end shadow property as string. All existing shadows are allowed.

```
app.dimensions.endShadow = "Autom. Long"
```

app.dimensions.stripZeros

Returns and sets the strip zeros property as boolean.

```
app.dimensions.stripZeros = true
```

app.dimensions.txtFormat

Returns and sets the text format property as string. All existing text formats are allowed.

```
app.dimensions.txtFormat = "Normal"
```

app.dimensions.txtFont

Returns and sets the text font property as string. All installed fonts are allowed.

```
app.dimensions.txtFont = "Arial"
```

app.dimensions.txtFace

Returns and sets the text face property as string. Allowed values are "normal", "bold", "italic" and "bolditalic".

```
app.dimensions.txtFace = "bold"
```

Returns and sets the text size property as float.

```
app.dimensions.txtSize = 24.5
```

app.dimensions.txtSize

app.dimensions.txtStrokeColor

Returns and sets the text strokecolor property as ColorSpec (see ColorSpec on page).

```
app.dimensions.txtStrokeColor.type = "cmykValues"
```

app.dimensions.txtStroke

Returns and sets the text stroke property as float.

```
app.dimensions.txtStroke
= 2.2
```

app.dimensions.textFillcolor

Returns and sets the text fillcolor property as ColorSpec (see ColorSpec on page).

```
app.dimensions.textFillColor.type = "rgbValues"
app.dimensions.textFillColor.rgb.red = 168
app.dimensions.textFillColor.rgb.green = 168
app.dimensions.textFillColor.rgb.blue = 168
```

app.grid

This property gives access to all grid attributes but it can not be used directly.

app.grid.gridSize

Returns and sets the grid size property as float.

```
app.grid.gridSize = 20
```

app.grid.radius

Returns and sets the magnetic radius property as float.

```
app.grid.radius = 5
```

app.grid.elementMagnet

Returns and sets the magnetic elements property as boolean.

```
app.grid.elementMagnet = true
```

app.grid.gridMagnet

Returns and sets the magnetic grid points property as boolean.

```
app.grid.gridMagnet = true
```

app.grid.showGrid

Returns and sets the show grid property as boolean.

```
app.grid.showGrid = true
```

app.grid.showDim

Returns and sets the show dimensions property as boolean.

```
app.grid.showDim = true
```

app.grid.constrain

Returns and sets the align to grid property as boolean.

```
app.grid.constrain = true
```

app.grid.gridInFront

Returns and sets the grid in front property as boolean.

```
app.grid.gridInFront = true
```

app.grid.noFShortInIso

Returns and sets the no isometric foreshortening property as boolean.

```
app.grid.noFShortInIso = true
```

app.grid.gridColor

Returns and sets the color property as RGBColor.

```
app.grid.gridColor.red = 0
```

app.grid.firstSelColor

Returns and sets the first color property as RGBColor.

```
app.grid.firstSelColor.green = 0
```

app.grid.secondSelColor

Returns and sets the second color property as RGBColor.

```
app.grid.secondSelColor.blue = 0
```

app.grid.preferredGrid

Returns and sets the name of the standard Grid as string. Only the names of existing grids can be set.

```
app.grid.preferredGrid = "Plane view"
```

app.window

This property gives access to all window attributes but it can not be used directly.

app.window.pageX

Returns and sets the window x-cords property as float.

```
app.window.pageX = 200
```

app.window.pageY

Returns and sets the window y-cords property as float.

```
app.window.pageY = 230
```

app.window.overlapX

Returns and sets the print overlap x-cords property as float.

```
app.window.overlapX = 10
```

app.window.overlapY

Returns and sets the print overlap y-cords property as float.

```
app.window.overlapY = 10
```

app.window.Scale

Returns and sets the window scale property as float.

```
app.window.Scale = 10
```

app.window.dimScale

Returns and sets the dimension scaling property as float.

```
app.window.dimScale = 7
```

app.window.preview

Returns and sets the objects preview property as integer. The allowed values are 0 for “no preview”, 1 for “a preview”, 2 for “visible hotspots” and 4 for “visible objects”. 3 is for “preview and hotspots”, 6 is for “hotspots and objects” and 7 is for everything.

```
app.window.preview = 6
```

app.window.is3D

Returns the 3D property as boolean. This property is read-only.

```
MESSAGE app.window.is3D
```

app.window.curSystem

Returns and sets the current system property as string. Allowed values are "mm", "in" and "pt".

```
app.window.curSystem = "mm"
```

app.shadow

This property gives access to all shadow attributes but it can not be used directly.

app.shadow.shadowWidth

Returns and sets the shadow width property as float.

```
app.shadow.shadowWidth = 5
```

app.shadow.shadowFactor

Returns and sets the shadow factor property as float.

```
app.shadow.shadowFactor = 2
```

app.thread

This property gives access to all thread attributes but it can not be used directly.

app.thread.inner.upTo1

Returns and sets the first "inner thread up to" property as float.

```
app.thread.inner.upTo1 = 20
```

app.thread.inner.upTo2

Returns and sets the second "inner thread up to" property as float.

```
app.thread.inner.upTo2 = 40
```

app.thread.inner.size1

Returns and sets the first inner thread distance property as float.

```
app.thread.inner.size1 = 2
```

app.thread.inner.size2

Returns and sets the second inner thread distance property as float.

```
app.thread.inner.size2 = 2.5
```

app.thread.inner.size3

Returns and sets the third inner thread distance property as float.

```
app.thread.inner.size3 = 3
```

app.thread.outer.upTo1

Returns and sets the first "outer thread up to" property as float.

```
app.thread.outer.upTo1 = 20
```

app.thread.outer.upTo2

Returns and sets the second "outer thread up to" property as float.

```
app.thread.outer.upTo2 = 40
```

app.thread.outer.size1

Returns and sets the first outer thread distance property as float.

```
app.thread.outer.size1 = 2
```

app.thread.outer.size2

Returns and sets the second outer thread distance property as float.

```
app.thread.outer.size2 = 2.5
```

app.thread.outer.size3

Returns and sets the third outer thread distance property as float.

```
app.thread.outer.size3 = 3
```

app.thickthin

This property gives access to all thick/thin attributes but it can not be used directly.

app.thickthin.useThickThin

Returns and sets the "use thick thin technique" property as boolean.

```
app.thickthin.useThickThin = true
```

app.thickthin.thickPen

Returns and sets the thick pen property as string. The name of every existing pen is an allowed value.

```
app.thickthin.thickPen = "Thick"
```

app.thickthin.thinPen

Returns and sets the thin pen property as string. The name of every existing pen is an allowed value.

```
app.thickthin.thinPen = "Thin"
```

app.thickthin.useEllipsePens

Returns and sets the "specify pens for ellipses and threads" property as boolean.

```
app.thickthin.useEllipsePens = true
```

app.thickthin.thick1

Returns and sets the first thick pen property as string. The name of every existing pen is an allowed value.

```
app.thickthin.thick1 = "Thick"
```

app.thickthin.thin1

Returns and sets the first thin pen property as string. The name of every existing pen is an allowed value.

```
app.thickthin.thin1 = "Thin"
```

app.thickthin.thick2

Returns and sets the second thick pen property as string. The name of every existing pen is an allowed value.

```
app.thickthin.thick2 = "Thick"
```

app.thickthin.thin2

Returns and sets the second thin pen property as string. The name of every existing pen is an allowed value.

```
app.thickthin.thin2 = "Thin"
```

app.thickthin.thick3

Returns and sets the third thick pen property as string. The name of every existing pen is an allowed value.

```
app.thickthin.thick3 = "Thick"
```

app.thickthin.thin3

Returns and sets the third thin pen property as string. The name of every existing pen is an allowed value.

```
app.thickthin.thin3 = "Thin"
```

app.thickthin.upto1

Returns and sets the first up to property as float.

```
app.thickthin.upto1 = 20
```

app.thickthin.upto2

Returns and sets the second up to property as float.

```
app.thickthin.upto2 = 40
```

app.polygontool

This property gives access to all polygon tool attributes but it can not be used directly.

app.polygontool.sides

Returns and sets the polygon sides property as integer.

```
app.polygontool.sides = 7
```

app.ellipsetool

This property gives access to all ellipse tool attributes but it can not be used directly.

app.ellipsetool.value

Returns and sets the ellipse value property as float.

```
app.ellipsetool.value = 50
```

app.rectangletool

This property gives access to all rectangle tool attributes but it can not be used directly.

app.rectangletool.radius

Returns and sets the rectangle radius value property as float.

```
app.rectangletool.radius = 25
```

app.background

This property gives access to all background attributes but it can not be used directly.

app.background.inColor

Returns and sets the in color property as boolean.

```
app.background.inColor = true
```

Returns and sets the intensity property as float. The properties range is from 0 to 100.

```
app.background.intensity = 50
```

app.background.intensity

app.background.Color

Returns and sets the screen color property as RGBColor.

```
app.background.Color = "{RGB 107 69 77}"
```

app.lineOptions

This property gives access to all line option attributes but it can not be used directly.

app.lineOptions.lineCap

Returns and sets the ends property as integer. Allowed values are 0 for “flat”, 1 for “round” and 2 for “square”.

```
app.lineOptions.lineCap = 1
```

app.lineOptions.lineJoin

Returns and sets the corners property as integer. Allowed values are 0 for “mitered”, 1 for “rounded” and 2 for “bevel”.

```
app.lineOptions.lineJoin = 2
```

app.lineOptions.miterLimit

Returns and sets the miter limit property as integer.

```
app.lineOptions.miterLimit = 4
```

app.lineOptions.overPrint

Returns and sets the overprint property as integer. Allowed values are 0 for “none”, 1 for “stroke”, 2 for “square” and 3 for “stroke and square”.

```
app.lineOptions.overPrint = 2
```

app.simpleEllipsePrinting

This property sets and returns the simple ellipse printing as boolean.

```
app.simpleEllipsePrinting = true
```

app.curMacroTransform

This property returns the current transformation properties but it can not be used directly.

app.curMacroTransform.offset

Returns the offset property as point. This property is read-only.

```
MESSAGE app.curMacroTransform.offset
```

app.curMacroTransform.angle

Returns the angle property as float. This property is read-only.

```
MESSAGE app.curMacroTransform.angle
```

app.curMacroTransform.scale

Returns the scale property as point. This property is read-only.

```
MESSAGE app.curMacroTransform.scale
```

app.dtd

This property returns and sets the default dtd-file as string.

```
app.dtd = "webcgm"
```

app.standardTxtFormat

This property returns and sets which text format is the standard text format as string.

```
app.standardTxtFormat = "Normal"
```

app.option

This property gives access to the optimize options.

app.option.smooth_tolerance

Returns and sets the Tolerance for smoothing property as float. This setting does not make a permanent change. The value will be reset to 0.1 each time Arbortext IsoDraw is restarted.

```
app.option.smooth_tolerance
= 0.1
```

app.interaction

Returns and sets the display of certain dialogs as boolean. We do not recommend setting this to false permanently as many dialogs will not appear during your use of Arbortext IsoDraw.

```
app.interaction = false
```

```
#Turning off dialogs
app.interaction = false
Select IF Type is equal to 'callout'
```

```
Cut
#Turn dialogs back on
app.interaction = true
```

app.lastMacroError

Applies to Arbortext IsoDraw 7.0 F000 and later:

Returns the number of the last error that occurred. This property is read-only.

app.currentMacro

Applies to Arbortext IsoDraw 7.0 F000 and later:

Returns descriptive and status information about the macro that is currently running, such as its name and how many variables or parameters it contains. This property is read-only.

app.currentMacro.name

Returns the current macro's name as string.

```
DEFINE mName AS string
mName = app.currentMacro.name
MESSAGE "The macro '" + mName + "' is running."
```

app.currentMacro.countVars

Returns the number of variables defined in the macro as integer.

```
Define x AS float
Define y AS float
DEFINE vCnt AS integer
vCnt = app.currentMacro.countVars
MESSAGE "The variable count is " + vCnt
```

app.currentMacro.countParams

Returns the number of parameters as integer.

```
DEFINE pCnt AS integer
pCnt = app.currentMacro.countParams
MESSAGE "The parameter count is: " + pCnt
```

app.currentMacro.unicodeFlag

Returns the current Unicode flag value as integer. The flag values are:

0	8-bit / 0*x*FFFE
0*x*FEFF	Unicode file

```
DEFINE uFlag AS integer
uFlag = app.currentMacro.unicodeFlag
MESSAGE "The current unicode flag is: " + uflag
```

app.currentMacro.status

Returns a status value for the currently running macro as integer. Status values are:

0	Temporarily stopped
1	In menu
2	SubMacro
3	Write protected
4	Internal

```
MESSAGE "The current macro status is: " + app.currentMacro.status
```

app.currentMacro.callDepth

Returns the call stack depth of the current macro as integer.

```
MESSAGE app.currentMacro.callDepth
```

app.currentMacro.activeLine

Returns the text content of the active macro line as string.

```
MESSAGE app.currentMacro.activeLine
```

23

Application Object - Data Exchange Preferences

You can gain access to the data exchange settings through the Application Object.

Import CGM

app.cgm.showDialog

Returns and sets if the cgm import dialog is shown as boolean.

```
app.cgm.showDialog = TRUE
```

app.cgm.usebackcolor

Returns and sets if the backcolor is used as boolean.

```
app.cgm.usebackcolor = TRUE
```

app.cgm.scale

Returns and sets the scale as float. 1 stands for 100% and 0.1 for 10%.

```
#setting the scale to 55%
app.cgm.scale = 0.55
```

app.cgm.entities

Sets the CGM options as integer. Please use this formula to calculate the allowed values:

value = 2^x

x	Option
0	“Polylines“
1	“Disjoint Polyline“
2	“Polymarker“
3	“Text“
4	“Restricted Text“
5	“Append Text“
6	“Polygon“
7	“Polygon Set“
8	“Cell Array“
9	“Rectangle“
10	“Circle“
11	“Circular Arc 3 Point“
12	“Circular Arc 3 Point Close“
13	“Circular Arc Center“
14	“Circular Arc Center Close“
15	“Ellipse“
16	“Elliptical Arc“
17	“Elliptical Arc Close“
18	“Circular Arc Centre Reversed“
19	“Hyperbolic Arc“
20	“Parabolic Arc“
21	“Non-Uniform B-Spline“
22	“Non-Uniform Rational B-Spline“

23	“Polybezier“
24	“Segments“
25	“Figures“
26	“Tiles“
27	“Background Colour“
28	“Clip Element V1“
29	“Clip Element V3“
30	“Appl. Str. Elements“

Add values to activate different options.

```
#activating the text-option only
app.cgm.entities = 8

#activating the text- (2^2=4), polymarker-
#(2^3=8) and restricted text-option (2^4=16)
app.cgm.entities = 4 + 8 + 16
```

app.cgm.importtype

Returns and sets the “Read from version 1/2 as” property as integer. Allowed values are 1 for “basic”, 2 for “boxed cap” and 3 for “isotropic cap”.

```
app.cgm.importtype = 1
```

app.cgm.pictures

Returns and sets the import pictures property as integer. Setting this property to 0 means importing all pictures and setting the property to another number means importing just one specific picture.

```
app.cgm.pictures = 0
```

app.cgm.cgmlinecap

Returns and sets the line caps as integer. Allowed values are 0 for “flat”, 1 for “rounded” and 2 for “rectangle”.

```
app.cgm.cgmlinecap = 2
```

app.cgm.cgmlinejoin

Returns and sets the line join as integer. Allowed values are 0 for “mitered”, 1 for “rounded” and 2 for “bevel”.

```
app.cgm.cgmlinejoin = 2
```

app.cgm.pageSizeKind

Returns and sets the page size value as integer. Allowed values are 1 for “default” and 2 for use “VDC extend”.

```
app.cgm.pageSizeKind = 1
```

Export CGM

app.cgm.profile

Returns and sets the CGM profile as integer. The allowed values (*n*) and their equivalent macro constant names are listed in the table below. (See CGM Profile Numbers and Names on page 303 for values in older Arbortext IsoDraw releases.)

n	CGM Profile Name	Macro Constant Name
1	ISO 8632:1999	$CGM_Profile.ISO_8632_1999
2	WebCGM 1.0	$CGM_Profile.WebCGM_1_0
3	ATA GREXCHANGE V2.8	$CGM_Profile.ATA_GREXCHANGE_2_8
4	ATA GREXCHANGE V2.7	$CGM_Profile.ATA_GREXCHANGE_2_7
5	ATA GREXCHANGE V2.6	$CGM_Profile.ATA_GREXCHANGE_2_6
6	ATA GREXCHANGE V2.5	$CGM_Profile.ATA_GREXCHANGE_2_5
7	ATA GREXCHANGE V2.4	$CGM_Profile.ATA_GREXCHANGE_2_4
8	ATA GREXCHANGE V2.5/IsoDraw	$CGM_Profile.ATA_GREX-CHANGE_2_5_ISODRAW
9	MIL-D-28003A	$CGM_Profile.MIL_D_28003A
10	SAE J2008	$CGM_Profile.SAE_J2008
11	Model (8632:1992)	$CGM_Profile.Model_8632_1992
12	ISO ISP 12071-1	$CGM_Profile.ISO_ISP_12071_1
13	ISO ISP 12072-1	$CGM_Profile.ISO_ISP_12072_1
14	ISO ISP 12073-1	$CGM_Profile.ISO_ISP_12073_1
15	ISO ISP 12074-1	$CGM_Profile.ISO_ISP_12074_1
16	ATA GREXCHANGE V2.9	$CGM_Profile.ATA_GREXCHANGE_2_9
17	S1000D V2.2[1]	$CGM_Profile.S1000D_2_2
18	WebCGM 2.0	$CGM_Profile.WebCGM_2_0
19	ATA GREXCHANGE V2.10	$CGM_Profile.Current_ATA[2]
20	S1000D V2.3	No macro constant name
21	WebCGM 2.1	$CGM_Profile.WebCGM_2_1

1. S1000D V2.2 export is no longer supported.
2. The constant name $CGM_Profile.Current_AT always applies to the newest ATA GREXCHANGE profile that this release of Arbortext IsoDraw supports. When Arbortext IsoDraw supports a newer version, V2.10 will be assigned a different constant name; e.g., $CGM_Profile.ATA_GREXCHANGE_2_10.

Note

When setting the value of `app.cgm.profile`, it is best to use the constant name, which is fixed, rather than the number, which might change from one release to the next.

```
app.cgm.profile = $CGM_Profile.ATA_GREXCHANGE_2_9
```

app.cgm.useAECMAs1000d

Returns and sets the S1000d compliant as integer. Allowed values are 0 for “do not use” and 1 for “use”.

```
app.cgm.useAECMAs1000d = 1
```

app.cgm.encoding

Returns and sets the CGM encoding type as integer. Allowed values are 1 for “binary” and 2 for “text”.

```
app.cgm.encoding = 1
```

app.cgm.vdcType

Returns and sets the VDC-Type as integer. Allowed values are 1 for “16bit integer”, 2 for “32bit integer” and 3 for “real”.

```
app.cgm.vdcType = 2
```

app.cgm.compressionColor

Returns and sets the compression color type as integer. Allowed values are:

1	“none”
2	“Runlength”
3	“JPEG”
4	“PNG”

```
app.cgm.compressionColor = 3
```

app.cgm.compressionBW

Returns and sets the B/W compression type as integer. Allowed values are:

1	“none”
2	“runlength”
3	“CCITT Group3 (T4)”
4	“CCITT Group3 (T6)”
5	“PNG”

```
app.cgm.compressionBW = 2
```

app.cgm.jpegQuality

(Applies to Arbortext IsoDraw 7.0 F000 and later.)

Returns and sets the quality for embedded JPEG images as integer. Allowed values are 0 to 100.

0	Optimizes image quality.
1	Optimizes image compression.
2	Lowest quality; highest compression.
3…99	As the value increases, quality increases and compression decreases.
100	Highest quality; lowest compression.

- If a macro changes the value of `app.cgm.jpegQuality`, the change is saved as the new CGM export JPEG image quality preference setting.

- Recording a macro that changes this property adds an `app.cgm.jpegQuality = value` statement to the macro.

```
app.cgm.jpegQuality = 40
```

app.cgm.textType

Returns and sets the text type as integer. Allowed values are:

0	“none”
1	“RT basic”
2	“RT boxed-cap”
3	“RT boxed-all”
4	“RT isotropic-cap”
5	“RT isotropic-all”
6	“RT justified”

```
app.cgm.textType = 6
```

app.cgm.exportType

Returns and sets the export text type as integer. Allowed values are 1 for “basic”, 2 for “boxed-cap” and 3 for “isotropic-cap”.

```
app.cgm.exportType = 1
```

app.cgm.ellipseAsPoly

Returns and sets if ellipses should be exported as polylines as boolean.

```
app.cgm.ellipseAsPoly = FALSE
```

app.cgm.layerToPicture

Returns and sets if a separate picture should be generated for every layer as boolean.

```
app.cgm.layerToPicture = FALSE
```

app.cgm.nativeCGM

Returns and sets if CGM should be used as the native file format as boolean.

```
app.cgm.nativeCGM = FALSE
```

app.cgm.changeURL

Returns and sets if .iso should be replaced with .cgm in links as boolean.

```
app.cgm.changeURL = TRUE
```

app.cgm.vdcExtentMode

Returns and sets the VDC extent mode as integer. Allowed values are 0 for “include all points”, 1 for “bounding box” and 2 for “page size”.

```
app.cgm.vdcExtentMode = 2
```

app.cgm.styleHandling

Returns and sets the linestyle handling mode as integer. Allowed values are:

1	“map”
2	“split”
3	“LETD if not solid”
4	“LETD always”

```
app.cgm.styleHandling = 3
```

app.cgm.extension

Returns and sets the file extension as string.

```
app.cgm.extension = "cgm"
```

app.cgm.version

Returns and sets the CGM version as integer. Allowed values are 1 for “1”, 2 for “2”, 3 for “3” and 4 for “4”.

```
app.cgm.version = 1
```

app.cgm.createXCF

(Applies to Arbortext IsoDraw 7.0 F000 and later.)

Returns and sets the enabled or disabled state of XML companion file (XCF) export as integer. This property corresponds to the advanced CGM export option, **Generate XML companion files**, on the **CGM Export** preferences page. Allowed values are 0 and 1.

Note

Some CGM export profiles do not support XCF export.

0	Do not write XCF during CGM export.
1	Write XCF during CGM export.

- If a macro changes the value of `app.cgm.createXCF`, the change is saved as the new **Generate XML companion files** preference setting.
- The macro recorder will not record changes to this property.

```
app.cgm.createXCF = 1
```

Export EPS

app.eps.extension

Returns and sets the file extension as string.

```
app.eps.extension = "eps"
```

app.eps.preview

Returns and sets the preview as integer. Allowed values are 1 for “no preview”, 2 for “for MS-DOS” and 3 for “for Mac”.

```
app.eps.preview = 2
```

app.eps.embedFonts

Returns and sets if type 1 fonts should be embedded as boolean.

```
app.eps.embedFonts = FALSE
```

Import Illustrator

app.ill.ignoreGuides

Returns and sets if the guides are ignored as boolean.

```
app.ill.ignoreGuides = TRUE
```

Export Illustrator

app.ill.extension

Returns and sets the file extension as string.

```
app.ill.extension = "ai"
```

app.ill.includeEPS

Returns and sets if the EPS file is included as boolean.

```
app.ill.includeEPS = FALSE
```

app.ill.version

Returns and sets the illustrator version as integer. Allowed values are 0 for “Adobe Illustrator™ 7.0”, 1 for “Adobe Illustrator™ 88” and 2 for “Adobe Illustrator™ 6.0”.

```
app.ill.version = 0
```

Export SVG

app.svg.extension

Returns and sets the file extension as string.

```
app.svg.extension = "svg"
```

app.svg.compressed

Returns and sets if an compressed svgz file is created as boolean.

```
app.svg.compressed = FALSE
```

app.svg.embedRaster

Returns and sets if the raster image is embedded as boolean.

```
app.svg.embedRaster = FALSE
```

app.svg.size

Returns and sets the Illustration size as integer. Allowed values are:

1	Scalable illustration
2	Fixed size illustration

```
app.svg.size = 2
```

app.svg.page

Returns and sets the Dimensions as integer. Allowed values are:

1	Drawing size
2	Extent

```
app.svg.page = 1
```

app.svg.encoding

Returns and sets the file encoding as integer. Allowed values are:

1	ISO Latin
2	UTF-8
3	UTF-16

```
app.svg.encoding = 2
```

app.svg.hotspot_regions

Returns and sets the Hotspot regions as integer. Allowed values are:

1	Use visible geometry only
2	Create region paths

```
app.svg.hotspot_regions = 2
```

app.svg.object_attributes

Returns and sets the Object attributes as integer. Allowed values are:

1	Don’t export object attributes
2	Export attributes using namespace

```
app.svg.object_attributes= 2
```

app.svg.export_metadata

Returns and sets the option to export SVG metadata as integer. Allowed values are:

1	Do not export
2	Export all
3	Export Arbortext IsoDraw metadata only

Import SVG

app.svg.showDialog

Returns and sets if the SVG import dialog is shown as boolean.

app.svg.pageSizeKind

Returns and sets the page size kind as integer. Allowed values are:

1	Default
2	Adjust to drawing size

app.svg.import_metadata

Returns and sets the option to import SVG metadata as integer. Allowed values are:

1	Ignore
2	Import all
3	Import Arbortext IsoDraw metadata only

Import IGES

app.iges.showDialog

Returns and sets if the IGES import dialog is shown as boolean.

```
app.iges.showDialog = TRUE
```

app.iges.scale

Returns and sets the scale as float. 1 is used for 100% and 0.1 for 10%.

```
#setting the scale to 21%
app.iges.scale = 0.21
```

app.iges.platformIn

Returns and sets the import platform as integer. Allowed values are 1 for “Mac”, 2 for “MS Win” and 3 for “Unix”.

```
app.iges.platformIn = 3
```

app.iges.selectGroups

Returns and sets if the assemblies should be selected as boolean.

```
app.iges.selectGroups = FALSE
```

app.iges.createInfo

Returns and sets if the object informations for the assemblies should be created as boolean.

```
app.iges.createInfo = FALSE
```

app.iges.hsType

Returns and sets which hotspot type is created for the assemblies as integer. Allowed values are 0 for “no hotspot” and 1 for “lines of objects”.

```
app.iges.hsType = 1
```

app.iges.ignoreInvisElts

Returns and sets if invisibly elements should be ignored as boolean.

```
app.iges.ignoreInvisElts = TRUE
```

app.iges.use102

Returns and sets if the entity 102 should be used as bezier path as boolean.

```
app.iges.use102 = TRUE
```

app.iges.convertAnnotations

Returns and sets if the annotations of the elements should be converted as boolean.

```
app.iges.convertAnnotations = TRUE
```

app.iges.views

Returns and sets how elements which are not assigned to a view should be handled as integer. Allowed values are 0 for “without any view” and 1 for “in all views”.

```
app.iges.views = 1
```

app.iges.bestView

Returns and sets is the best view is chosen automatically as integer. Allowed values are 0 for “do not check best view” and 1 for “check best view automatically”.

```
app.iges.bestView = 1
```

app.iges.entityFlags1

Returns and sets the IGES options as integer. Please use this formula to calculate the allowed values:

value = 2^x

x	Option
0	“100 Circular Arc“
1	“102 Composite Curve“

2	“104 Conic Arc“
3	“106 Copious Data“
4	“108 Plane“
5	“110 Line“
6	“112 Parametric Spline Curve“
7	“114 Parametric Spline Surface“
8	“116 Point“
9	“118 Ruled Surface“
10	“120 Surface of Revolution“
11	“122 Tabulated Cylinder“
12	“124 Transformation Matrix“
13	“126 Rational B-Spline Curve“
14	“128 Rational B-Spline Surface“
15	“130 Offset Curve“
16	“132 Connect Point“
17	“140 Offset Surface“
18	“141 Boundary“
19	“142 Curve on a Parametric Surface“
20	“143 Bounded Surface“
21	“144 Trimmed (Parametric) Surface“
22	“186 Manifold Solid B-Rep Object“
23	“202 Angular Dimension“
24	“206 Diameter Dimension“
25	“210 General Label“
26	“212 General codeblock“
27	“214 Leader/Wedge“
28	“216 Linear Dimension“
29	“218 Ordinate Dimension“
30	“220 Point Dimension“
31	“222 Radius Dimension“

Add values to activate different options.

```
#activating the Copius Data-option only
app.iges.entityFlags1 = 8

#activating the Composite Curve- (2^1=2),
#Copious Data-(2^3=8) and Lines-option (2^5=32)
app.iges.entityFlags1 =  2 + 8 + 32
```

app.iges.entityFlags2

Returns and sets the IGES options as integer. Please use this formula to calculate the allowed values:

value = 2^x

x	Option
0	“228 General Symbol“
1	“230 Sectioned Area“
2	“304 Line Font Definition“

3	“308 Subfigure Definition“
4	“312 Text Display“
5	“314 Color Definition“
6	“320 Network Subfigure Definition“
7	“402 Form 1, 7, 15 Group“
8	“402 Form 18 Flow“
9	“402 Form 9, 13, 16“
10	“406 Property/Drawing Size“
11	“408 Singular Subfigure Instance“
12	“412 Rectangle Array Subfigure Instance“
13	“414 Circular Array Subfigure Instance“
14	“416 External Reference“

Add values to activate different options.

```
#activatingthe General Symbol-option only
app.iges.entityFlags2 = 1

#activating the General Symbol- (2^0=1),
#Sectioned Area-(2^1=2) and Line Font Definition
#-option (2^2=4)
app.iges.entityFlags2 =  1 + 2 + 4
```

Export IGES

app.iges.platformout

Returns and sets the export platform as integer. Allowed values are 1 for “Mac”, 2 for “MS Win” and 3 for “Unix”.

```
app.iges.platformout = 2
```

app.iges.extension

Returns and sets the file extension as string.

```
app.iges.extension = "igs"
```

app.iges.standard

Returns and sets the standard header as integer. Allowed values are 1 for “CALS I Subset”, 2 for “CALS II Subset” and 3 for “VDAIS”.

```
app.iges.standard = 3
```

app.iges.sender

Returns and sets the sender information as string.

```
app.iges.sender = "ITEDO SOFTWARE"
```

app.iges.receiver

Returns and sets the receiver information as string.

```
app.iges.receiver = "ITEDO LLC"
```

app.iges.author

Returns and sets the author information as string.

```
app.iges.author = "MS"
```

app.iges.company

Returns and sets the company information as string.

```
app.iges.company = "ITEDO SOFTWARE GMBH"
```

app.iges.VDAIS.sender

Returns and sets the VDAIS Sendefirma information as string.

```
app.iges.vdais.sender = "ITEDO SOFTWARE GMBH"
```

app.iges.VDAIS.partner

Returns and sets the VDAIS Ansprechpartner information as string.

```
app.iges.vdais.partner
= "MS"
```

app.iges.VDAIS.tel

Returns and sets the VDAIS Telefon information as string.

```
app.iges.vdais.tel = "02242 - 92210"
```

app.iges.VDAIS.adresse

Returns and sets the VDAIS Adresse information as string.

```
app.iges.vdais.adresse = "D-53773 Hennef"
```

app.iges.VDAIS.system

Returns and sets the VDAIS Erzeugendes System information as string.

```
app.iges.vdais.system = "WIN2K"
```

app.iges.VDAIS.projekt

Returns and sets the VDAIS Projekt information as string.

```
app.iges.vdais.projekt
= "T 305 UHF"
```

app.iges.VDAIS.kennung

Returns and sets the VDAIS Projektkennung information as string.

```
app.iges.vdais.kennung = "4711-2"
```

app.iges.VDAIS.variante

Returns and sets the VDAIS Variante information as string.

```
app.iges.vdais.variante = "V666"
```

app.iges.VDAIS.conf

Returns and sets the VDAIS Vertraulichkeit information as string.

```
app.iges.vdais.conf = "Nur für den Dienstgebrauch"
```

app.iges.VDAIS.datum

Returns and sets the VDAIS Gültigkeitsdatum information as string.

```
app.iges.vdais.datum = "01 01 2005"
```

app.iges.VDAIS.receiver

Returns and sets the VDAIS Empfängerfirma information as string.

```
app.iges.vdais.receiver = "ITEDO LLC"
```

app.iges.VDAIS.recName

Returns and sets the VDAIS Empfängername/Abteilung information as string.

```
app.iges.vdais.recName = "DR"
```

app.iges.CALS1.identifier

Returns and sets the CALS1 Identifier information as string.

```
app.iges.CALS1.identifier = "11299874548"
```

app.iges.CALS1.description

Returns and sets the CALS1 Description information as string.

```
app.iges.CALS1.description = "Big Project"
```

app.iges.CALS2.creator

Returns and sets the CALS2 Creator information as string.

```
app.iges.CALS2.Creator = "HunchBack"
```

app.iges.CALS2.PartName

Returns and sets the CALS2 Part Name information as string.

```
app.iges.CALS2.partName = "EL4456:8876-1B"
```

app.iges.CALS2.drawingName

Returns and sets the CALS2 Drawing Name information as string.

```
app.iges.CALS2.drawingName = "Front Left"
```

app.iges.CALS2.description

Returns and sets the CALS2 Description information as string.

```
app.iges.CALS2.description = "put a onto b"
```

app.iges.CALS2.revision

Returns and sets the CALS2 Revision information as string.

```
app.iges.CALS2.revision = "445:7889:223211c"
```

app.iges.CALS2.sizeNumber

Returns and sets the CALS2 Size Number information as string.

```
app.iges.CALS2.sizeNumber = "76DD"
```

Import DWG

app.dwg.showDialog

Returns and sets if the DWG import dialog is shown as boolean.

```
app.dwg.showDialog = TRUE
```

app.dwg.scale

Returns and sets the scale as float. 1 stands for 100% and 0.1 for 10%.

```
#setting the scale to 37%
app.dwg.scale = 0.37
```

app.dwg.platformIn

Returns and sets the import platform as integer. Allowed values are 1 for “Mac”, 2 for “MS Win” and 3 for “Unix”.

```
app.dwg.platformIn = 3
```

app.dwg.unit

Returns and sets the the corresponding unit as integer. Allowed values are:

1	“mm”
2	“inch”
3	“foot”
4	“m”

```
app.dwg.unit = 1
```

app.dwg.polyAsElements

Returns and sets if polylines should be converted into lines and circles as boolean.

```
app.dwg.polyAsElements = FALSE
```

app.dwg.ignoreVarPolyWidth

Returns and sets if variable width of polylines should be ignored as boolean.

```
app.dwg.ignoreVarPolyWidth = TRUE
```

app.dwg.ignoreTextFactor

Returns and sets if the width factor of text elements should be ignored as boolean.

```
app.dwg.ignoreTextFactor = TRUE
```

app.dwg.ignoreHeight

Returns and sets if the height and elevation should be ignored as boolean.

```
app.dwg.ignoreHeight = FALSE
```

app.dwg.entities

Returns and sets the DWG options as integer. Please use this formula to calculate the allowed values:

value = 2^x

x	Option
0	“LINE“
1	“POINT“
2	“CIRCLE“
3	“SHAPE“
4	“ELLIPSE“
5	“SPLINE“
6	“TEXT“
7	“ARC“
8	“TRACE“
9	“SOLID“
10	“INSERT“
11	“ATTDEF“
12	“ATTRIB“
13	“POLYLINE“
14	“LINE3D“
15	“FACE3D“
16	“DIMENSION“
17	“RAY“
18	“XLINE“
19	“MTEXT“
20	“LEADER“
21	“MLINE“
22	“LWLINE“
23	“PROXY“
24	“HATCH“
25	“VIEWPORT“

Add values to activate different options.

```
#activating the TEXT-option only
app.dwg.entities = 64

#activating the CIRCLE- (2^2=4), SHAPE-
#(2^3=8) and ELLIPSE-option (2^4=16)
```

```
app.dwg.entities =  4 + 8 + 16
```

Export DWG

app.dwg.release

Returns and sets the AutoCAD release as integer. Allowed values are:

0	“Release 12”
1	“Release 13”
2	“Release 14”
3	“AutoCAD 2000”

```
app.dwg.release = 3
```

app.dwg.flags

Returns and sets the conversion flags as integer. Allowed values are 0 for “Convert ellipses and beziers to 2D polylines”, 1 for “Convert ellipses to 3D arcs and beziers to 2D polylines”, 2 for “Convert ellipses to 2D Polylines and beziers to 3D Polylines” and 3 for “Convert ellipses and beziers to 3D”. **Conversion of ellipses (2 and 3) is only valid for “Release 12”!**

```
app.dwg.flags = 2
```

app.dwg.platformOut

Returns and sets the export platform as integer. Allowed values are 1 for “Mac”, 2 for “MS Win” and 3 for “Unix”.

```
app.dwg.platformOut = 3
```

app.dwg.extension

Returns and sets file extension as string.

```
app.dwg.extension = "dwg"
```

Import DXF

app.dxf.showDialog

Returns and sets if the DXF import dialog is shown as boolean.

```
app.dxf.showDialog = TRUE
```

app.dxf.scale

Returns and sets the scale as float. 1 stands for 100% and 0.1 for 10%.

```
#setting the scale to 37%
app.dxf.scale = 0.37
```

app.dxf.platformIn

Returns and sets the import platform as integer. Allowed values are 1 for “Mac”, 2 for “MS Win” and 3 for “Unix”.

```
app.dxf.platformIn = 3
```

app.dxf.unit

Returns and sets the corresponding unit as integer. Allowed values are:

1	“mm”
2	“inch”
3	“foot”
4	“m”

```
app.dxf.unit = 1
```

app.dxf.polyAsElements

Returns and sets if polylines should be converted into lines and circles as boolean.

```
app.dxf.polyAsElements = FALSE
```

app.dxf.ignoreVarPolyWidth

Returns and sets if variable width of polylines should be ignored as boolean.

```
app.dxf.ignoreVarPolyWidth = TRUE
```

app.dxf.ignoreTextFactor

Returns and sets if the width factor of text elements should be ignored as boolean.

```
app.dxf.ignoreTextFactor = TRUE
```

app.dxf.ignoreHeight

Returns and sets if the height and elevation should be ignored as boolean.

```
app.dxf.ignoreHeight = FALSE
```

app.dxf.entities

Returns and sets the DXF options as integer. Please use this formula to calculate the allowed values:

value = 2^x

x	**Option**
0	“LINE“
1	“POINT“
2	“CIRCLE“
3	“SHAPE“
4	“ELLIPSE“
5	“SPLINE“
6	“TEXT“
7	“ARC“

8	“TRACE“
9	“SOLID“
10	“INSERT“
11	“ATTDEF“
12	“ATTRIB“
13	“POLYLINE“
14	“LINE3D“
15	“FACE3D“
16	“DIMENSION“
17	“RAY“
18	“XLINE“
19	“MTEXT”
20	“LEADER“
21	“MLINE“
22	“LWLINE“
23	“PROXY“
24	“HATCH“
25	“VIEWPORT“

Add values to activate different options.

```
#activating the TEXT-option only
app.dxf.entities = 64
```

```
#activating the CIRCLE- (2^2=4), SHAPE-
#(2^3=8) and ELLIPSE-option (2^4=16)
app.dxf.entities =  4 + 8 + 16
```

Export DXF

app.dxf.release

Returns and sets the AutoCAD release as integer. Allowed values are:

1	“Release”
2	“Release 13”
3	“Release 14”
4	“AutoCAD 2000”

```
app.dxf.release = 4
```

app.dxf.flags

Returns and sets the conversion flags as integer. Allowed values are 0 for “Convert ellipses and beziers to 2D polylines”, 1 for “Convert ellipses to 3D arcs and beziers to 2D polylines”, 2 for “Convert ellipses to 2D Polylines and beziers to 3D Polylines” and 3 for “Convert ellipses and beziers to 3D”.

Note

Conversion of ellipses (2 and 3) is only valid for “Release 12”!

```
app.dxf.flags = 2
```

app.dxf.platformOut

Returns and sets the export platform as integer. Allowed values are 1 for “Mac”, 2 for “MS Win” and 3 for “Unix”.

```
app.dxf.platformOut = 3
```

app.dxf.extension

Returns and sets file extension as string.

```
app.dxf.extension = "dxf"
```

DXF/DWG Import Options: app.dxf / app.dwg

Applies to Arbortext IsoDraw CADprocess 7.1 F000 and later.

The **DXF Options** dialog box below shows the new DXF and DWG entites that IML can import.

app.dxf.entities / app.dwg.entities

Bit	Option	Available in Release
26	TOLERANCE	7.0; 7.1
27	SURFACE	7.1
28	BODY	7.1
29	SOLID 3D	7.1
30	POLYGON MESH	7.1
31	REGION	7.1

app.dxf.entityFlags2 / app.dwg.entityFlags2

Bit	Option	Available in Release
0	HELIX	7.1

Import VRML

app.vrml.showDialog

Returns and sets if the VRML import dialog is shown as boolean.

```
app.vrml.showDialog = TRUE
```

app.vrml.scale

Returns and sets the scale as float. 1 stands for 100% and 0.1 for 10%.

```
#setting the scale to 37%
app.vrml.scale = 0.37
```

app.vrml.selectGroups

Returns and sets if assemblies should be selected as boolean.

```
app.vrml.selectGroups = FALSE
```

app.vrml.createInfo

Returns and sets if the object info for assemblies should be created as boolean.

```
app.vrml.createInfo = FALSE
```

app.vrml.hsType

Returns and sets the hotspot type as integer. Allowed values are 0 for “no Hotspot” and 1 for “lines of object”.

```
app.vrml.hsType = 1
```

app.vrml.readflag

Returns and sets if the buffers should be read for all file sizes as integer. Allowed values are 1 for “off” and 17 for “create reading buffer for all file sizes”.

```
app.vrml.readflag = 17
```

app.vrml.entities

Returns and sets the VRML options as integer. Please use this formula to calculate the allowed values:

value = 2^x

x	Option
0	“Box“
1	“Cube“
2	“Cone“
3	“Cylinder“
4	“Sphere“
5	“Elevation Grid“
6	“Point Set“
7	“Indexed Line Set“
8	“Indexed Face Set“
9	“Extrusion“

Add values to activate different options.

```
#activating the Cone-option only
app.vrml.entities = 4

#activating the Cone- (2^2=4), Cylinder-
#(2^3=8) and Sphere-option (2^4=16)
```

```
app.vrml.entities =  4 + 8 + 16
```

app.vrml.viewpointFlag

Returns and sets the VRML Viewpoints option as integer. Use 0 to not use viewpoints, use 1 to Use viewpoints; Select from list and use 3 to Use viewpoints; Predefined viewpoint.

```
app.vrml.viewpointFlag = 0
```

app.vrml.viewpointPredef

Returns and sets the Predefined viewpoint option as string. The option to Use viewpoints; Predefined viewpoint must be selected for this setting to take affect.

```
app.vrml.viewpointPredef = "test"
```

Import Wavefront

app.wavefront.showDialog

Returns and sets if the Wavefront import dialog is shown as boolean.

```
app.wavefront.showDialog = TRUE
```

app.wavefront.scale

Returns and sets the scale as float. 1 stands for 100% and 0.1 for 10%.

```
#setting the scale to 13%
app.wavefront.scale = 0.13
```

app.wavefront.selectGroups

Returns and sets if assemblies should be selected as boolean.

```
app.wavefront.selectGroups = FALSE
```

app.wavefront.createInfo

Returns and sets if the object info for assemblies should be created as boolean.

```
app.wavefront.createInfo = FALSE
```

app.wavefront.hsType

Returns and sets the hotspot type as integer. Allowed values are 0 for “no Hotspot” and 1 for “lines of object”.

```
app.wavefront.hsType = 1
```

app.wavefront.readflag

Returns and sets if the buffers should be read for all file sizes as integer. Allowed values are 1 for “off” and 17 for “create reading buffer for all file sizes”.

```
app.wavefront.readflag = 17
```

Import with ProductView Adapters

Applies to Arbortext IsoDraw CADprocess 7.1 F000 and later.

app.adapterCount

Returns and sets the number of successfully installed ProductView (PV) adapters in Arbortext IsoDraw CADprocess as integer. This property is read-only.

app.adapters[*"name"* | "*index*"]

The properties of a ProductView (PV) adapter are accessed by PV adapter *name* or *index* number. The name of the folder that contains the PV adapter preferences file, `pvad_xlte.prf`, is the same as the PV adapter *name*. You cannot assign the PV adapter itself to a variable, but you can use the `exists()` function to verify that it is installed and configured correctly.

Example

```
# Examples:

MACRO configureJTadapter
# Overwrite PV Adapter file configuration of JT adapter
 IF ( exists( app.adapters["JT"] ) )
  #  only if adapter is installed...
  app.adapters["JT"].showDialog = false
  app.adapters["JT"].selectGroups = false
  app.adapters["JT"].createInfo = true
  app.adapters["JT"].hsType = 1
  app.adapters["JT"].scale = 0.1
 END IF
END MACRO

MACRO ListAdapters
# Write list of installed adapters to external file
 DEFINE n AS Integer
 DEFINE i AS Integer
 DEFINE sOut AS String

 sOut = "D:\temp\AdaptersList.txt"
 FNew sOut
 i = app.adapterCount
 WHILE (i>0)
  FWRITE sOut "-------------------------------------------------------------------------------
  FWRITE sOut "Name  : "+app.adapters[i].name
  FWRITE sOut "Text  : "+app.adapters[i].descriptiveText
  FWRITE sOut "Extension : "+app.adapters[i].Extension
  FWRITE sOut "Path  : "+app.adapters[i].path
  FWRITE sOut "Executable : "+app.adapters[i].executable
  FWRITE sOut "Recipe  : "+app.adapters[i].recipe
  i = i-1
 END WHILE
END MACRO
```

app.adapters[ ].showDialog

Returns and sets if the **Import** dialog box for the PV adapter is shown. Type is boolean.

```
app.adapters["JT"].showDialog = true
```

app.adapters[ ].selectGroups

Returns and sets if the assemblies should be selected for this PV adapter as boolean.

```
app.adapters["JT"].selectGroups = false
```

app.adapters[ ].createInfo

Returns and sets if the object information for the assemblies should be created for this PV adapter as boolean.

```
app.adapters["JT"].createInfo = false
```

app.adapters[ ].hsType

Returns and sets which hotspot type is created for the assemblies for this PV adapter as boolean.

0	no hotspot
1	lines of object

```
app.adapters["JT"].hsType = 1
```

app.adapters[ ].scale

Returns and sets the import scale for this adapter as float. `1` stands for 100% and `0.1` for 10%.

```
app.adapters["JT"].scale = 0.5
```

app.adapters[].descriptiveText

Returns the file extensions for this adapter as string. This property is read-only.

```
MESSAGE app.adapters[1].extension
```

app.adapters[].extension

Returns the list of file extensions for this adapter as string with a maximum length of 80 characters. This property is read-only

```
MESSAGE app.adapters[1].extension
```

app.adapters[].name

Returns the name for this adapter as string. The name of the adapter is the same as the folder name where the file `pvad_xlte.prf` is installed.

```
MESSAGE app.adapters[1].name
```

app.adapters[].path

Returns the installation path for this adapter as string. This property is read-only.

```
MESSAGE app.adapters[1].path
```

app.adapters[].executable

Returns the path and filename of the adapters executable as string. This property is read-only.

```
MESSAGE getFileName(app.adapters[1].executable)
```

app.adapters[].recipe

Returns the path and name of the adapters recipe file as string. This property is read-only.

```
MESSAGE getFileName(app.adapters[1].recipe)
```

Export HPGL

app.hpgl.extension

Returns and sets the file extension as string.

```
app.hpgl.extension = "hpl"
```

app.hpgl.sortPens

Returns and sets if the pens should be sorted for the plotter output as boolean.

```
app.hpgl.sortPens = TRUE
```

app.hpgl.refPoint.x

Returns and sets the reference point x-coord as float.

```
app.hpgl.refPoint.x = 10.1
```

app.hpgl.refPoint.y

Returns and sets the reference point y-coord as float.

```
app.hpgl.refPoint.y = 10.2
```

Export TIFF

app.tiff.extension

Returns and sets the file extension as string.

```
app.tiff.extension = "tif"
```

app.tiff.resolution

Returns and sets the resolution as float.

```
app.tiff.resolution = 200
```

app.tiff.border

Returns and sets the border thickness as float.

```
app.tiff.border = 2
```

app.tiff.depth

Returns the color depth as integer. This property is read only. (Returned values could be 1, 8, and 24 for 1, 8, and 24 bit)

```
MESSAGE app.tiff.depth
```

app.tiff.kind

Returns and sets the color depth as string. Allowed values are “bitmap”, “grayscale”, “color8” and “color24”.

```
app.tiff.kind = "color24"
```

app.tiff.page

Returns and sets the dimension as integer. Allowed values are 1 for “Drawing size” and 2 for “Extent”.

```
app.tiff.page = 2
```

app.tiff.compression

Returns and sets the compression as integer. Allowed values are 1 for “none”, 7 for “PackBits”, for “CCITT Group3”, 4 for “CCITT Group4”, 5 for “LZW” and 6 for “LZW2”.

```
app.tiff.compression = 2
```

app.tiff.intelOrder

Returns and sets the byte order as boolean.

```
app.tiff.intelOrder = TRUE
```

Export JPEG

app.jpeg.extension

Returns and sets the file extension as string.

```
app.jpeg.extension = "jpg"
```

app.jpeg.resolution

Returns and sets the resolution as float.

```
app.jpeg.resolution = 200
```

app.jpeg.border

Returns and sets the border thickness as float.

```
app.jpeg.border = 2
```

app.jpeg.kind

Returns and sets the color depth as string. Allowed values are “grayscale” and “color24”.

```
app.jpeg.kind = "color24"
```

app.jpeg.page

Returns and sets the dimension as integer. Allowed values are 1 for “Drawing size” and 2 for “Extent”.

```
app.jpeg.page = 2
```

app.jpeg.quality

Returns and sets the export quality as float. Allowed values are between 1 and 100. A value to 40 would represent 4 on the quality scale.

```
app.jpeg.quality = 40
```

Export PNG

app.png.extension

Returns and sets the file extension as string.

```
app.png.extension = "png"
```

app.png.resolution

Returns and sets the resolution as float.

```
app.png.resolution = 200
```

app.png.border

Returns and sets the border thickness as float.

```
app.png.border = 2
```

app.png.kind

Returns and sets the color depth as string. Allowed values are “bitmap”, “grayscale”, “color8” and “color32”.

```
app.png.kind = "color32"
```

app.png.page

Returns and sets the dimension as integer. Allowed values are 1 for “Drawing size” and 2 for “Extent”.

```
app.png.page = 2
```

app.png.filter

Returns and sets the export filter as float. Allowed values are:

0	“None”
1	“Sub”
2	“Up”
3	“Average”
4	“Paeth”
5	“Adaptive”

```
app.png.filter = 3
```

Export BMP

app.bmp.extension

Returns and sets the file extension as string.

```
app.bmp.extension = "bmp"
```

app.bmp.resolution

Returns and sets the resolution as float.

```
app.bmp.resolution = 200
```

app.bmp.border

Returns and sets the border thickness as float.

```
app.bmp.border = 2
```

app.bmp.kind

Returns and sets the color depth as string. Allowed values are “bitmap”, “grayscale”, “color8” and “color24”.

```
app.bmp.kind = "color24"
```

app.bmp.page

Returns and sets the dimension as integer. Allowed values are 1 for “Drawing size” and 2 for “Extent”.

```
app.bmp.page = 2
```

app.bmp.version

Returns and sets the version as integer. Allowed values are 1 for “Windows” and 2 for “OS/2”.

```
app.bmp.version = 1
```

app.bmp.compression

Returns and sets the compression as integer. Allowed values are 0 for “none” and 1 for “RLE 8”.

```
app.bmp.compression = 1
```

Export PCX

app.pcx.extension

Returns and sets the file extension as string.

```
app.pcx.extension = "pcx"
```

app.pcx.resolution

Returns and sets the resolution as float.

```
app.pcx.resolution = 200
```

app.pcx.border

Returns and sets the border thickness as float.

```
app.pcx.border = 2
```

app.pcx.kind

Returns and sets the color depth as string. Allowed values are “bitmap”, “grayscale”, “color8” and “color24”.

```
app.pcx.kind = "color24"
```

app.pcx.page

Returns and sets the dimension as integer. Allowed values are 1 for “Drawing size” and 2 for “Extent”.

```
app.pcx.page = 2
```

Returns and sets the compression as integer. Allowed values are 1 for “none” and 2 for “RLE 8”.

```
app.pcx.encoding = 1
```

app.pcx.encoding

Export CALS

app.cals.extension

Returns and sets the file extension as string.

```
app.cals.extension = "cc4"
```

app.cals.resolution

Returns and sets the resolution as float.

```
app.cals.resolution = 200
```

app.cals.border

Returns and sets the border thickness as float.

```
app.cals.border = 2
```

app.cals.page

Returns and sets the dimension as integer. Allowed values are 1 for “Drawing size” and 2 for “Extent”.

```
app.cals.page = 2
```

app.cals.srcdocid

Returns and sets the srcdocid information as string.

```
app.cals.srcdocid = "sample"
```

app.cals.dstdocid

Returns and sets the dstdocid information as string.

```
app.cals.dstdocid = "sample"
```

app.cals.textfilid

Returns and sets the textfilid information as string.

```
app.cals.textfilid = "sample"
```

app.cals.figid

Returns and sets the figid information as string.

```
app.cals.figid = "sample"
```

app.cals.srcgph

Returns and sets the srcgph information as string.

```
app.cals.srcgph = "sample"
```

app.cals.doccls

Returns and sets the doccls information as string.

```
app.cals.doccls = "sample"
```

Returns and sets the codeblocks information as string.

```
app.cals.codeblocks = "sample"
```

app.cals.codeblocks

Export Text

app.txt.extension

Returns and sets the file extension as string.

```
app.txt.extension = "txt"
```

app.txt.useUnicode

Returns and sets if Arbortext IsoDraw should generate a unicode text file as boolean.

```
app.txt.useUnicode = TRUE
```

Export Object List

app.objectList.extension

Returns and sets the file extension as string.

```
app.objectList.extension = "txt"
```

app.objectList.fileTag

Returns and sets the file element as string.

```
app.objectList.fileTag = "MyFile"
```

app.objectList.objTag

Returns and sets the object element as string.

```
app.objectList.objTag = "MyObject"
```

app.objectList.layerTag

Returns and sets the layer element as string.

```
app.objectList.layerTag = "MyLayer"
```

app.objectList.flags

Returns and sets the object list flags as integer. Allowed values are:

1	“Write as XML”
2	“ID”
4	“Name”
8	“Object tip”
16	“Other attributes”
32	“XML attributes”
64	“XML: obj-type as element name”

Add values to combine different flags.

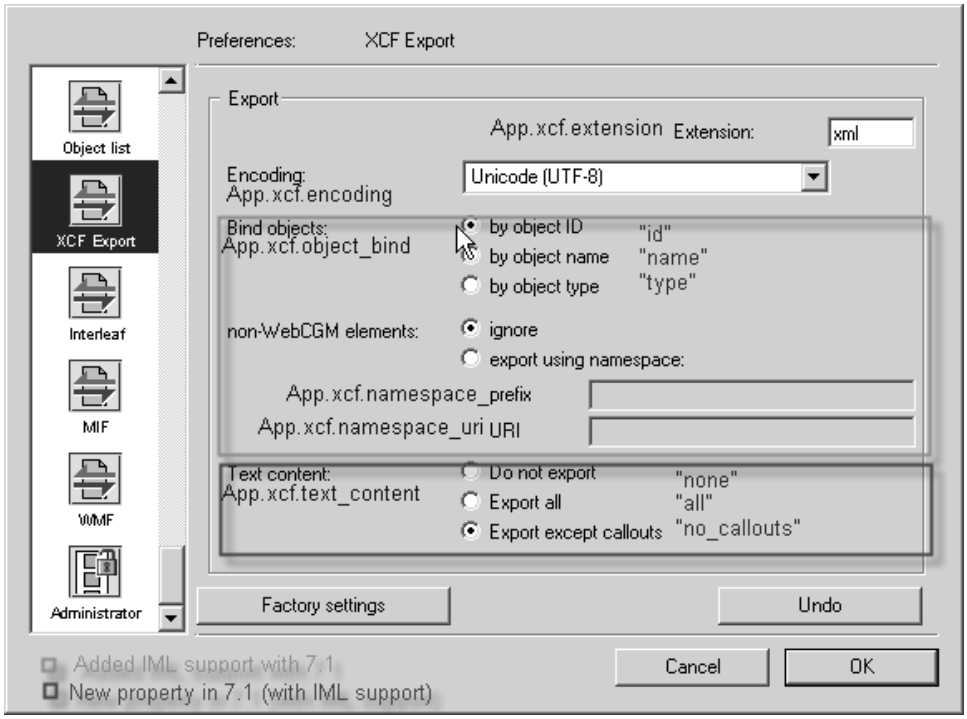
```
#enable all attributes
app.objectList.flags = (1+2+4+8+16+32+64)
```

Export XCF

Applies to Arbortext IsoDraw CADprocess 7.1 F000 and later.

app.xcf

The **XCF Export** preferences dialog box below shows the XCF export settings corresponding to the IML `app.xcf` properties below.



app.xcf.object_bind

Returns and sets the **Bind objects** setting as string. Allowed values are:

"id"	for **by Object ID**
"name"	for **by object name**
"type"	for **by object type**

app.xcf.namespace_prefix

Returns and sets the XCF namespace **prefix** value as string with a maximum length of 256 characters. If `app.xcf.namespace_prefix` and `app.xcf.namespace_uri` are both empty, namespaces are ignored.

app.xcf.namespace_uri

Returns and sets the XCF namespace **uri** value as string with a maximum length of 2048 characters. If `app.xcf.namespace_uri` and `app.xcf.namespace_prefix` are both empty, namespaces are ignored.

app.xcf.text_content

Returns and sets the **Text content** setting as string. Allowed values are:

"none"	for **Do not export**
"all"	for **Export all**
"no_callouts"	for **Export except callouts**

app.version.title

Returns the licensed version of Arbortext IsoDraw as displayed in the title of the main application window as string. This property is read-only. It returns either "`IsoDraw Foundation`" or "`IsoDraw CADprocess`".

Export Interleaf

app.interleaf.extension

Returns and sets the file extension as string.

```
app.interleaf.extension = "doc"
```

app.interleaf.useStyles

Returns and sets if linestyles should be converted into single elements as boolean.

```
app.interleaf.useStyles = TRUE
```

Export MIF

app.mif.extension

Returns and sets the file extension as string.

```
app.mif.extension = "mif"
```

Export PICT

The PICT format is a format which could only be used on a Macintosh compatible computer.

app.pict.extension

Returns and sets the file extension as string.

```
app.pict.extension = "pic"
```

Export PDF

Applies to Arbortext IsoDraw CADprocess 7.1 F000 and later.

app.pdf

Returns and sets the preferences of the **PDF Export** preference panel and the **PDF options** dialog box.

Example

```
# Examples:

MACRO Log_PDFU3D_Prefs
# writes current preference settings to macro logfile
 Log "App.u3d.Extension         = "+ App.u3d.Extension
 Log "App.pdf.Extension         = "+ App.pdf.Extension
 Log "App.pdf.text_to_path = "+ App.pdf.text_to_path
 Log "App.pdf.convert_to_raster = "+ App.pdf.convert_to_raster
 Log "App.pdf.meta.title         = "+ App.pdf.meta.title
 Log "App.pdf.meta.subject = "+ App.pdf.meta.subject
 Log "App.pdf.meta.author = "+ App.pdf.meta.author
 Log "App.pdf.meta.keywords = "+ App.pdf.meta.keywords
 Log "App.pdf.raster.resolution = "+ App.pdf.raster.resolution
 Log "App.pdf.raster.border = "+ App.pdf.raster.border
 Log "App.pdf.raster.page = "+ App.pdf.raster.page
 Log "App.pdf.raster.kind = "+ App.pdf.raster.kind
END MACRO

MACRO PDF_Set_MetaDescription
 App.pdf.meta.author  = sys.user.name
 App.pdf.meta.subject = stripext(activeDoc.name)
END MACRO

MACRO PDF_Prefs_Factory_Setting
# recorded with Arbortext IsoDraw 7.1
 App.pdf.Extension = "pdf"
 App.pdf.text_to_path = false
 App.pdf.convert_to_raster = false
 App.pdf.meta.title = ""
 App.pdf.meta.author = ""
 App.pdf.meta.subject = ""
 App.pdf.meta.keywords = ""
 App.pdf.raster.resolution = 200
 App.pdf.raster.border = 2
 App.pdf.raster.page = 2
 App.pdf.raster.kind = "color24"
END MACRO
```

app.pdf.extension

Returns and sets the file extension as string with a maximum length of 8 characters.

app.pdf.text_to_path

Returns and sets if text elements are exported to Bézier-paths as boolean.

app.pdf.convert_to_raster

Returns and sets if the content is exported as raster as boolean.

app.pdf.meta

Returns and sets PDF metadata field values.

app.pdf.meta.title

Returns and sets the PDF title metadata field. Semicolon character (";") is not allowed. Type is string; maximum length is 256 characters.

app.pdf.meta.author

Returns and sets the PDF author metadata field. Semicolon character (";") is not allowed. Type is string; maximum length is 256 characters.

app.pdf.meta.subject

Returns and sets the PDF subject metadata field. Semicolon character (";") is not allowed. Type is string; maximum length is 256 characters.

app.pdf.meta.keywords

Returns and sets the PDF keywords metadata field. Semicolon character (";") is not allowed. Type is string; maximum length is 256 characters.

app.pdf.raster

Returns and sets PDF raster image properties.

app.pdf.raster.resolution

Returns and sets the resolution for included raster in dpi as float.

app.pdf.raster.border

Returns and sets the border for included raster in mm as float.

app.pdf.raster.kind

Returns and sets the image depth for included raster as string.

Allowed values are: "`bitmap`": black & white, "grayscale": 8 bit gray tone, "color8": 8 bit color, "color24": 24 bit color.

app.pdf.raster.page

Type is integer. Returns and sets if the dimensions as integer.

Allowed values are: 1: use full page size; 2: use illustration extent.

Export U3D

Applies to Arbortext IsoDraw CADprocess 7.1 F000 and later.

app.u3d.extension

Returns and sets the file extension as string with a maximum length of 8 characters.

```
app.u3d.extension = "u3d"
```

Import WMF

The WMF format is a format which could only be used on a Windows compatible computer.

app.wmf.showDialog

Returns and sets if the wmf import dialog is shown as boolean.

```
app.wmf.showDialog = TRUE
```

app.wmf.asRaster

Returns and sets if the wmf file should be imported as image element as boolean.

```
app.wmf.asRaster = FALSE
```

Export WMF

The WMF format is a format which could only be used on a Windows compatible computer.

app.wmf.extension

Returns and sets the file extension as string.

```
app.wmf.extension = "wmf"
```

app.wmf.enhanced

Returns and sets if the file should be written as enhanced metafile as boolean.

```
app.wmf.enhanced = TRUE
```

24

Sub Data Types for Attribute Preferences

IML provides several “sub data types” that you do not `DEFINE` directly (as you do for other complex data types). Rather, you use sub data types to return or set attribute preferences for the currently active Document or Application object.

.Pens[ ]

This property gives access to all pen attributes but it can not be used directly.

.Pens[ ].name

Returns and sets the name of a specific default pen as string.

```
app.Pens["Thick"].name = "myThickPen"
```

.Pens[ ].width

Returns and sets the width of a specific default pen as float.

```
app.Pens[1].width = 2.5
```

.Pens[ ].color

This is a ColorSpec (see ColorSpec on page) type property.

```
MESSAGE app.Pens["Medium"].color.type
```

.Pens[ ].style

Returns and sets the style of a specific pen as string.

```
app.Pens["Thick"].style = "Solid"
```

.Pens[ ].shadow

Returns and sets the type of shadow as string.

```
app.Pens[1].shadow = "Autom. Long"
```

.Pens[ ].switchPen

Returns and sets the type of pen switched to as string.

```
app.Pens[1].switchPen = "Thin"
```

.Pens[ ].screenColor

Returns and sets the screen color as RGBColor.

```
app.Pens[3].screenColor.red = 122
app.Pens[3].screenColor.green = 122
app.Pens[3].screenColor.blue = 122
```

.linestyles[ ]

This property gives access to all style attributes but it can not be used directly.

.linestyles[].name

Returns and sets the name of the linestyle as string.

```
app.linestyles[1].name = "myLineStyle"
```

.linestyles[].type

Returns and sets the type of the linestyle as integer. The allowed values are 0 for “solid”, 1 for “dashed” and 2 for “dotted”.

```
app.linestyles[1].type = 2
```

.linestyles[ ].startMark

Returns and sets the start mark of the linestyle as integer between 0 and 46 (0 = no mark).

```
app.linestyles[1].startMark = 33
```

.linestyles[ ].endMark

Returns and sets the end mark of the linestyle as integer between 0 and 46 (0 = no mark).

```
app.linestyles[1].endMark = 7
```

.linestyles[ ].minEndLength

Returns and sets the minimum end length of the linestyle as float.

```
app.linestyles[1].minEndLength = 33
```

.linestyles[ ].pattern[ ]

Returns and sets the pattern values.

```
app.linestyles[1].pattern[1]
```

Returns and sets the pattern values as float. This applies to linestyles.type 1 and 2. If the linestyle.type property is set to "dotted" and the first pattern is greater than 0 the first pattern defines the point distance. If the linestyle.type property is set to "dotted" and the first pattern is equal to (-256) the point distance is set to automatic.

```
app.linestyles[1].pattern[1] = 1
app.linestyles[1].pattern[2] = 1.5
```

The linestyles can be addressed by the language independent specifier (see International Names on page 299):

```
app.linestyles[“$ISO_DOTTED”].pattern[1] = 1.5
```

.shadows[ ]

This property gives access to all shadow attributes but it can not be used directly.

.shadows[ ].name

Returns and sets the name of the shadow as string.

```
app.shadows[1].name = "myPrettyShadow"
```

.shadows[ ].width

Returns and sets the width of the shadow in mm as float. Setting this value to 0 means automatic; all positive values defining the thickness of the shadow in mm; all negative values defining the thickness of the shadow relative to the thickness of the line thickness.

```
app.shadows[1].width = 7
```

.shadows[ ].color

Returns and sets the color of the shadow as ColorSpec on page .

```
app.shadows[1].color.rgb.red = 155
app.shadows[1].color.rgb.green = 155
app.shadows[1].color.rgb.blue = 155
```

.shadows[ ].start

Returns and sets the start type of the shadow as integer. Allowed values are 1 for “long”, 2 for “middle”, 3 for “short”.

```
app.shadows[1].start = 1
```

.shadows[ ].end

Returns and sets the end type of the shadow as integer. Allowed values are 1 for “long”, 2 for “middle”, 3 for “short”.

```
app.shadows[1].end = 2
```

.colors[ ]

This property gives access to all color attributes but it can not be used directly.

.colors[ ].name

Returns and sets the name of the color as string.

```
app.colors[1].name = "myLovelyBlue"
```

.colors[ ].kind

Returns and sets the type of the color as string. The allowed values are "Custom_color", "RGB_color". "CMYK_color" and "Color_Tone".

```
app.colors[1].kind = "RGB_color"
```

.colors[ ].color

Returns and sets the color as ColorSpec (see ColorSpec on page).

```
app.colors[1].color.rgb.red = 188
app.colors[1].color.rgb.green = 188
app.colors[1].color.rgb.blue = 188
```

.hatchings[ ]

This property gives access to one hatching attribute but it can not be used directly.

.hatchings[ ].name

Returns and sets the name of the hatching as string.

```
app.hatchings[1].name = "firstHatch"
```

.patterns[ ]

This property gives access to one pattern attribute but it can not be used directly.

.patterns[ ].name

Returns and sets the name of the hatching as string.

```
app.patterns[1].name = "firstPatch"
```

.txtFormats[ ]

This property gives access to all text format attributes but it can not be used directly.

.txtFormats[ ].name

Returns and sets the name of the text format as string.

```
app.txtFormats[1].name = "My New Format"
```

.txtFormats[ ].font

Returns and sets the name of the font of the text format as string. Any installed font can be used.

```
app.txtFormats[1].font = "Arial"
```

.txtFormats[ ].face

Returns and sets the text face of the text format as string. Allowed values are “normal”, “bold”, “italic” and “bolditallic”.

```
app.txtFormats[1].face = "bold"
```

.txtFormats[ ].size

Returns and sets the size of the text format as float.

```
app.txtFormats[1].size = 12
```

.txtFormats[ ].leading

Returns and sets the leading of the text format as float.

```
app.txtFormats[1].leading = 2.5
```

.txtFormats[ ].position

Returns and sets the position of the text format as float.

```
app.txtFormats[1].position = 8
```

.txtFormats[ ].kerning

Returns and sets the kerning of the text format as float.

```
app.txtFormats[1].kerning = 1.2
```

.callouts[ ]

This property gives access to all callout attributes but it can not be used directly.

.callouts[ ].style_name

Returns and sets the name of the callout style as string.

```
app.callouts[1].style_name = "myNewCallout"
```

.callouts[ ].shape_type

Returns and sets the type of the callout shape type as integer. Allowed values are 0 for "None", 1 for "Circle", 2 for "Triangle Up", 3 for "Triangle Down", 4 for "Rectangle", 5 for "Pentagon" and 6 for "Hexagon".

```
app.callouts[1].shape_type = 2
```

.callouts[ ].shape_width

Returns and sets the value of the callout shape width as float.

```
app.callouts[1].shape_width = 2.5
```

.callouts[ ].shape_height

Returns and sets the value of the callout shape height as float.

```
app.callouts[1].shape_height = 1.5
```

.callouts[ ].shape_pen

Returns and sets the pen of the shape of the callout as string. Any defined pen can be used.

```
app.callouts[1].shape_pen = "Thick"
```

.callouts[ ].shape_style

Returns and sets the style of the shape of the callout as string. Any defined style can be used.

```
app.callouts[1].shape_style = "Dashed"
```

.callouts[ ].shape_shadow

Returns and sets the shadow of the shape of the callout as string. Any defined shadow can be used.

```
app.callouts[1].shape_shadow = "Autom. Long"
```

.callouts[ ].text_update

Returns and sets the status of the text update width as string. The allowed values are "none" and "auto".

```
app.callouts[1].text_update = "auto"
```

.callouts[ ].text_position

Returns and sets the type of the text alignment as string. The allowed values are "aligned" and "centered".

```
app.callouts[1].text_position = "centered"
```

.callouts[ ].text_prefix

Returns and sets the text prefix as string.

```
app.callouts[1].text_prefix = "partNo:"
```

.callouts[ ].text_postfix

Returns and sets the text postfix as string.

```
app.callouts[1].text_postfix = "_66765"
```

.callouts[ ].text_gap

Returns and sets the text gap as float.

```
app.callouts[1].text_gap = "1.5"
```

.callouts[ ].line_pen

Returns and sets the line pen as string. Any existing pen can be used.

```
app.callouts[1].line_pen
= "Thick"
```

.callouts[ ].line_style

Returns and sets the line style as string. Any existing style can be used.

```
app.callouts[1].line_style = "Solid"
```

.callouts[ ].line_shadow

Returns and sets the line shadow as string. Any existing shadow can be used.

```
app.callouts[1].line_shadow = "Autom. Long"
```

.callouts[ ].fill

Returns and sets the callout fill as fill.

```
app.callouts[1].fill.type = "no_fill"
```

.callouts[ ].text_format

Returns and sets the text format of the callout as string. Any defined format can be used.

```
app.callouts[1].text_format = "Normal"
```

.callouts[ ].text_font

Returns and sets the text font of the callout as string. Any installed font can be used.

```
app.callouts[1].text_font = "Arial"
```

.callouts[ ].text_face

Returns and sets the text face of the callout as string. Allowed values are "normal", "bold", "italic" and "bolditalic".

```
app.callouts[1].text_face = "bold"
```

.callouts[ ].text_size

Returns and sets the text size of the callout as float.

```
app.callouts[1].text_size = 24.5
```

.callouts[ ].text_strokecolor

Returns and sets the text stroke color of the callout as ColorSpec (see ColorSpec on page).

```
app.callouts[1].text_strokecolor.type = "cmykValues"
```

.callouts[ ].text_stroke

Returns and sets the text stroke of the callout as float.

```
app.callouts[1].text_stroke = 2.2
```

.callouts[ ].text_fillcolor

Returns and sets the color of the fill of the text of the callout as ColorSpec (see ColorSpec on page).

```
app.callouts[1].text_fillcolor.rgb.red = 168
app.callouts[1].text_fillcolor.rgb.green = 168
app.callouts[1].text_fillcolor.rgb.blue = 168
```

.callouts[ ].text_scheme

Returns and sets the text scheme of the callout as string. Allowed values are "no_scheme", "alpha_uppercase", "alpha_lowercase" and "numeric".

```
app.callouts[1].text_scheme = "no_scheme"
```

.callouts[ ].text_hotspot_flag

Returns and sets if the hotspot flag is set as boolean.

```
app.callouts[1].text_hotspot_flag = false
```

.grids[ ]

This property gives access to all grid attributes but it can not be used directly.

.grids[ ].name

Returns and sets the name of the grid as string.

```
app.grids[1].name = "Isometric"
```

.grids[ ].xAngle

Returns and sets the x-angle of the grid as float.

```
app.grids[1].xAngle = 30
```

.grids[ ].zAngle

Returns and sets the z-angle of the grid as float.

```
app.grids[1].zAngle = 30
```

VII

Appendix

25 International Names

Writing Language-Independent Macros

In order to develop macros that are independent from the language version of Arbortext IsoDraw, use the international attribute-names.

Using these names rather than the language specific names will enable these macros to run in Arbortext IsoDraw disregarding the language version.

Even if it is valid to switch the active pen with:

```
myDoc.active_pen = "No Pen"
```

This would switch the pen only if the Pen is really named "No Pen". This is true for the English language version of Arbortext IsoDraw only. In the German version the appropriate command would be:

```
myDoc.active_pen = "Kein Stift"
```

In order to write a macro which works always, we recommend using the international names:

```
myDoc.active_pen = "$ISO_NOPEN"
```

The proper pen name is calculated during run time of the macro. This command line would work on Arbortext IsoDraw, no matter if it is an English, German, Italian, French or even Japanese installation.

Pen Names

- $ISO_NOPEN
- $ISO_THICK
- $ISO_MEDIUM
- $ISO_THIN
- $ISO_CENTERLINE

Style Names

- $ISO_SOLID
- $ISO_DASHED
- $ISO_CENTERLINE
- $ISO_DOTTED
- $ISO_DOTDASHED
- $ISO_DASHDOTDOT
- $ISO_STARTARROW
- $ISO_ENDARROW
- $ISO_ENDDOT
- $ISO_ARROW
- $ISO_STITCHLINE
- $ISO_CENTERLINE2
- $ISO_HIDDENLINE
- $ISO_PHANTOMLINE
- $ISO_BREAKLINE1
- $ISO_BREAKLINE2

Shadow Names

- $ISO_NOSHADOW
- $ISO_LONG
- $ISO_MIDDLE
- $ISO_SHORT
- $ISO_CENTER

Color Names

- $ISO_NOFILL
- $ISO_WHITE
- $ISO_BLACK

Callout Names

- $ISO_NORMAL

26 IML File Format Names

The table below lists the file format names you can use in place of the keyword variables "*exportformat*" and "*processformat*" in the EXPORT and PROCESS commands.

For more information on these two commands, see Export on page 51 and Export on page 51.

"*exportformat*" Names	"*processformat*" Names	File Format
		Standard IsoDraw format of current version
	ISO_4	IsoDraw 4
		IsoDraw 3
		IsoDraw 2.6 format (not supported on Windows)
EPSF	EPSF	Encapsulated PostScript File
		Adobe Illustrator 1.1
AI	AI	Adobe Illustrator (88 if not spec. in preference)
IGES	IGES	Initial Graphics Exchange Standard format
DXF	DXF	Drawing Exchange Format
HPGL	HPGL	Hewlett Packard Graphics Language
CGM	CGM	Computer Graphics Metafile
PICT	PICT	PICTure format (not supported on Windows)
TIFF	TIFF	Tagged Image File
BMP	BMP	Bitmap
PCX	PCX	PCX
CALS_Raster	CALS_Raster	CALS Raster
Text_Excerpt	Text_Excerpt	Text Excerpt

"*exportformat*" Names	"*processformat*" Names	File Format
`Interleaf`	`Interleaf`	Interleaf
`MIF`	`MIF`	Maker Interchange Format
	`ISO 5`	IsoDraw 5
		IsoDraw 5, packed
`DWG`	`DWG`	DWG (AutoCAD)
`SVG`	`SVG`	Scalable Vector Graphics
`JPEG`	`JPEG`	JPEG (Joint Photographic Experts Group)
`PNG`	`PNG`	Portable Network Graphic
`WMF`	`WMF`	Windows Metafile
	`ISO 6`	IsoDraw 6
		IsoDraw 6, packed
	`ISO 7`	IsoDraw 7
	`ISOZ 7`	IsoDraw 7, packed
`XCF`	`XCF`	XML companion file)
`Objects_Text`	`Objects_Text`	Objects text excerpt
	`ISO_71`	IsoDraw 7.1
	`ISOZ_71`	IsoDraw 7.1, packed
`PDF`	`PDF`	3D PDF files
`U3D`	`U3D`	Universal 3D files

27

CGM Profile Numbers and Names

CGM Export profile numbers and names for the current release and older releases of Arbortext IsoDraw are listed below. The numbers (*n*) are integer values returned and set in the `app.cgm.profile` property of the Application object.

Arbortext IsoDraw 7.1 `app.cgm.profile` Numbers and Names

n	CGM Profile Name	Macro Constant Name
1	ISO 8632:1999	$CGM_Profile.ISO_8632_1999
2	WebCGM 1.0	$CGM_Profile.WebCGM_1_0
3	ATA GREXCHANGE V2.8	$CGM_Profile.ATA_GREXCHANGE_2_8
4	ATA GREXCHANGE V2.7	$CGM_Profile.ATA_GREXCHANGE_2_7
5	ATA GREXCHANGE V2.6	$CGM_Profile.ATA_GREXCHANGE_2_6
6	ATA GREXCHANGE V2.5	$CGM_Profile.ATA_GREXCHANGE_2_5
7	ATA GREXCHANGE V2.4	$CGM_Profile.ATA_GREXCHANGE_2_4
8	ATA GREXCHANGE V2.5/IsoDraw	$CGM_Profile.ATA_GREX-CHANGE_2_5_ISODRAW
9	MIL-D-28003A	$CGM_Profile.MIL_D_28003A
10	SAE J2008	$CGM_Profile.SAE_J2008
11	Model (8632:1992)	$CGM_Profile.Model_8632_1992
12	ISO ISP 12071-1	$CGM_Profile.ISO_ISP_12071_1
13	ISO ISP 12072-1	$CGM_Profile.ISO_ISP_12072_1
14	ISO ISP 12073-1	$CGM_Profile.ISO_ISP_12073_1
15	ISO ISP 12074-1	$CGM_Profile.ISO_ISP_12074_1
16	ATA GREXCHANGE V2.9	$CGM_Profile.ATA_GREXCHANGE_2_9
17	S1000D V2.2[1]	$CGM_Profile.S1000D_2_2
18	WebCGM 2.0	$CGM_Profile.WebCGM_2_0
19	ATA GREXCHANGE V2.10	$CGM_Profile.Current_ATA[2]

Arbortext IsoDraw 7.1 `app.cgm.profile` Numbers and Names (continued)

n	CGM Profile Name	Macro Constant Name
20	S1000D V2.3	No macro constant name
21	WebCGM 2.1	$CGM_Profile.WebCGM_2_1

1. S1000D V2.2 export is not supported in release 7.1 F000 and later.
2. The constant name $CGM_Profile.Current_AT always applies to the newest ATA GREXCHANGE profile that this release of Arbortext IsoDraw supports. When Arbortext IsoDraw supports a newer version, V2.10 will be assigned a different constant name; e.g., $CGM_Profile.ATA_GREXCHANGE_2_10.

Arbortext IsoDraw 7.0 F000 `app.cgm.profile` Numbers and Names

n	CGM Profile Name	Macro Constant Name
1	ISO 8632:1999	$CGM_Profile.ISO_8632_1999
2	WebCGM 1.0	$CGM_Profile.WebCGM_1_0
3	ATA GREXCHANGE V2.8	$CGM_Profile.ATA_GREXCHANGE_2_8
4	ATA GREXCHANGE V2.7	$CGM_Profile.ATA_GREXCHANGE_2_7
5	ATA GREXCHANGE V2.6	$CGM_Profile.ATA_GREXCHANGE_2_6
6	ATA GREXCHANGE V2.5	$CGM_Profile.ATA_GREXCHANGE_2_5
7	ATA GREXCHANGE V2.4	$CGM_Profile.ATA_GREXCHANGE_2_4
8	ATA GREXCHANGE V2.5/IsoDraw	$CGM_Profile.ATA_GREX-CHANGE_2_5_ISODRAW
9	MIL-D-28003A	$CGM_Profile.MIL_D_28003A
10	SAE J2008	$CGM_Profile.SAE_J2008
11	Model (8632:1992)	$CGM_Profile.Model_8632_1992
12	ISO ISP 12071-1	$CGM_Profile.ISO_ISP_12071_1
13	ISO ISP 12072-1	$CGM_Profile.ISO_ISP_12072_1
14	ISO ISP 12073-1	$CGM_Profile.ISO_ISP_12073_1
15	ISO ISP 12074-1	$CGM_Profile.ISO_ISP_12074_1
16	ATA GREXCHANGE V2.9	$CGM_Profile.ATA_GREXCHANGE_2_9
17	S1000D V2.2	$CGM_Profile.S1000D_2_2
18	WebCGM 2.0	$CGM_Profile.WebCGM_2_0
19	ATA GREXCHANGE V2.10	$CGM_Profile.Current_ATA[1]

1. The constant name $CGM_Profile.Current_AT always applies to the newest ATA GREXCHANGE profile that this release of Arbortext IsoDraw supports. When Arbortext IsoDraw supports a newer version, V2.10 will be assigned a different constant name; e.g., $CGM_Profile.ATA_GREXCHANGE_2_10.

Arbortext IsoDraw 6.1 M030 `app.cgm.profile` Numbers and Names

n	CGM Profile Name
1	ISO 8632:1999
2	WebCGM
3	ATA GREXCHANGE V2.8

Arbortext IsoDraw 6.1 M030 `app.cgm.profile` Numbers and Names (continued)

n	CGM Profile Name
4	ATA GREXCHANGE V2.7
5	ATA GREXCHANGE V2.6
6	ATA GREXCHANGE V2.5
7	ATA GREXCHANGE V2.4
8	ATA GREXCHANGE 2.5/IsoDraw
9	MIL-D-28003A
10	SAE J2008
11	Model (8632:1992)
12	ISO ISP 12071-1
13	ISO ISP 12072-1
14	ISO ISP 12073-1
15	ISO ISP 12074-1
16	ATA GREXCHANGE V2.9
17	AECMA S1000D

Arbortext IsoDraw 6.1 M020 `app.cgm.profile` Numbers and Names

n	CGM Profile Name
1	ISO 8632:1999
2	WebCGM
3	ATA GREXCHANGE V2.8
4	ATA GREXCHANGE V2.7
5	ATA GREXCHANGE V2.6
6	ATA GREXCHANGE V2.5
7	ATA GREXCHANGE V2.4
8	ATA GREXCHANGE 2.5/IsoDraw
9	MIL-D-28003A
10	SAE J2008
11	Model (8632:1992)
12	ISO ISP 12071-1
13	ISO ISP 12072-1
14	ISO ISP 12073-1
15	ISO ISP 12074-1
16	ATA GREXCHANGE V2.9

Arbortext IsoDraw 6.1 M010 `app.cgm.profile` Numbers and Names

n	CGM Profile Name
1	ISO 8632:1999
2	WebCGM

Arbortext IsoDraw 6.1 M010 `app.cgm.profile` Numbers and Names (continued)

n	CGM Profile Name
3	ATA GREXCHANGE V2.9
4	ATA GREXCHANGE V2.8
5	ATA GREXCHANGE V2.7
6	ATA GREXCHANGE V2.6
7	ATA GREXCHANGE V2.5
8	ATA GREXCHANGE V2.4
9	ATA GREXCHANGE 2.5/IsoDraw
10	MIL-D-28003A
11	SAE J2008
12	Model (8632:1992)
13	ISO ISP 12071-1
14	ISO ISP 12072-1
15	ISO ISP 12073-1
16	ISO ISP 12074-1

Arbortext IsoDraw 6.0 (M010 to M070) `app.cgm.profile` Numbers and Names

n	CGM Profile Name
1	ISO 8632:1999
2	WebCGM
3	ATA GREXCHANGE V2.8
4	ATA GREXCHANGE V2.7
5	ATA GREXCHANGE V2.6
6	ATA GREXCHANGE V2.5
7	ATA GREXCHANGE V2.4
8	ATA GREXCHANGE 2.5/IsoDraw
9	MIL-D-28003A
10	SAE J2008
11	Model (8632:1992)
12	ISO ISP 12071-1
13	ISO ISP 12072-1
14	ISO ISP 12073-1
15	ISO ISP 12074-1