ECLIPSE INCUBATION

ToolDef documentation (Incubation)

Version 0.1.0.20210305-171633

Table of Contents

ToolDef is a cross-platform and machine-independent scripting language. It supports command line execution, but is also available as plug-in for the Eclipse IDE, providing an integrated development experience.

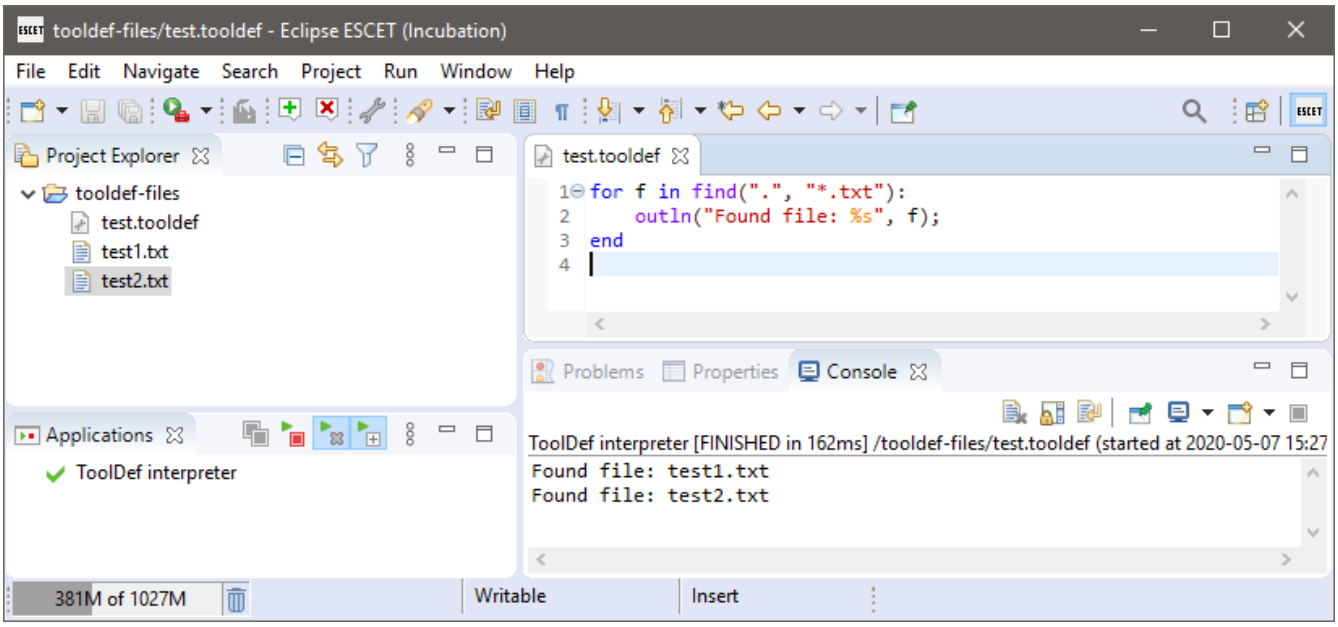
ToolDef is one of the tools of the Eclipse ESCET™ project. Visit the project website for downloads, installation instructions, source code, general tool usage information, information on how to contribute, and more.

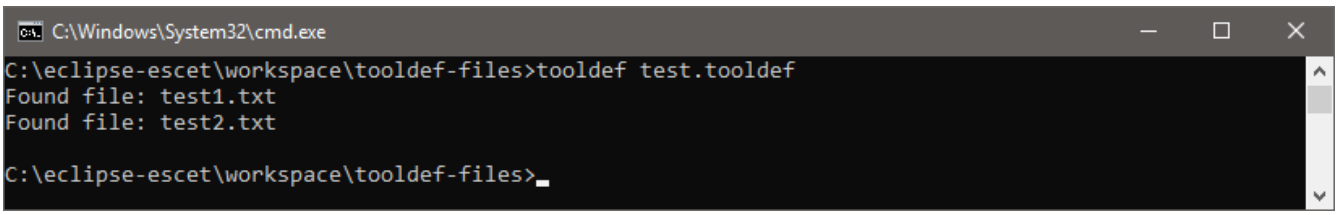
The Eclipse ESCET project, including the ToolDef language and toolset, is currently in the Incubation Phase.

The documentation consists of:

- ToolDef language reference manual
- ToolDef tool manual
- ToolDef release notes
- Legal information

Some screenshots of ToolDef editing and execution:





1. Introduction

The ToolDef language features the following concepts:

- Simple and intuitive syntax makes it easy to writing scripts.
- A large number of built-in data types, including booleans, integers, longs, doubles, strings, lists, sets, maps, tuples, and objects.
- Static typing allows to catch simple mistakes.
- The type system is compatible with Java, allowing easy integration with existing Java code.
- A large library of useful built-in operators and tools, including tools to write to the console, to start applications or execute other ToolDef scripts, to manipulate paths, to interact with files and directories, and to manipulate data of the various supported data types.
- Easy to extend with new tools, written in ToolDef itself, or in Java.
- Tools can be parameterized with types.
- Tools can be overloaded.
- Tools can have optional parameters.
- Tools can have a variable number of arguments.
- Tool invocations can use both positional and named arguments.
- Tools can return a single value, multiple values, or no values at all.
- Automatic type widening.
- Type declarations allow for reuse.
- Value semantics for ease of use.
- The usual imperative statements, including assignments, tool invocation, `if` statement, `for` statement, `while` statement, `return` statement, `exit` statement, `break` statement, and `continue` statement.
- Powerful assignment statement, including partial variable assignment, and multi-assignments.
- Variable declarations as statements, allowing variables to be declared in the middle of scripts, or in scopes, such as in bodies of `for` statements.
- Powerful `for` statement, allowing tuple unpacking, and iteration over values of various data types.
- A powerful import mechanism, allowing reuse of scripts, tools, and code.
- A registered library mechanism, allowing easy importing of libraries of tools.
- Platform independent path handling, allowing a single script with relative paths to be executed on Window, Linux, and Mac OS X.

For more information, see also the language reference documentation.

The ToolDef tooling features:

- Platform and machine independent execution of ToolDef scripts, on Microsoft Windows, Linux, and Mac OS X.
- Command line execution.
- Tight integration with the Eclipse IDE.
- User-friendly error messages.

For more information, see also the tool documentation.

2. Language reference

The documentation in this language reference manual serves as reference. It is not intended to be read from beginning to end, but more to look up specific information when needed. The reference documentation generally provides detailed and complete information, but this information may not always be presented in the most accessible way.

The information is divided into categories. The following categories and information is part of the language reference documentation:

Syntax

- Lexical syntax
- Grammar

Built-in tools and operators

- Built-in operators
- Built-in data tools
- Built-in I/O tools
- Built-in generic tools
- Built-in path tools
- Built-in file tools

2.1. Syntax

2.1.1. Lexical syntax

This page describes the ToolDef lexical syntax.

Keywords

General

```
as          else        if          map         tool
bool        end         import      null        true
break       exit        in          object      tuple
continue    false       int         return      type
double      for         list        set         while
elif        from        long        string
```

Operators

```
and         div         mod         not         or
```

Built-in data tools

```
abs         enumerate      log         reverse       str
ceil        floor          lower       round         strdup
contains    fmt            ltrim       rtrim         subset
del         indexof        max         size          trim
delidx      join           min         sorted        upper
empty       keys           pow         split         values
endswith    lastindexof    range       sqrt
entries     ln             replace     startswith
```

Built-in I/O tools

```
err         errln       out         outln
```

Built-in generic tools

```
app         exec        tooldef
```

Built-in path tools

```
abspath     dirname
basename    fileext
chdir       hasfileext
chfileext   pathjoin
curdir      scriptpath
```

Built-in file tools

```
cpdir       filenewer   isfile      readlines
cpfile      filesize    mkdir       rmdir
diff        find        mvdir       rmfile
exists      isdir       mvfile      writefile
```

Terminals

Besides the keyword terminals listed above, ToolDef features several other terminals:

IDENTIFIERTK

An identifier. Defined by the regular expression: `[$]?[a-zA-Z_][a-zA-Z0-9_]*`. They thus consist of letters, numbers and underscore characters (`_`). Identifiers may not start with a numeric digit. Keywords take priority over identifiers. To use a keyword as an identifier, prefix it with a `$` character. The `$` is not part of the identifier name.

Examples:

```
apple       // identifier
bear        // identifier
int         // keyword
$int        // identifier 'int' (override keyword priority with $)
```

RELATIVENAMETK

A name. Defined by the regular expression: `[$]?[a-zA-Z_][a-zA-Z0-9_]*(\.[$]?[a-zA-Z_][a-zA-Z0-9_]*)+`. It thus consists of two or more `IDENTIFIERTK` joined together with periods (`.`).

Examples:

```
some_library.some_tool
```

NUMBERTK

An integer literal. Defined by the regular expression: `0|[1-9][0-9]*`. Integers thus consist of numeric digits. Only for the number `0` may an integer literal start with `0`. E.g. `02` is invalid.

Examples:

```
0
1
123
```

DOUBLETK

A double literal. Defined by the regular expression: `(0|[1-9][0-9]*)(\.[0-9]+|(\.[0-9]+)?[eE][\-\+]?[0-9]+)`. Simple double literals consist of an integer literal followed by a period (`.`) and some numeric digits. Double literals using scientific notation start with either an integer literal or a simple double literal. They then contain either an `e` or `E`, followed by the exponent. The exponent consists of numeric digits, optionally preceded by `+` or `-`.

Examples:

```
0.0
1e5
1E+03
1.05e-78
```

`STRINGTK`

A string literal. Defined by the regular expression: `\"([^\\\"\n]|\\[nt\\\"])*\"`. String literals are enclosed in double quotes (`"`). String literals must be on a single line and must thus not include new line characters (`\n`, Unicode U+0A). To include a double quote (`"`) in a string literal, it must be escaped as `\"`. Since a backslash (`\`) serves as escape character, to include a backslash in a string literal it must be escaped as `\\`. To include a tab character in a string literal, use `\t`. To include a newline in a string literal, use `\n`.

Examples:

```
"hello world"
"first line\nsecond line"
```

Whitespace

ToolDef supports spaces, tabs, and new line characters as whitespace. Whitespace is ignored (except in string literals), but can be used to separate tokens as well as for layout purposes. The use of tab characters is allowed, but should be avoided if possible, as layout will be different for text editors with different tab settings. You may generally format a ToolDef script as you see fit, and start on a new line when desired.

Examples:

```
// Normal layout.
int x = 5;

// Alternative layout.
int
  x    =
    5
  ;
```

Comments

ToolDef features two types of comments. Single line comments start with `//` and end at end of the line. Multi line comments start with `/*` and end at `*/`. Comments are ignored.

Examples:

```
int x = 5; // Single line comment.

int /* some comment */ x = /* some
  more comments
  and some more
 end of the multi line comment */ 5;
```

2.1.2. Grammar

Below, the grammar of the ToolDef language is shown, in a form closely resembling Backus-Naur Form (BNF). The `Script` non-terminal is the start symbol of the grammar. Text between quotes are terminals. Names using only upper case letters are defined in the lexical syntax. The remaining names are the non-terminals of the grammar.

```
Script : /* empty */
       | Decls
       ;

Decls : Decl
      | Decls Decl
      ;

Decl : Import
     | "type" TypeDecls ";"
     | "tool" IDENTIFIERTK OptTypeParams ToolParameters ":" OptStatements "end"
     | "tool" Types IDENTIFIERTK OptTypeParams ToolParameters ":" OptStatements "end"
     | Statement
     ;

Import : "import" STRINGTK ";"
       | "import" STRINGTK "as" IDENTIFIERTK ";"
       | "import" Name ";"
       | "import" Name "as" IDENTIFIERTK ";"
       | "import" Name ":" Name ";"
       | "import" Name ":" Name "as" IDENTIFIERTK ";"
       | "from" STRINGTK "import" ImportParts ";"
       ;

ImportParts : "*"
            | IDENTIFIERTK
            | IDENTIFIERTK "as" IDENTIFIERTK
            | ImportParts "," "*"
            | ImportParts "," IDENTIFIERTK
            | ImportParts "," IDENTIFIERTK "as" IDENTIFIERTK
            ;
```

```
TypeDecls : TypeDecl
          | TypeDecls "," TypeDecl
          ;

TypeDecl : IDENTIFIERTK "=" Type
         ;

OptTypeParams : /* empty */
              | "<" Names ">"
              ;

ToolParameters : "(" ")"
               | "(" ToolParams ")"
               ;

ToolParams : Type ToolParam
           | Type "..." ToolParam
           | ToolParams "," Type ToolParam
           | ToolParams "," Type "..." ToolParam
           ;

ToolParam : IDENTIFIERTK
          | IDENTIFIERTK "=" Expression
          ;

OptStatements : /* empty */
              | OptStatements Statement
              ;

Statement : Type VarDecls ";"
          | "while" Expression ":" OptStatements "end"
          | "while" Expression "::" Statement
          | "for" AddressableDecls "in" Expression ":" OptStatements "end"
          | "for" AddressableDecls "in" Expression "::" Statement
          | "if" Expression ":" OptStatements OptElifStatements OptElseStatement "end"
          | "if" Expression "::" Statement
          | "break" ";"
          | "continue" ";"
          | Addressables "=" Expressions ";"
          | "return" ";"
          | "return" Expressions ";"
          | ToolInvokeExpression ";"
          | "exit" ";"
          | "exit" Expression ";"
          ;

VarDecls : VarDecl
         | VarDecls "," VarDecl
         ;
```

```
VarDecl : IDENTIFIERTK
        | IDENTIFIERTK "=" Expression
        ;

OptElifStatements : /* empty */
                  | OptElifStatements "elif" Expression ":" OptStatements
                  ;

OptElseStatement : /* empty */
                 | "else" OptStatements
                 ;

AddressableDecls : AddressableDecl
                 | AddressableDecls "," AddressableDecl
                 ;

AddressableDecl : IDENTIFIERTK
                | "(" AddressableDecl "," AddressableDecls ")"
                ;

Addressables : Addressable
             | Addressables "," Addressable
             ;

Addressable : IDENTIFIERTK
            | IDENTIFIERTK Projections
            | "(" Addressable "," Addressables ")"
            ;

Projections : Projection
            | Projections Projection
            ;

Projection : "[" Expression "]"
           ;

Types : Type
      | Types "," Type
      ;

Type : "bool"
     | "bool" "?"
     | "int"
     | "int" "?"
     | "long"
     | "long" "?"
     | "double"
     | "double" "?"
     | "string"
     | "string" "?"
     | "list" Type
```

```
     | "list" "?" Type
     | "set" Type
     | "set" "?" Type
     | "map" "(" Type ":" Type ")"
     | "map" "?" "(" Type ":" Type ")"
     | "tuple" "(" Type "," Types ")"
     | "tuple" "?" "(" Type "," Types ")"
     | "object"
     | "object" "?"
     | Name
     ;

Expressions : Expression
            | Expressions "," Expression
            ;

OptExpression : /* empty */
              | Expression
              ;

Expression : AndExpression
           | Expression "or" AndExpression
           ;

AndExpression : CompareExpression
              | AndExpression "and" CompareExpression
              ;

CompareExpression : AddExpression
                  | CompareExpression "<" AddExpression
                  | CompareExpression "<=" AddExpression
                  | CompareExpression "==" AddExpression
                  | CompareExpression "!=" AddExpression
                  | CompareExpression ">=" AddExpression
                  | CompareExpression ">" AddExpression
                  ;

AddExpression : MulExpression
              | AddExpression "-" MulExpression
              | AddExpression "+" MulExpression
              ;

MulExpression : UnaryExpression
              | MulExpression "*" UnaryExpression
              | MulExpression "/" UnaryExpression
              | MulExpression "div" UnaryExpression
              | MulExpression "mod" UnaryExpression
              ;

UnaryExpression : ProjExpression
                | "-" UnaryExpression
```

```
               | "+" UnaryExpression
               | "not" UnaryExpression
               ;

ProjExpression : ExpressionFactor
               | ProjExpression "[" Expression "]"
               | ProjExpression "[" OptExpression ":" OptExpression "]"
               ;

ExpressionFactor : "true"
                 | "false"
                 | NUMBERTK
                 | DOUBLETK
                 | "null"
                 | STRINGTK
                 | "<" Type ">" ExpressionFactor
                 | "[" "]"
                 | "[" Expressions OptComma "]"
                 | "{" "}"
                 | "{" Expressions OptComma "}"
                 | "{" MapEntries OptComma "}"
                 | "(" Expression "," Expressions OptComma ")"
                 | "(" Expression ")"
                 | ToolInvokeExpression
                 | Name
                 ;

MapEntries : Expression ":" Expression
           | MapEntries "," Expression ":" Expression
           ;

ToolInvokeExpression : ToolRef "(" ToolArgs OptComma ")"
                     | ToolRef "(" ")"
                     ;

ToolRef : BuiltInTool
        | Name
        ;

BuiltInTool : BuiltInIoTool
            | BuiltInGenericTool
            | BuiltInPathTool
            | BuiltInFileTool
            | BuiltInDataTool
            ;

ToolArgs : Expression
         | IDENTIFIERTK "=" Expression
         | ToolArgs "," Expression
         | ToolArgs "," IDENTIFIERTK "=" Expression
         ;
```

```
Names : Name
      | Names "," Name
      ;

Name : IDENTIFIERTK
     | RELATIVENAMETK
     ;

OptComma : /* empty */
         | ","
         ;

BuiltInIoTool : "err"
              | "errln"
              | "out"
              | "outln"
              ;

BuiltInGenericTool : "app"
                   | "exec"
                   | "tooldef"
                   ;

BuiltInPathTool : "abspath"
                | "basename"
                | "chdir"
                | "chfileext"
                | "curdir"
                | "dirname"
                | "fileext"
                | "hasfileext"
                | "pathjoin"
                | "scriptpath"
                ;

BuiltInFileTool : "cpdir"
                | "cpfile"
                | "diff"
                | "exists"
                | "filenewer"
                | "filesize"
                | "find"
                | "isdir"
                | "isfile"
                | "mkdir"
                | "mvdir"
                | "mvfile"
                | "readlines"
                | "rmdir"
                | "rmfile"
```

```
                | "writefile"
                ;

BuiltInDataTool : "abs"
                | "ceil"
                | "contains"
                | "del"
                | "delidx"
                | "empty"
                | "endswith"
                | "entries"
                | "enumerate"
                | "floor"
                | "fmt"
                | "indexof"
                | "join"
                | "keys"
                | "lastindexof"
                | "ln"
                | "log"
                | "lower"
                | "ltrim"
                | "max"
                | "min"
                | "pow"
                | "range"
                | "replace"
                | "reverse"
                | "round"
                | "rtrim"
                | "size"
                | "sorted"
                | "split"
                | "sqrt"
                | "startswith"
                | "str"
                | "strdup"
                | "subset"
                | "trim"
                | "upper"
                | "values"
                ;
```

2.2. Built-in tools and operators

2.2.1. Built-in operators

This page describes the built-in operators:

- not

- and
- or
- + (unary)
- - (unary)
- + (binary)
- - (binary)
- *
- /
- div
- mod
- <
- <=
- >
- >=
- ==
- !=

The signatures of the operators are given using tool headers, to show the name of the operator, the types of the arguments, and the type of the resulting value. Operators however, can not be used by means of tool invocations. Operators with one argument are put directly before the argument (e.g. `not true`, `-5`), while operators with two arguments are put between the arguments (e.g. `true and false`, `1 + 3`).

not operator

```
tool bool not(bool arg)
```

Returns the logical inverse of a boolean value.

Arguments

`arg`
: The boolean value.

Returns

The logical inverse result.

and operator

```
tool bool  and(bool left, bool right)
tool set T and<T>(set T left, set T right)
```

Returns the conjunction of two boolean values, or the intersection of two sets. For boolean values, the operator uses short circuit evaluation. That is, the `right` argument is only evaluated if necessary, i.e. only if the `left` argument evaluates to `true`.

Type parameters

`T`

The type of the elements of the set.

Arguments

`left`

The first boolean value or set.

`right`

The second boolean value or set.

Returns

The conjunction or intersection result.

or operator

```
tool bool  or(bool left, bool right)
tool set T or<T>(set T left, set T right)
```

Returns the disjunction of two boolean values, or the union of two sets. For boolean values, the operator uses short circuit evaluation. That is, the `right` argument is only evaluated if necessary, i.e. only if the `left` argument evaluates to `false`.

Type parameters

`T`

The type of the elements of the set.

Arguments

`left`

The first boolean value or set.

`right`

The second boolean value or set.

Returns

The disjunction or union result.

+ operator (unary)

```
tool int    +(int arg)
tool long   +(long arg)
tool double +(double arg)
```

Returns the unary plus of an integer, long, or double value. This is essentially the identity function.

Arguments

`arg`

The integer, long, or double value.

Returns

The integer, long, or double value.

+ operator (binary)

```
tool int      +(int left, int right)
tool long     +(long left, long right)
tool double   +(double left, double right)
tool string   +(string left, string right)
tool list T   +<T>(list T left, list T right)
tool map(K:V) +<K, V>(map(K:V) left, map(K:V) right)
```

Returns the addition of two integer, long, or double numbers, the concatenation of two strings or lists, or the update of a first map with the entries of a second map. For two maps, essentially, the entries of the first map are overwritten by the entries of the second map, while entries for new keys are added.

Type parameters

`T`

The type of the elements of the list.

`K`

The type of the keys of the map.

`V`

The type of the values of the map.

Arguments

`left`

The first integer, long, or double number, string, list, or map.

`right`

The second integer, long, or double number, string, list, or map.

Returns

The addition, concatenation, or map update result.

Runtime errors

- If the operation results in overflow (for integer, long, and double numbers only).

- operator (unary)

```
tool int    -(int arg)
tool long   -(long arg)
tool double -(double arg)
```

Returns the negation of an integer, long, or double value.

Arguments

`arg`

The integer, long, or double value.

Returns

The negation result.

Runtime errors

- If the operation results in overflow.

- operator (binary)

```
tool int      -(int left, int right)
tool long     -(long left, long right)
tool double   -(double left, double right)
tool set T    -<T>(set T left, set T right)
tool map(K:V) -<K, V>(map(K:V) left, list K right)
tool map(K:V) -<K, V>(map(K:V) left, set K right)
tool map(K:V) -<K, V, V2>(map(K:V) left, map(K:V2) right)
```

Returns the subtraction of two integer, long, or double numbers, the set difference of two sets, the map with the keys from the list removed from it, the map with the keys from the set removed from it, or the first map with the keys from the second map removed from it.

Type parameters

`T`

The type of the elements of the list.

`K`

The type of the keys of the map.

`V`

The type of the values of the map.

Arguments

`left`

The first integer, long, or double number, the first set, or the (first) map.

`right`

The second integer, long, or double number, the (second) set, the list, or the second map.

Returns

The subtraction, set difference, or map removal result.

Runtime errors

- If the operation results in overflow (for integer, long, and double numbers only).

*** operator**

```
tool int    *(int left, int right)
tool long   *(long left, long right)
tool double *(double left, double right)
```

Returns the multiplication of two integer, long, or double numbers.

Arguments

`left`
: The first integer, long, or double number.

`right`
: The second integer, long, or double number.

Returns

The multiplication result.

Runtime errors

- If the operation results in overflow.

/ operator

```
tool double /(double left, double right)
```

Returns the division of two double numbers.

Arguments

`left`
: The first double number.

`right`
: The second double number.

Returns

The division result.

Runtime errors

- If the operation results in overflow or division by zero.

div operator

```
tool int  div(int left, int right)
tool long div(long left, long right)
```

Returns the integer division of two integer or long numbers.

Arguments

`left`
: The first integer or long number.

`right`
: The second integer or long number.

Returns

The integer division result.

Runtime errors

- If the operation results in overflow or division by zero.

mod operator

```
tool int  mod(int left, int right)
tool long mod(long left, long right)
```

Returns the modulus of two integer or long numbers.

Arguments

`left`
: The first integer or long number.

`right`
: The second integer or long number.

Returns

The modulus result.

Runtime errors

- If the operation results in division by zero.

< operator

```
tool bool <(int left, int right)
tool bool <(long left, long right)
tool bool <(double left, double right)
```

Returns whether the first integer, long, or double number is less than the second integer, long, or double number.

Arguments

`left`

The first integer, long, or double number.

`right`

The second integer, long, or double number.

Returns

`true` if the first number is less than the second number, `false` otherwise.

<= operator

```
tool bool <=(int left, int right)
tool bool <=(long left, long right)
tool bool <=(double left, double right)
```

Returns whether the first integer, long, or double number is less than or equal to the second integer, long, or double number.

Arguments

`left`

The first integer, long, or double number.

`right`

The second integer, long, or double number.

Returns

`true` if the first number is less than or equal to the second number, `false` otherwise.

> operator

```
tool bool >(int left, int right)
tool bool >(long left, long right)
tool bool >(double left, double right)
```

Returns whether the first integer, long, or double number is greater than the second integer, long, or double number.

Arguments

`left`

The first integer, long, or double number.

`right`

The second integer, long, or double number.

Returns

`true` if the first number is greater than the second number, `false` otherwise.

>= operator

```
tool bool >=(int left, int right)
tool bool >=(long left, long right)
tool bool >=(double left, double right)
```

Returns whether the first integer, long, or double number is greater than or equal to the second integer, long, or double number.

Arguments

`left`

The first integer, long, or double number.

`right`

The second integer, long, or double number.

Returns

`true` if the first number is greater than or equal to the second number, `false` otherwise.

== operator

```
tool bool ==<T>(T left, T right)
```

Returns whether the first value is equal to the second value.

Type parameters

`T`

The type of the values.

Arguments

`left`
: The first value. May be `null`.

`right`
: The second value. May be `null`.

Returns

`true` if the first value is equal to the second value, `false` otherwise.

!= operator

```
tool bool !=<T>(T left, T right)
```

Returns whether the first value is unequal to the second value.

Type parameters

`T`
: The type of the values.

Arguments

`left`
: The first value. May be `null`.

`right`
: The second value. May be `null`.

Returns

`true` if the first value is unequal to the second value, `false` otherwise.

2.2.2. Built-in data tools

This page describes the built-in data tools:

- abs
- ceil
- contains
- del
- delidx
- empty
- endswith

- entries
- enumerate
- floor
- fmt
- indexof
- join
- keys
- lastindexof
- ln
- log
- lower
- ltrim
- max
- min
- pow
- range
- replace
- reverse
- round
- rtrim
- size
- sorted
- split
- sqrt
- startswith
- str
- strdup
- subset
- trim
- upper
- values

abs tool

```
tool int    abs(int x)
tool long   abs(long x)
tool double abs(double x)
```

Returns the absolute value of an integer, long, or double number.

Parameters

`x`
: The integer, long, or double number.

Returns

The absolute value.

Runtime errors

- If the operation results in overflow (for integer and long numbers only).

ceil tool

```
tool long ceil(double x)
```

Returns the given double number rounded to a whole long number, towards positive infinity.

Parameters

`x`
: The double number.

Returns

The double number rounded to a whole long number, towards positive infinity.

Runtime errors

- If the operation results in overflow.

contains tool

```
tool bool contains(string whole, string part)
tool bool contains<T>(list T $list, T elem)
tool bool contains<T>(set T $set, T elem)
tool bool contains<K,V>(map(K:V) $map, K key)
```

Returns whether a given string is contained in another given string, a given value is contained in a given list, a given value is contained in a given set, or a given value is a key of a given map.

Type parameters

`T`

The type of the elements of the list or set.

`K`

The type of the keys of the map.

`V`

The type of the values of the map.

Parameters

`whole`, `list`, `set`, `map`

The whole string, the list, the set, or the map.

`part`, `elem`, `key`

The part string (potentially contained in the whole string), or the value (potential element of the list or set, or potential key of the map).

Returns

`true` if the part string is contained in the whole string, if the value is contained in the list, if the value is contained in the set, or if the value is a key of the map, `false` otherwise.

del tool

```
tool list T   del<T>(list T $list, T elem)
tool set T    del<T>(set T $set, T elem)
tool map(K:V) del<K,V>(map(K:V) $map, K key)
```

Returns the given list with all occurrences of the given element removed from it, the given set with the given element removed from it, or the given map with the given entry with the given key removed from it. If the element or key does not exist, the list, set or map is returned unmodified.

Type parameters

`T`

The type of the elements of the list or set.

`K`

The type of the keys of the map.

`V`

The type of the values of the map.

Parameters

`list`, `set`, `map`

The list, set, or map.

`elem`, `key`

The element or key to remove. May be `null`.

Returns

The list with all occurrences of the element removed from it, the set with the element removed from it, or the map with the entry with the given key removed from it.

delidx tool

```
tool list T delidx<T>(list T $list, int index)
```

Removes an element from a list, and returns the list without that element.

Type parameters

`T`

The type of the elements of the list.

Parameters

`list`

The list.

`index`

The 0-based index into the list of the element to remove. Negative indices are allowed, and count from the right.

Returns

The list with the element removed from it.

Runtime errors

- If the index is out of bounds for the list.

empty tool

```
tool bool empty(string x)
tool bool empty<T>(list T x)
tool bool empty<T>(set T x)
tool bool empty<K,V>(map(K:V) x)
```

Returns whether a given string, list, set, or map is empty (no characters, elements, or entries).

Type parameters

`T`

The type of the elements of the list or set.

`K`

The type of the keys of the map.

`V`

The type of the values of the map.

Parameters

`x`

The string, list, set, or map.

Returns

`true` if the given string, list, set, or map is empty, `false` otherwise.

endswith tool

```
tool bool endswith(string whole, string suffix)
```

Returns whether the first given string ends with the second given string. For an empty suffix, always returns `true`.

Parameters

`whole`

The first given string.

`suffix`

The second given string.

Returns

`true` if the first given string ends with the second given string, `false` otherwise.

entries tool

```
tool list tuple(K, V) entries<K,V>(map(K:V) $map)
```

Returns a list with 2-tuples of keys and values for all entries of the given map.

Type parameters

`K`
: The type of the keys of the map.

`V`
: The type of the values of the map.

Parameters

`map`
: The map.

Returns

The list with 2-tuples of keys and values for all entries of the given map.

enumerate tool

```
tool list tuple(int, T) enumerate<T>(list T $list)
```

Returns a list with 2-tuples of 0-based indices and elements for all elements of the given list.

Type parameters

`T`
: The type of the elements of the list.

Parameters

`list`
: The list.

Returns

The list with 2-tuples of 0-based indices and elements for all elements of the given list.

floor tool

```
tool long floor(double x)
```

Returns the given double number rounded to a whole long number, towards negative infinity.

Parameters

`x`
: The double number.

Returns

The double number rounded to a whole long number, towards negative infinity.

Runtime errors

- If the operation results in overflow.

fmt tool

```
tool string fmt(string pattern, object?... args)
```

Formats a text based on a pattern and arguments.

See also the str tool, as `str(value)` is equal to `fmt("%s", value)`.

Parameters

`pattern`
: The format pattern.

`args`
: The arguments.

Returns

The formatted text.

Runtime errors

- If the format pattern is invalid.
- If the format pattern and arguments don't match.

indexof tool

```
tool int indexof(string whole, string part)
tool int indexof(string whole, string part, int offset)
tool int indexof<T>(list T $list, T elem)
tool int indexof<T>(list T $list, T elem, int offset)
```

Returns the 0-based index of the first occurrence of the second given string or value, in the first given string or list. The tool looks for an occurrence from left to right of the first given string or elements of the first given list. If an offset is given, the tool only looks for an occurrence at or after that 0-based offset. If no offset is not given, the tool starts looking at the first (left most) character or list element.

Returns `-1` if the second given string or value doesn't occur in the first given string or list, at or after the 0-based offset if given, or at all if no offset is given. For an empty second string, always returns `0` if no offset is given. For an empty second string, if an offset is given, always returns the offset, unless there is no character at the given offset, in which `-1` is always returned.

Type parameters

`T`

: The type of the elements of the list.

Parameters

`whole`, `list`

: The first given string, or the list.

`part`, `elem`

: The second given string, or the value.

`offset`

: The optional 0-based offset.

Returns

The 0-based index of the first occurrence, at or after the given offset if given, or `-1`.

join tool

```
tool string join(string... texts)
tool string join(list string texts)
tool string join(list string texts, string separator)
```

Returns the given texts joined together. They can be joined without any separator, or using a given separator.

Parameters

`texts`
: The texts to join.

`separator`
: The optional separator text to use.

Returns

The joined texts.

keys tool

```
tool set K keys<K,V>(map(K:V) $map)
```

Returns a set with the keys of the given map.

Type parameters

`K`
: The type of the keys of the map.

`V`
: The type of the values of the map.

Parameters

`map`
: The map.

Returns

The set with the keys of the map.

lastindexof tool

```
tool int lastindexof(string whole, string part)
tool int lastindexof(string whole, string part, int offset)
tool int lastindexof<T>(list T $list, T elem)
tool int lastindexof<T>(list T $list, T elem, int offset)
```

Returns the 0-based index of the last occurrence of the second given string or value, in the first given string or list. The tool looks for an occurrence from left to right of the first given string or

elements of the first given list. If an offset is given, the tool only looks for an occurrence at or before that 0-based offset. If no offset is not given, the tool starts looking at the first (left most) character or list element.

Returns `-1` if the second given string or value doesn't occur in the first given string or list, at or before the 0-based offset if given, or at all if no offset is given. For an empty second string, always returns the size of the first given string, if no offset is given. For an empty second string, if an offset is given, always returns the offset, unless there is no character at the given offset, in which `-1` is always returned.

Type parameters

`T`

The type of the elements of the list.

Parameters

`whole`, `list`

The first given string, or the list.

`part`, `elem`

The second given string, or the value.

`offset`

The optional 0-based offset.

Returns

The 0-based index of the last occurrence, at or before the given offset if given, or `-1`.

ln tool

```
tool double ln(double x)
```

Returns the natural logarithm of a double number.

Parameters

`x`

The double number.

Returns

The natural logarithm of the double number.

Runtime errors

- If the double number is not positive.

log tool

```
tool double log(double x)
```

Returns the logarithm (base 10) of a double number.

Parameters

`x`
: The double number.

Returns

The logarithm (base 10) of the double number.

Runtime errors

- If the double number is not positive.

lower tool

```
tool string lower(string text)
```

Returns the given text, converted to lower case. Uses a US English locale.

Parameters

`text`
: The text.

Returns

The text, in lower case.

ltrim tool

```
tool string ltrim(string text)
```

Returns the given text with whitespace at the left/start of the text removed.

Parameters

`text`

The text.

Returns

The text with whitespace at the left/start of the text removed.

max tool

```
tool int    max(int... x)
tool long   max(long... x)
tool double max(double... x)
tool int    max(list int x)
tool long   max(list long x)
tool double max(list double x)
```

Returns the maximum of some integer, long, or double numbers. If no numbers are given, the minimum representable finite integer, long, or double value is returned.

Parameters

`x`

The integer, long, or double numbers.

Returns

The maximum number.

min tool

```
tool int    min(int... x)
tool long   min(long... x)
tool double min(double... x)
tool int    min(list int x)
tool long   min(list long x)
tool double min(list double x)
```

Returns the minimum of some integer, long, or double numbers. If no numbers are given, the maximum representable finite integer, long, or double value is returned.

Parameters

`x`

The integer, long, or double numbers.

Returns

The minimum number.

pow tool

```
tool double pow(double base, double exponent)
```

Returns the exponentiation (power) of two double numbers.

Parameters

`base`
: The base double number.

`exponent`
: The exponent double number.

Returns

The exponentiation result.

Runtime errors

- If the operation results in double overflow, or `NaN`.

range tool

```
tool list int range(int count)
tool list int range(int begin, int $end)
tool list int range<T>(list T $list)
```

Returns a list of numbers representing the range `[0..count-1]`, `[begin..end]`, or `[0..size(list)-1]`. That is, for `count` = `5` the list `[0, 1, 2, 3, 4]` is returned, for `begin` = `-3` and `end` = `2` the list `[-3, -2, -1, 0, 1, 2]` is returned, and for a `list` with `5` elements the list `[0, 1, 2, 3, 4]` is returned.

Type parameters

`T`
: The type of the elements of the list.

Parameters

`count`
: The number of elements in the resulting list. Negative counts are treated as zero.

`begin`

The lower bound of the range.

`end`

The upper bound of the range.

`list`

The list.

Returns

The range `[0..count-1]`, `[begin..end]`, or `[0..size(list)-1]`.

replace tool

```
tool string replace(string text, string oldtext, string newtext)
tool list T replace<T>(list T $list, T oldelem, T newelem)
```

Returns the string or list with old sub strings or elements replaced by new text or elements. For strings, the replacement proceeds from the beginning of the string to the end. That is, replacing `"aa"` with `"b"` in the string `"aaa"` will result in `"ba"` rather than `"ab"`. For lists, the replacement is performed for all matching old elements.

Type parameters

`T`

The type of the elements of the list.

Parameters

`text`, `list`

The text (string) or list in which to replace text or elements.

`oldtext`, `oldelem`

The text (sub string) or element to replace. For list elements, it may be `null`.

`newtext`, `newelem`

The replacement text or element. For list elements, it may be `null`.

Returns

The text or list with all replacements applied.

reverse tool

```
tool string reverse(string text)
tool list T reverse<T>(list T $list)
```

Returns the reverse of the given string or list.

Type parameters

`T`

The type of the elements of the list.

Parameters

`text`, `list`

The string or list.

Returns

The reverse string or list.

round tool

```
tool long round(double x)
```

Returns the given double number rounded to the closest whole long number, with ties being rounded toward positive infinity.

Parameters

`x`

The double number.

Returns

The rounded number.

Runtime errors

- If the operation results in overflow.

rtrim tool

```
tool string rtrim(string text)
```

Returns the given text with whitespace at the right/end of the text removed.

Parameters

`text`
: The text.

Returns

The text with whitespace at the right/end of the text removed.

size tool

```
tool int size(string x)
tool int size<T>(list T x)
tool int size<T>(set T x)
tool int size<K,V>(map(K:V) x)
```

Returns the size (number of characters, elements, or entries) of the given string, list, set, or map.

Type parameters

`T`
: The type of the elements of the list or set.

`K`
: The type of the keys of the map.

`V`
: The type of the values of the map.

Parameters

`x`
: The string, list, set, or map.

Returns

The size.

sorted tool

```
tool list T sorted<T>(list T $list)
tool list T sorted<T>(set T $set)
```

Returns the given list, or the given set as a list, with the elements sorted in ascending order.

Type parameters

`T`

The type of the elements of the list or set.

Parameters

`list`

The list.

`set`

The set.

Returns

The sorted list.

split tool

```
tool list string split(string text, string? separator = null, bool removeEmpty = true)
```

Splits the given text at all non-overlapping occurrences of the given separator.

Parameters

`text`

The text to split.

`separator`

The separator text. May be empty or `null` to split on whitespace.

`removeEmpty`

Whether to remove empty parts (in case the text starts or ends with the separator, or in case the text contains consecutive separators), or not.

Returns

The parts of the original text resulting from splitting the original text at the separators.

sqrt tool

```
tool double sqrt(double x)
```

Returns the square root of a double number.

Parameters

`x`
: The double number.

Returns

The square root.

Runtime errors

- If the double number is negative.

startswith tool

```
tool bool startswith(string whole, string prefix)
```

Returns whether the first given string starts with the second given string. For an empty prefix, always returns `true`.

Parameters

`whole`
: The first given string.

`prefix`
: The second given string.

Returns

`true` if the first given string starts with the second given string, `false` otherwise.

str tool

```
tool string str(object? value)
```

Converts the given value into a textual representation, closely resembling the ToolDef syntax.

Values of type `string` as returned as provided. Values of type `string` included in containers such as lists are escaped and surrounded by double quotes, as in the ToolDef syntax.

See also the fmt tool, as `str(value)` is equal to `fmt("%s", value)`.

Parameters

`value`
: The value to convert to a textual representation.

Returns

The textual representation.

strdup tool

```
tool string strdup(string text, int count)
```

Duplicates a string a given number of times.

Parameters

`text`
: The string to duplicate.

`count`
: The number of times that the input string should occur in the output.

Returns

The concatenation of `count` times the `text`.

Runtime errors

- If the count is negative.

subset tool

```
tool bool subset<T>(set T part, set T whole)
```

Returns whether the first given set is a subset (proper subset or equal) of the second given set.

Type parameters

`T`
: The type of the elements of the sets.

Parameters

`part`
: The first set.

`whole`
: The second set.

Returns

`true` if the first given set is a subset of the second given set, `false` otherwise.

trim tool

```
tool string trim(string text)
```

Returns the given text with whitespace at the left/start and right/end of the text removed.

Parameters

`text`
: The text.

Returns

The text with whitespace at the left/start and right/end of the text removed.

upper tool

```
tool string upper(string text)
```

Returns the given text, converted to upper case. Uses a US English locale.

Parameters

`text`
: The text.

Returns

The text, in upper case.

values tool

```
tool set V values<K,V>(map(K:V) $map)
```

Returns a set with the values of the given map.

Type parameters

`K`
: The type of the keys of the map.

`V`
: The type of the values of the map.

Parameters

`map`
: The map.

Returns

The set with the values of the map.

2.2.3. Built-in I/O tools

This page describes the built-in I/O tools:

- out
- err
- outln
- errln

out tool

```
tool out(string pattern, object?... args)
```

Prints formatted text (based on a pattern and arguments) to the standard output stream (stdout).

Parameters

`pattern`
: The format pattern.

`args`
: The arguments.

Runtime errors

- If the format pattern is invalid.
- If the format pattern and arguments don't match.

err tool

```
tool err(string pattern, object?... args)
```

Prints formatted text (based on a pattern and arguments) to the standard error stream (stderr).

Parameters

`pattern`

The format pattern.

`args`

The arguments.

Runtime errors

- If the format pattern is invalid.
- If the format pattern and arguments don't match.

outln tool

```
tool outln(string pattern = "", object?... args)
```

Prints formatted text (based on a pattern and arguments) to the standard output stream (stdout), printing an additional new line at the end.

Parameters

`pattern`

The format pattern.

`args`

The arguments.

Runtime errors

- If the format pattern is invalid.
- If the format pattern and arguments don't match.

errln tool

```
tool errln(string pattern = "", object?... args)
```

Prints formatted text (based on a pattern and arguments) to the standard error stream (stderr), printing an additional new line at the end.

Parameters

`pattern`

The format pattern.

`args`

The arguments.

Runtime errors

- If the format pattern is invalid.
- If the format pattern and arguments don't match.

2.2.4. Built-in generic tools

This page describes the built-in generic tools:

- app
- exec
- tooldef

app tool

```
tool int app(string? plugin, string classname, string... args,        string stdin =
"-", string stdout = "-", string stderr = "-", bool appendOut = false, bool appendErr
= false, bool errToOut = false, bool ignoreNonZeroExitCode = false)
tool int app(string? plugin, string classname, list string args = [], string stdin =
"-", string stdout = "-", string stderr = "-", bool appendOut = false, bool appendErr
= false, bool errToOut = false, bool ignoreNonZeroExitCode = false)
```

Executes an application framework application.

The application is specified as a Java class that extends the `org.eclipse.escet.common.app.framework.Application` class. It should have a constructor that accepts one parameter, of type `org.eclipse.escet.common.app.framework.io.AppStreams`.

The application is started within the current Java interpreter, in a new thread. If executed from Eclipse, the class is loaded using the class loader of the given bundle (if provided), otherwise, it is

loaded using the default class loader. In the latter case, the class should be in the class path.

Note that this tool is only machine independent and platform independent, as long as the applications that are executed, are machine independent and platform independent as well.

Parameters

`plugin`

The name of the plug-in (OSGi bundle) in which to resolve the application class. If the Java application is executed from Eclipse, then the class is loaded by the class loader of this bundle, otherwise it is resolved using the default class loader. May be `null` to always use the default class loader.

`classname`

The absolute name of the Java class to execute as application.

`args`

The command line arguments of the application to execute. Each argument string is parsed to zero or more actual arguments. It is possible to use a single string with all arguments (where the arguments themselves are separated by spaces), comma separated strings for each of the arguments (each string has one argument), or a mix of those.

In argument strings, individual arguments are separated by whitespace (spaces, tabs, new lines, etc). The whitespace itself is ignored, and only serves as separation. To include whitespace in an argument, the argument (or a part of it), may be quoted, by putting it between double quotes (`"`) or single quotes (`'`). Characters may be escaped by prefixing them with a backslash (`\`). This is particularly useful for single/double quotes, and escape characters, and can also be used to escape spaces. Escapes work the same inside of quoted parts as they do outside of quoted parts.

`stdin`

Specify whether to have a standard input (stdin) stream and where the input comes from. Use `""` to not have a stdin stream, `"-"` to use the stdin stream of the ToolDef interpreter, or otherwise an absolute or relative local file system path of the file from which to read the standard input. May contain both `\` and `/` as path separators.

`stdout`

Specify whether to have a standard output (stdout) stream and where to write the standard output. Use `""` to not have a stdout stream, `"-"` to use the stdout stream of the ToolDef interpreter, or otherwise an absolute or relative local file system path of the file to which to write the standard output. May contain both `\` and `/` as path separators.

`stderr`

Specify whether to have a standard error (stderr) stream and where to write the standard error output. Use `""` to not have a stderr stream, `"-"` to use the stderr stream of the ToolDef interpreter, or otherwise an absolute or relative local file system path of the file to which to write the standard error output. May contain both `\` and `/` as path separators. Is ignored when the standard error stream is redirected to the standard output stream.

`appendOut`

Whether to append to the stdout file (`true`) or overwrite it (`false`). Is ignored when standard output is not written to a file.

`appendErr`

Whether to append to the stderr file (`true`) or overwrite it (`false`). Is ignored if standard error output is not written to a file. Is also ignored when standard error stream is redirected to the standard output stream.

`errToOut`

Whether to redirect the standard error stream to the standard output stream (`true`) or use separate streams (`false`).

`ignoreNonZeroExitCode`

Whether to ignore non-zero exit codes (`true`) or consider them as errors (`false`).

Returns

The application exit code. See the application framework documentation for a description of the exit codes.

Runtime errors

- If the class loader can not be obtained.
- If the class can not be found or loaded.
- If the class is not an application framework application.
- If an appropriate constructor is not available.
- If an instance of the application class can not be constructed.
- If parsing of the command line arguments fails.
- If the application fails to execute due to its thread being interrupted.
- If the application exits with a non-zero exit code and non-zero exit codes are not ignored.

exec tool

```
tool int exec(string cmd, string... args,        string stdin = "", string stdout = "-
", string stderr = "-", bool appendOut = false, bool appendErr = false, bool errToOut
= false, bool ignoreNonZeroExitCode = false)
tool int exec(string cmd, list string args = [], string stdin = "", string stdout = "-
", string stderr = "-", bool appendOut = false, bool appendErr = false, bool errToOut
= false, bool ignoreNonZeroExitCode = false)
```

Executes a system command or external application as a sub process.

This tool should be avoided if possible, as it is in general not guaranteed to be cross platform and

machine independent. Commands and external applications may not be available on all systems, or may have different behavior on different systems.

Parameters

`cmd`

The name of the command or the absolute or relative local file system path of the external application to execute, using platform specific new line characters.

`args`

The command line arguments of the command or external application to execute. Each argument string is parsed to zero or more actual arguments. It is possible to use a single string with all arguments (where the arguments themselves are separated by spaces), comma separated strings for each of the arguments (each string has one argument), or a mix of those.

In argument strings, individual arguments are separated by whitespace (spaces, tabs, new lines, etc). The whitespace itself is ignored, and only serves as separation. To include whitespace in an argument, the argument (or a part of it), may be quoted, by putting it between double quotes (`"`) or single quotes (`'`). Characters may be escaped by prefixing them with a backslash (`\`). This is particularly useful for single/double quotes, and escape characters, and can also be used to escape spaces. Escapes work the same inside of quoted parts as they do outside of quoted parts.

`stdin`

Specify whether to have a standard input (stdin) stream and where the input comes from. Use `""` to not have a stdin stream, `"-"` to use the stdin stream of the ToolDef interpreter, or otherwise an absolute or relative local file system path of the file from which to read the standard input. May contain both `\` and `/` as path separators.

`stdout`

Specify whether to have a standard output (stdout) stream and where to write the standard output. Use `""` to not have a stdout stream, `"-"` to use the stdout stream of the ToolDef interpreter, or otherwise an absolute or relative local file system path of the file to which to write the standard output. May contain both `\` and `/` as path separators.

`stderr`

Specify whether to have a standard error (stderr) stream and where to write the standard error output. Use `""` to not have a stderr stream, `"-"` to use the stderr stream of the ToolDef interpreter, or otherwise an absolute or relative local file system path of the file to which to write the standard error output. May contain both `\` and `/` as path separators. Is ignored when the standard error stream is redirected to the standard output stream.

`appendOut`

Whether to append to the stdout file (`true`) or overwrite it (`false`). Is ignored when standard output is not written to a file.

`appendErr`

Whether to append to the stderr file (`true`) or overwrite it (`false`). Is ignored if standard error output is not written to a file. Is also ignored when standard error stream is redirected to the

standard output stream.

`errToOut`

Whether to redirect the standard error stream to the standard output stream (`true`) or use separate streams (`false`).

`ignoreNonZeroExitCode`

Whether to ignore non-zero exit codes (`true`) or consider them as errors (`false`).

Returns

The exit code resulting from the execution of the command or external application.

Runtime errors

- If parsing of the command line arguments fails.
- If ToolDef fails to execution the command or external application.
- If execution results in a non-zero exit code and non-zero exit codes are not ignored.

tooldef tool

```
tool int tooldef(string... args,        string stdin = "-", string stdout = "-",
string stderr = "-", bool appendOut = false, bool appendErr = false, bool errToOut =
false, bool ignoreNonZeroExitCode = false)
tool int tooldef(list string args = [], string stdin = "-", string stdout = "-",
string stderr = "-", bool appendOut = false, bool appendErr = false, bool errToOut =
false, bool ignoreNonZeroExitCode = false)
```

Executes a ToolDef script. Waits for that script to finish its execution before continuing with the current script.

The script is executed within the current Java interpreter, in a new thread, using a new ToolDef interpreter.

Parameters

`args`

The command line arguments for the ToolDef interpreter, including the path to the script to execute. Each argument string is parsed to zero or more actual arguments. It is possible to use a single string with all arguments (where the arguments themselves are separated by spaces), comma separated strings for each of the arguments (each string has one argument), or a mix of those.

In argument strings, individual arguments are separated by whitespace (spaces, tabs, new lines, etc). The whitespace itself is ignored, and only serves as separation. To include whitespace in an argument, the argument (or a part of it), may be quoted, by putting it between double quotes (")

or single quotes (`'`). Characters may be escaped by prefixing them with a backslash (`\`). This is particularly useful for single/double quotes, and escape characters, and can also be used to escape spaces. Escapes work the same inside of quoted parts as they do outside of quoted parts.

`stdin`

Specify whether to have a standard input (stdin) stream and where the input comes from. Use `""` to not have a stdin stream, `"-"` to use the stdin stream of the ToolDef interpreter, or otherwise an absolute or relative local file system path of the file from which to read the standard input. May contain both `\` and `/` as path separators.

`stdout`

Specify whether to have a standard output (stdout) stream and where to write the standard output. Use `""` to not have a stdout stream, `"-"` to use the stdout stream of the ToolDef interpreter, or otherwise an absolute or relative local file system path of the file to which to write the standard output. May contain both `\` and `/` as path separators.

`stderr`

Specify whether to have a standard error (stderr) stream and where to write the standard error output. Use `""` to not have a stderr stream, `"-"` to use the stderr stream of the ToolDef interpreter, or otherwise an absolute or relative local file system path of the file to which to write the standard error output. May contain both `\` and `/` as path separators. Is ignored when the standard error stream is redirected to the standard output stream.

`appendOut`

Whether to append to the stdout file (`true`) or overwrite it (`false`). Is ignored when standard output is not written to a file.

`appendErr`

Whether to append to the stderr file (`true`) or overwrite it (`false`). Is ignored if standard error output is not written to a file. Is also ignored when standard error stream is redirected to the standard output stream.

`errToOut`

Whether to redirect the standard error stream to the standard output stream (`true`) or use separate streams (`false`).

`ignoreNonZeroExitCode`

Whether to ignore non-zero exit codes (`true`) or consider them as errors (`false`).

Returns

The ToolDef interpreter exit code, i.e. the exit code of the script.

Runtime errors

- If parsing of the command line arguments fails.
- If the application fails to execute due to its thread being interrupted.
- If execution results in a non-zero exit code and non-zero exit codes are not ignored.

2.2.5. Built-in path tools

This page describes the built-in I/O tools:

- abspath
- basename
- chdir
- chfileext
- curdir
- dirname
- fileext
- hasfileext
- pathjoin
- scriptpath

abspath tool

```
tool string abspath(string path)
tool string abspath(string path, string workdir)
```

Returns an absolute local file system path, given an absolute or relative local file system path.

Parameters

`path`

The absolute or relative local file system path. May contain both `\` and `/` as path separators.

`workdir`

The optional absolute local file system path of the working directory against which to resolve relative paths. May contain both `\` and `/` as path separators. If not specified, relative paths are resolved against the current working directory.

Returns

The absolute local file system path. The path contains separators for the current platform.

basename tool

```
tool string basename(string path)
```

Returns the base name of the given absolute or relative local file system path. The base name of a file or directory is the name of that file or directory. In other words, returns the last part of the given path.

Parameters

`path`

The absolute or relative local file system path to the file or directory for which to return the base name. May contain both `\` and `/` as path separators. Must not end with `\` or `/`.

Returns

The base name.

Runtime errors

- If the path ends with `\` or `/`.

chdir tool

```
tool chdir(string path)
```

Changes the current working directory to the directory referred to by the given path.

Parameters

`path`

The absolute or relative local file system path to the new current working directory. Relative paths are resolved against the current working directory. May contain both `\` and `/` as path separators.

Runtime errors

- If the given path does not exist.
- If the given path refers to a file rather than a directory.
- If it can not be determined whether the given path refers to a file or a directory.

chfileext tool

```
tool string chfileext(string path, string? oldext = null, string? newext = null)
```

Modifies a path such that it ends with a new extension, removing an old extension if it exists.

Parameters

`path`

The absolute or relative local file system path to modify. May contain both `\` and `/` as path separators.

`oldext`

The old extension that can be removed (case insensitive, no . at the start). Use `null` to not remove an old extension.

`newext`

The new extension to use (case sensitive, no . at the start). Use `null` to not add a new extension.

Returns

The changed path, ending with the new extension (if any).

curdir tool

```
tool string curdir()
```

Returns the script execution's current working directory, as an absolute local file system path. The path contains separators for the current platform.

Returns

The script execution's current working directory.

dirname tool

```
tool string dirname(string path)
```

Returns the absolute directory path of the directory that contains the given file or directory.

Parameters

`path`

The absolute local file system path that refers to a file or directory. May contain both `\` and `/` as path separators. Must not end with `\` or `/`.

Returns

The absolute directory path of the directory that contains the given file or directory. May contain both `\` and `/` as path separators.

Runtime errors

- If the given path is not an absolute local file system path.
- If the given path ends with `\` or `/`.

fileext tool

```
tool string fileext(string path)
```

Returns the file extension of the given file, or `""` if the file has no file extension.

Parameters

`path`

The absolute or relative local file system path to the file. May contain both `\` and `/` as path separators.

Returns

The file extension, or `""`.

hasfileext tool

```
tool bool hasfileext(string path, string ext)
```

Does the given file have the given file extension?

Parameters

`path`

The absolute or relative local file system path to the file. May contain both `\` and `/` as path separators.

`ext`

The file extension to check for (case sensitive, no `.` at the start).

Returns

`true` if the file has the given file extension, `false` otherwise.

pathjoin tool

```
tool string pathjoin(string... paths)
tool string pathjoin(list string paths)
```

Joins paths together. If no paths are given, an empty string is returned. If one path is given, the path is returned.

Parameters

`paths`

: The paths to join together. The first path may be an absolute or relative local file system path. The remaining paths must be relative local file system paths. All paths may contain both `\` and `/` as path separators.

Returns

The joined path. The path contains separators for the current platform.

scriptpath tool

```
tool string scriptpath()
```

Returns the absolute local file system path to the script being executed. The path contains separators for the current platform.

Returns

The absolute local file system path to the script being executed.

2.2.6. Built-in file tools

This page describes the built-in I/O tools:

- cpfile
- cpdir
- diff
- exists
- filenewer
- filesize
- find
- isdir
- isfile
- mkdir

- mvfile
- mvdir
- readlines
- rmfile
- rmdir
- writefile

cpfile tool

```
tool cpfile(string source, string target, bool overwrite = false)
```

Copies a file from a source location to a target location.

Parameters

`source`

The absolute or relative local file system path of the source file. May contain both `\` and `/` as path separators.

`target`

The absolute or relative local file system path of the target file. May contain both `\` and `/` as path separators.

`overwrite`

Whether to overwrite the target file if it already exists.

Runtime errors

- If the source file does not exist.
- If the source is a directory rather than a file.
- If it can not be determined whether the source path refers to a file or a directory.
- If the target file exists and overwriting is not allowed.
- If the target file exists and overwriting is allowed, but the target refers to a directory rather than a file.
- If the target file exists and overwriting is allowed, but it can not be determined whether the target path refers to a file or a directory.
- If copying the file failed due to an I/O error.

cpdir tool

```
tool cpdir(string source, string target)
```

Copies a directory from a source location to a target location. All files and directories in the source directory are copied recursively.

If the operation fails, part of the operation may have already been performed.

Parameters

`source`

The absolute or relative local file system path of the source directory. All files and directories in the source directory are copied recursively. May contain both `\` and `/` as path separators.

`target`

The absolute or relative local file system path of the target directory. This is the directory in which the contents of the source directory are copied. The source directory itself is not copied, only the files and directories contained in the source directory. May contain both `\` and `/` as path separators.

Runtime errors

- If the source directory does not exist.
- If the source is a file rather than a directory.
- If it can not be determined whether the source path refers to a file or a directory.
- If the target directory already exists.
- If the target directory doesn't exist, but one of the ancestors is not a directory.
- If walking the directory (recursively) fails.
- If walking the directory (recursively) encounters a file system cycle (due to symbolic links).
- If a file or (sub-)directory could not be copied due to an I/O error.

diff tool

```
tool bool diff(string file1, string file2, string output = "-", bool missingAsEmpty =
false, bool warnOnDiff = false, bool failOnDiff = false)
```

Computes the differences between two files.

Parameters

`file1`

The absolute or relative local file system path of the first file. May contain both `\` and `/` as path

separators.

`file2`

The absolute or relative local file system path of the second file. May contain both `\` and `/` as path separators.

`output`

Specify whether/where to write a unified diff if the files differ. Use `""` to not write a unified diff, `"-"` to write the unified diff to the standard output stream (stdout), or otherwise an absolute or relative local file system path of the file to which to write the unified diff. May contain both `\` and `/` as path separators.

`missingAsEmpty`

Treat a missing first/second file as empty (`true`) or as an error (`false`).

`warnOnDiff`

Emit a warning for differing files (`true`) or not (`false`). If a warning is emitted to the standard error stream (stderr), the unified diff (if enabled) is printed first.

`failOnDiff`

Treat differing files as an error (`true`) or not (`false`). If an error is emitted, the unified diff (if enabled) and warning (if enabled) are printed first.

Returns

`true` if the files differ, `false` otherwise.

Runtime errors

- If either the first or second file doesn't exist and `missingAsEmpty` is disabled.
- If the first or second file is not a file but a directory.
- If it can not be determined whether the first or second path refers to a file or a directory.
- If an I/O error occurs.
- If the `out` file exists but is a directory rather than a regular file.
- If the `out` file does not exist but cannot be created.
- If the `out` file cannot be opened for writing for any other reason.
- If an I/O error occurs while writing to the `out` file.
- If the `out` file can not be closed.
- If the files differ and `failOnDiff` is enabled.

exists tool

```
tool bool exists(string path)
```

Does a file or directory with the given path exist?

Parameters

`path`

The absolute or relative local file system path of the file or directory. May contain both `\` and `/` as path separators.

Returns

`true` if the file or directory exists, `false` otherwise.

filenewer tool

```
tool bool filenewer(string path, string refpath,       bool allowNonExisting = false,
bool sameAsNewer = true)
tool bool filenewer(string path, string... refpaths,   bool allowNonExisting = false,
bool sameAsNewer = true)
tool bool filenewer(string path, list string refpaths, bool allowNonExisting = false,
bool sameAsNewer = true)
```

Checks whether a file is newer (was modified at a later date/time) than some reference file(s).

Parameters

`path`

The absolute or relative local file system path of the file for which to check whether it is newer than the reference file(s). May contain both `\` and `/` as path separators.

`refpath`, `refpaths`

The absolute or relative local file system path of the reference file(s). May contain both `\` and `/` as path separators.

`allowNonExisting`

Whether to allow the first file to not exist (`true`) or consider it an error if the first file does not exist (`false`).

`sameAsNewer`

Whether to treat files with the same last change date as being the same (`false`) or as newer (`true`).

Returns

`false` if the first file does not exist (if allowed by enabling `allowNonExisting`), if the first file is older than any the reference files, or if the first file has the same last change date as any of the reference files and `sameAsNewer` is disabled. `true` if the first file is newer than all of the reference files, if the first file has the same last change date as some the reference files and `sameAsNewer` is enabled and is

newer than all of the other reference files, or if the first file has the same last change date as all the reference files and `sameAsNewer` is enabled.

Runtime errors

- If the first file does not exist and `allowNonExisting` is disabled.
- If any of the reference files does not exist.
- If any of the files is a directory rather than a file.
- If for any of the paths it can not be determined whether the path refers to a file or a directory.
- If the last change date/time of any of the files can not be determined.

filesize tool

```
tool long filesize(string path, bool missingAsZero = false)
```

Returns the size of the file, in bytes.

Parameters

`path`

The absolute or relative local file system path of the file. May contain both `\` and `/` as path separators.

`missingAsZero`

Whether to return 0 if the file does not exist (`true`) or consider it an error if the file does not exist (`false`).

Returns

The size of the file in bytes, or `0` if the file is missing and `missingAsZero` is enabled.

Runtime errors

- If the file does not exist and `missingAsZero` is disabled.
- If the file is a directory rather than a file.
- If it can not be determined whether the path refers to a file or a directory.
- If the size of the file can not be determined due to an I/O error.

find tool

```
tool list string find(string path, string pattern = "*", bool recursive = true, bool
files = true, bool dirs = true)
```

Searches a directory for files and/or directories matching a pattern.

Parameters

`path`

The absolute or relative local file system path of the directory in which to search. The directory itself is never returned. May contain both `\` and `/` as path separators.

`pattern`

The pattern to use to match files/directories. Is a Java NIO glob pattern, that is matched against single file names or single directory names, and never against paths. Pattern `"*"` matches all files and directories.

`recursive`

Whether to recursively look in sub-directories.

`files`

Whether to match files.

`dirs`

Whether to match directories.

Returns

The local file system paths of the matched files and directories, relative against the given root directory from which searching started.

Runtime errors

- If the given directory is not found.
- If the given directory is a file rather than a directory.
- If the can not be determined whether the given path refers to a file or a directory.
- If the pattern is invalid.
- If walking the directory (recursively) fails.
- If walking the directory (recursively) encounters a file system cycle (due to symbolic links).
- If the operation fails due to an I/O error.

isdir tool

```
tool bool isdir(string path)
```

Does a directory with the given path exist?

Parameters

`path`

The absolute or relative local file system path of the directory. May contain both `\` and `/` as path separators.

Returns

`true` if the directory exists, `false` if it doesn't exist or is not a directory.

isfile tool

```
tool bool isfile(string path)
```

Does a file with the given path exist?

Parameters

`path`

The absolute or relative local file system path of the file. May contain both `\` and `/` as path separators.

Returns

`true` if the file exists, `false` if it doesn't exist or is not a file.

mkdir tool

```
tool mkdir(string path, bool force = false, bool parents = true)
```

Creates the given directory, and optionally its parents as needed.

Parameters

`path`

The absolute or relative local file system path of the directory to create. May contain both `\` and `/` as path separators.

`force`

Whether to skip creating the directory if it already exists (`true`) or fail if it already exists (`false`).

`parents`

Whether to allow creating parents as needed (`true`) or fail if the parent directory does not exist (`false`).

Runtime errors

- If the directory already exists and `force` is disabled.
- If creating the directory or any of its parents fails, due to an I/O error.

mvfile tool

```
tool mvfile(string source, string target, bool overwrite = false)
```

Moves a file from a source location to a target location. This can be used to rename a file and/or move it to another directory.

Parameters

`source`

The absolute or relative local file system path of the source file. May contain both `\` and `/` as path separators.

`target`

The absolute or relative local file system path of the target file. May contain both `\` and `/` as path separators.

`overwrite`

Whether to overwrite the target file if it already exists.

Runtime errors

- If the source file does not exist.
- If the source is a directory rather than a file.
- If it can not be determined whether the source path refers to a file or a directory.
- If the target file exist and overwriting is not allowed.
- If the target file exists and overwriting is allowed, but the target refers to a directory rather than a file.
- If the target file exists and overwriting is allowed, but it can not be determined whether the target path refers to a file or a directory.
- If moving the file fails due to an I/O error.

mvdir tool

```
tool mvdir(string source, string target)
```

Moves a directory from a source location to a target location. The directory and all files and directories in it are moved recursively.

The operation is implemented as a copy from source to target, followed by a remove of the source. If the operation fails, part of the operation may have already been performed.

Parameters

`source`

: The absolute or relative local file system path of the source directory. The directory itself and all files and directories in it are moved recursively. May contain both `\` and `/` as path separators.

`target`

: The absolute or relative local file system path of the target directory. This is the directory into which the contents of the source directory are moved. May contain both `\` and `/` as path separators.

Runtime errors

- If the source directory does not exist.
- If the source is a file rather than a directory.
- If it can not be determined whether the source path refers to a file or a directory.
- If the target directory already exists.
- If the target directory doesn't exist but one of the ancestors is not a directory.
- If walking the directory (recursively) fails.
- If walking the directory (recursively) encounters a file system cycle (due to symbolic links).
- If a file or (sub-)directory can not be copied or removed, due to an I/O error.

readlines tool

```
tool list string readlines(string path)
```

Read lines of text from a file.

Parameters

`path`

: The absolute or relative local file system path of the file to read. May contain both `\` and `/` as path separators.

Returns

The lines of text from the file.

Runtime errors

- If the file does not exist.
- If the path refers to a directory rather than a file.
- If it can not be determined whether the path refers to a file or a directory.
- If for some other reason can not be opened for reading.
- If an I/O error occurs.
- If the file can not be closed.

rmfile tool

```
tool bool rmfile(string path, bool force = false)
```

Removes a file.

Parameters

`path`
: The absolute or relative local file system path of the file. May contain both `\` and `/` as path separators.

`force`
: Whether to ignore non-existing files (`true`) or consider it an error (`false`).

Returns

`true` if the file was removed, `false` if it could not be removed because it did not exist and force is enabled.

Runtime errors

- If the file does not exist and `force` is disabled.
- If the file is a directory rather than a file.
- If it can not be determined whether the path refers to a file or a directory.
- If an I/O error occurs.
- If the file can not be removed for some other reason.

rmdir tool

```
tool bool rmdir(string path, bool force = false)
```

Removes a directory, recursively.

Parameters

`path`

The absolute or relative local file system path of the directory. May contain both `\` and `/` as path separators.

`force`

Whether to ignore non-existing directories (`true`) or consider it an error (`false`).

Returns

`true` if the directory was removed, `false` if it could not be removed because it did not exist and `force` is enabled.

Runtime errors

- If the directory does not exist and `force` is disabled.
- If the directory is a file rather than a directory.
- If it can not be determined whether the path refers to a file or a directory.
- If an I/O error occurs.
- If walking the directory (recursively) fails.
- If walking the directory (recursively) encounters a file system cycle (due to symbolic links).
- If the directory or one of its sub-files or sub-directories can not be removed for some other reason.

writefile tool

```
tool writefile(string path, string text,       bool append = false)
tool writefile(string path, list string lines, bool append = false)
```

Writes text to a file.

Parameters

`path`

The absolute or relative local file system path of the file. May contain both `\` and `/` as path separators.

`text`, `lines`

The text or lines of text to write to the file.

`append`

Whether to append the lines text to the file if it already exists (`true`), or overwrite the file if it already exists (`false`).

Runtime errors

- If the path exists but is a directory rather than a regular file.
- If the file does not exist, but cannot be created.
- If the file can not be opened for writing for any other reason.
- If writing to the file fails due to an I/O error.
- If closing the file fails.

3. Tooling

Below you can find general information on how to use the ToolDef tooling, both on the command line, and in the Eclipse IDE.

To start using the ToolDef tooling, create a `.tooldef` file. See the language reference documentation for more information on the syntax.

3.1. Command line

To execute a ToolDef script on the command line, use the `tooldef` executable. To start executing a ToolDef script, enter something like this on the command line:

```
tooldef some_script.tooldef
```

Additional options are available, to influence how the script is executed. For details, execute the ToolDef interpreter with the `--help` or `-h` command line option:

```
tooldef --help
```

3.2. Eclipse IDE

To execute a ToolDef script in the Eclipse IDE, right click the file or an open text editor for the script, and choose **Execute ToolDef**. Alternatively, choose **Execute ToolDef...** to first shown an option dialog, where several options that influence how the script is executed, before actually executing the script.

It is also possible to start executing a script by pressing `F10`, while a `.tooldef` file is selected or an open text for such a file has the focus. Finally, clicking the corresponding toolbar icon (🔧) has the same effect.

Execution of ToolDef script can be manually terminated by means of the termination features of the *Applications* view.

4. ToolDef release notes

The release notes for the releases of ToolDef and the associated tools, as part of the Eclipse ESCET project, are listed below in reverse chronological order.

See also the Eclipse ESCET toolkit release notes covering those aspects that are common to the various Eclipse ESCET tools.

4.1. Version 0.1

The first release of ToolDef as part of the Eclipse ESCET project. This release is based on the initial contribution by the Eindhoven University of Technology (TU/e).

Most notable changes compared to the last TU/e release:

- We no longer use separate language and tool versions. The `.tooldef2` file extension has been changed to `.tooldef` as part of this change.

5. Legal

Index