### 16.1.2 Predefined functions

TTCN‑3 contains a number of predefined (built-in) functions that need not be declared before use. These are summarized in table 1.

Table 1: List of TTCN‑3 predefined functions

| Category | Function | Keyword |
| --- | --- | --- |
| **Conversion functions**  | Convert **integer** value to **charstring** value | **int2char** |
| Convert **integer** value to **universal** **charstring** value | **int2unichar** |
| Convert **integer** value to **bitstring** value | **int2bit** |
| Convert **integer** value to enumerated value | **int2enum** |
| Convert **integer** value to **hexstring** value | **int2hex** |
| Convert **integer** value to **octetstring** value | **int2oct** |
| Convert **integer** value to **charstring** value  | **int2str** |
| Convert **integer** value to **float** value | **int2float** |
| Convert **float** value to **integer** value | **float2int** |
| Convert **charstring** value to **integer** value | **char2int** |
| Convert **charstring** value to **octetstring** value | **char2oct** |
| Convert **universal charstring** value to **octetstring** value | **unichar2oct** |
| Convert **universal** **charstring** value to **integer** value | **unichar2int** |
| Convert **bitstring** value to **integer** value | **bit2int** |
| Convert **bitstring** value to **hexstring** value | **bit2hex** |
| Convert **bitstring** value to **octetstring** value | **bit2oct** |
| Convert **bitstring** value to **charstring** value | **bit2str** |
| Convert **hexstring** value to **integer** value | **hex2int** |
| Convert **hexstring** value to **bitstring** value | **hex2bit** |
| Convert **hexstring** value to **octetstring** value | **hex2oct** |
| Convert **hexstring** value to **charstring** value | **hex2str** |
| Convert **octetstring** value to **integer** value | **oct2int** |
| Convert **octetstring** value to **bitstring** value | **oct2bit** |
| Convert **octetstring** value to **hexstring** value | **oct2hex** |
| Convert **octetstring** value to **charstring** value | **oct2str** |
| Convert **octetstring** value to **charstring** value, version II | **oct2char** |
| Convert **octetstring** value to **universal charstring** value | **oct2unichar** |
| Convert **charstring** value to **integer** value  | **str2int** |
| Convert **charstring** value to **hexstring** value | **str2hex** |
| Convert **charstring** value to **octetstring** value | **str2oct** |
| Convert **charstring** value to **float** value | **str2float** |
| Convert enumerated value to **integer** value | **enum2int** |
| Convert value or template to **universal charstring** value | **any2unistr** |
| **Length/size functions** | Return the length of a value or template of any string type, **record of**, **set of** or **array** | **lengthof** |
| Return the number of elements in a value or a template of a **record** or **set** | **sizeof** |
| **Presence checking functions** | Determine if an optional field in a **record** or **set** value or template is present | **ispresent** |
| Determine which choice has been selected in a **union** value or template | **ischosen** |
| Determine if a template evaluates to a concrete value | **isvalue** |
| Determine if a template is uninitialized or not | **isbound** |
| Determine if a template contains certain matching mechanism | **istemplatekind** |
| **String/list handling functions** | Returns part of the input string matching the specified pattern group within a character pattern | **regexp** |
| Returns the specified portion of the input string/list value or template | **substr** |
| Replaces a substring of a string with or inserts the input string into a string, and similarly for lists | **replace** |
| **Codec functions** | Encode a value into a bitstring | **encvalue** |
| Decode a bitstring into a value | **decvalue** |
| Encode a value into a universal charstring | **encvalue\_unichar** |
| Decode a universal charstring into a value | **decvalue\_unichar** |
| **Other functions** | Generate a random float number | **rnd** |
| Returns the name of the currently executing test case | **testcasename** |
|  |  |

***Syntactical Structure***

**int2char** "(" *SingleExpression* ")" |

**int2unichar** "(" *SingleExpression* ")" |

**int2bit** "(" *SingleExpression* "," *SingleExpression* ")" |

**int2enum** "(" *SingleExpression* "," *SingleExpression* ")" |

**int2hex** "(" *SingleExpression* "," *SingleExpression* ")" |

**int2oct** "(" *SingleExpression* "," *SingleExpression* ")" |

**int2str** "(" *SingleExpression* ")" |

**int2float** "(" *SingleExpression* ")" |

**float2int** "(" *SingleExpression* ")" |

**char2int** "(" *SingleExpression* ")" |

**char2oct** "(" *SingleExpression* ")" |

**unichar2int** "(" *SingleExpression* ")" |

**unichar2oct** "(" *SingleExpression* [, *SingleExpression*] ")" |

**bit2int** "(" *SingleExpression* ")" |

**bit2hex** "(" *SingleExpression* ")" |

**bit2oct** "(" *SingleExpression* ")" |

**bit2str** "(" *SingleExpression* ")" |

**hex2int** "(" *SingleExpression* ")" |

**hex2bit** "(" *SingleExpression* ")" |

**hex2oct** "(" *SingleExpression* ")" |

**hex2str** "(" *SingleExpression* ")" |

**oct2int** "(" *SingleExpression* ")" |

**oct2bit** "(" *SingleExpression* ")" |

**oct2hex** "(" *SingleExpression* ")" |

**oct2str** "(" *SingleExpression* ")" |

**oct2char** "(" *SingleExpression* ")" |

**oct2unichar** "(" *SingleExpression* [, *SingleExpression*] ")" |

**str2int** "(" *SingleExpression* ")" |

**str2hex** "(" *SingleExpression* ")" |

**str2oct** "(" *SingleExpression* ")" |

**str2float** "(" *SingleExpression* ")" |

**enum2int** "(" *SingleExpression* ")" |

**any2unistr** "(" *SingleExpression* ")" |

**lengthof** "(" *TemplateInstance* ")" |

**sizeof** "(" *TemplateInstance* ")" |

**ispresent** "(" *TemplateInstance* ")" |

**ischosen** "(" *TemplateInstance* ")" |

**isvalue** "(" *TemplateInstance* ")" |

**isbound** "(" *TemplateInstance* ")" |

**istemplatekind** "(" *TemplateInstance* "," *TemplateInstance* ")" |**regexp** "(" *TemplateInstance*"," *TemplateInstance*"," *SingleExpression* ")" |

**substr** "(" *TemplateInstance* "," *SingleExpression* "," *SingleExpression* ")" |

**replace** "(" *SingleExpression* "," *SingleExpression* "," *SingleExpression* "," *SingleExpression* ")" |

**encvalue** "("*TemplateInstance*")" |

**decvalue** "("*SingleExpression*","*SingleExpression*")" |

**encvalue\_unichar** "("*TemplateInstance* **[,** *SingleExpression***]** ")" |

**decvalue\_unichar** "("*SingleExpression*","*SingleExpression* **[,** *SingleExpression***]**")" |

**rnd** "(" [ *SingleExpression* ] ")" |

**testcasename** "()"")" |

***Semantic Description***

The description of predefined functions is given in annex C.

***Restrictions***

In addition to the general static rules of TTCN‑3 given in clause **Error! Reference source not found.**, the following restrictions apply:

a) When a predefined function is invoked:

1) the number of the actual parameters shall be the same as the number of the formal parameters; and

2) each actual parameter shall evaluate to an element of its corresponding formal parameter's type; and

3) all actual in and inout parameters shall be initialized with the following exceptions:

* the actual in and inout parameter passed to the predefined functions isvalue, ischosen, ispresent and isbound may be uninitialized or even contain non-evaluable reference expressions
* the any\_string\_or\_sequence\_type parameters of the functions lengthof, substr and replace may be partially initialized.
* the invalue parameter of the ttcn2unistr function may be uninitialized, partially or completely initialized.

b) Restrictions on invoking functions from specific places are described in clause **Error! Reference source not found.**.

***Examples***

 **var hexstring** h:= **bit2hex** ('111010111'B);

 **var octetstring** o:= **substr** ('01AB23CD'O, 1, 2);

## C.1.33 Value or template to universal charstring

 **any2unistr**(**in template** any\_type invalue) **return universal charstring**

This function converts the content of a value or template to a single **universal charstring**. The resulting **universal charstring** is the same as the string produced by the log operation containing the same operand as the one passed to the **any2unistr** function. The value or template passed as a parameter to the **any2unichar** function may be uninitialized, partially or completely initialized.

The general error causes in clause 16.1.2 apply.

EXAMPLE:

 **var integer** v\_int1 :=5, v\_int2;

 **var template integer vmw\_int1** :=?;

 **var template integer vmw\_int2** :=-1 ifpresent;

 **var universal charstring** v\_chr1, v\_chr2, v\_chr3, v\_chr4;

 v\_chr1 := **any2unistr(v\_int1)**; // after the assignment v\_chr1 will be "5"

 v\_chr2 := **any2unistr(v\_int2)**; // after the assignment v\_chr2 will be "UNINITIALIZED"

 v\_chr3 := **any2unistr(vmw\_int1)**; // after the assignment v\_chr3 will be "?"

 v\_chr4 := **any2unistr(vmw\_int2)**; // after the assignment v\_chr3 will be "-1 ifpresent"

## C.3.5 Matching mechanism detection

 is**templatekind** (**in template** any\_type invalue, **in** **charstring** kind) **return boolean**

This function allows to examine what kind of matching mechanism a template contains.

If the searched matching mechanism is matching a specific value (B.1.1), a matching mechanism instead of values (B.1.2) or matching character pattern (B.1.5), the function shall return true if the content of the invalue parameter is of the same kind.

If the searched matching mechanism is a matching mechanism inside values (B.1.3), the function shall return true if the template in the invalue parameter contains this kind of matching mechanism on the first level of nesting.

If the searched matching mechanism is a matching attribute (B.1.4), the function shall return true if the template in the invalue parameter has this kind of matching attribute attached to it directly (i.e. it doesn’t count if the attribute is attached to a field of invalue at any level of nesting).

In all other cases the function returns false.

***Restrictions***

In addition to the general error causes given in clause 16.1.2, the following restrictions apply:

1. The kind parameter shall be one of the strings listed in the table C.1.34.

Table C.1.34: Values of kind parameter

|  |  |
| --- | --- |
| **Value of kind parameter** | **Searched matching mechanism** |
| "value" | Specific value (see clause B.1.1) |
| "list" | Template list (see clause B.1.2.1) |
| "complement" | Complemented template list (see clause B.1.2.2) |
| "AnyValue", "?" | Any value (see clause B.1.2.3) |
| "AnyValueOrNone", "\*" | Any value or none (see clause B.1.2.4) |
| "range" | Value range (see clause B.1.2.5) |
| "superset" | SuperSet (see clause B.1.2.6) |
| "subset" | SubSet (see clause B.1.2.7) |
| "omit" | Omit (see clause B.1.2.8) |
| "@encoded" | Encoded value (see clause B.1.X.X) |
| "AnyElement" | Any element (see clause B.1.3.1) |
| "AnyElementsOrNone" | Any number of elements or none (see clause B.1.3.2) |
| "permutation" | Permutation (see clause B.1.3.3) |
| "length" | Length restriction (see clause B.1.4.1) |
| "ifpresent" | The IfPresent indicator (see clause B.1.4.2) |
| "pattern" | Matching character pattern (see clause B.1.5) |

NOTE: Clause E.2.2.4 includes the type definition TemplateKind and the constant definitions VALUE, LIST, COMPLEMENT, ANY\_VALUE, ANY\_VALUE\_OR\_NONE, RANGE, SUBSET, SUPERSET, OMIT, ENCODED, ANY\_ELEMENT, ANY\_ELEMENTS\_OR\_NONE, PERMUTATION, LENGTH, IFPRESENT, PATTERN. It is recommended to use the istemplatekind function in combination with this type and these constants to ease the checking of correct usage and to improve the readability of test specs.

1. Calling the istemplatekind function with a different second parameter than stated in table C.1.34 shall lead to an error.

EXAMPLE:

**type record of integer** RoI;

...

**var template integer** vt\_1 := ?, vt\_2 := (0,1,2) ifpresent;

**var template** RoI vt\_3:= { **permutation**(1, 2, 3), ? };
**var boolean** v\_res;
...
v\_res := **istemplatekind**(vt\_1, "AnyValue"); // true
v\_res := **istemplatekind**(vt\_1, "AnyValueOrNone"); // false
v\_res := **istemplatekind**(vt\_2, "complement"); // false
v\_res := **istemplatekind**(vt\_2, "list"); // true
v\_res := **istemplatekind**(vt\_2, "ifpresent"); // true

v\_res := **istemplatekind**(vt\_3, "permutation"); // true

v\_res := **istemplatekind**(vt\_3, "AnyElement"); // true

### E.2.2.4 Template kinds of TTCN-3 objects

Type and constants defined in this clause support the secure usage of the predefined istemplatekind function defined in clause **Error! Reference source not found.**.

The type definition for this type is:

 **type** **charstring** TemplateKind ("value", "list", "complement", "AnyValue", "?", "AnyValueOrNone", "\*", "range", "subset", "superset", "omit", "@encoded", "AnyElement", "AnyElementsOrNone", "permutation", "length", "ifpresent", "pattern");

Useful constant definitions for working with template kinds are:

 **const** TemplateKind VALUE := "value";

 **const** TemplateKind LIST := "list";

 **const** TemplateKind COMPLEMENT := "complement";

 **const** TemplateKind ANY\_VALUE := "AnyValue";

 **const** TemplateKind ANY\_VALUE\_OR\_NONE := "AnyValueOrNone";

 **const** TemplateKind RANGE := "range";

 **const** TemplateKind SUBSET := "subset";

 **const** TemplateKind SUPERSET := "superset";

 **const** TemplateKind OMIT := "omit";

 **const** TemplateKind ENCODED := "@encoded";

 **const** TemplateKind ANY\_ELEMENT := "AnyElement";

 **const** TemplateKind ANY\_ELEMENTS\_OR\_NONE := "AnyElementsOrNone";

 **const** TemplateKind PERMUTATION := "permutation";

 **const** TemplateKind LENGTH := "length";

 **const** TemplateKind IFPRESENT := "ifpresent";

 **const** TemplateKind PATTERN := "pattern";