REQTIFY™

Programming Guide

3DEXPERIENCE

DASSAULT SYSTEMES | IF WE ask the right questions we can change the world.

Table of Contents

Introduction

You can manage Reqtify objects within the tool using the OTScript language or outside the tool in VB or VBA.

OTScript Overview

Semantics

Like most object-oriented languages, OTScript is a class-based object-oriented language.

This means that classes are repositories for the description and the behavior of objects. Every object belongs to a class and each class is a refinement of a root class called *Any_*. These aspects will be detailed along this report.

Beyond its classical OO aspects, OTScript is also sequence-oriented: data items are held by sequences which are kinds of collections. Some facilities are provided by the language to avoid hard sequence manipulations: for example, it is possible to iterate upon them or to apply a function on each element of them without involving the use of classical nested loop structures.

In a sequence several same scalar (Boolean, Integer, Real, String, Image, RichText, Class) values can be present whereas only entities can be present only once.

OTScript is also a strongly typed language. This means that the type of each expression must be known at compile-time. Some type identifiers are provided by the language to permit type declaration.

Notation and Terminology

Objects as Instances of Classes

Objects are distinct entities characterized by their own internal state containing attributes and/or roles, and their class. An attribute is valued by a primitive type such as a Boolean, a Number, a String or eventually another Object class, while a role is a part of an association, which is a uni-directional or bi-directional reference between two entities of the language. A class is a set of objects (its instances) that share the same behavior. Behavior is stored into the classes which also provide information to create new instances.

Inheritance and Concept of Element

OTScript allows simple inheritance: starting from a single existing class C, it is possible to define a new class S as a subclass of C. Instances of S inherits any attributes and any behavior defined in C. S may also define additional attributes and behavior particular to its instances or redefine some inherited behavior.

Given a class C, elements of C include instances of C and instances of every subclass of C.

Messages, Methods, Rules and Events

Interactions with objects are performed by sending them a message such as in JAVA, i.e. by using a dot notation. When an object receives a message, a method associated with this message is executed. The result of the execution is returned as an answer of the object to the message.

A rule is a specific method called to verify a condition. When condition is not met, an error is raised.

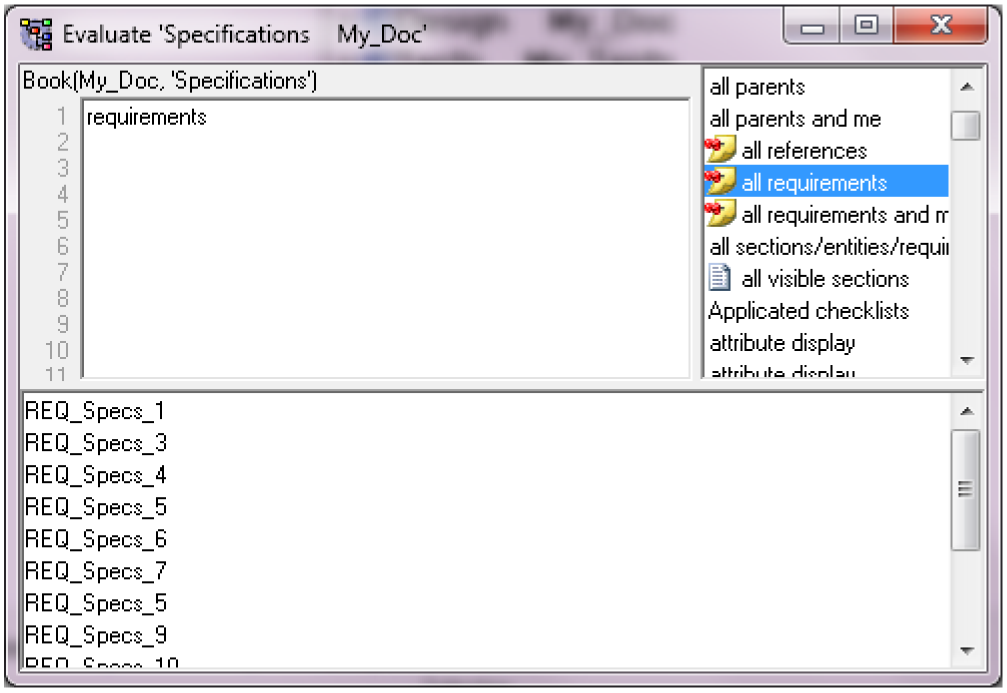
An event is defined by a specific prefix. When an event occurs, methods having event prefix are automatically called.

Instance Creation

Instance creation is performed by an evaluation of the *NEW* operator (that is not a message) followed by the name of the class to instantiate. This operation only returns a new instance of the class without initializing it. Initialization occurs when a constructor (in fact, a particular method) is applied to the newly created instance.

Evaluator Window

OTScript code can be tested using the evaluator window.



Book(My_Doc, 'Specifications')

```
requirements
```

```
REQ_Specs_1
REQ_Specs_3
REQ_Specs_4
REQ_Specs_5
REQ_Specs_6
REQ_Specs_7
REQ_Specs_5
REQ_Specs_9
```

This window is split into 4 areas:

- Receiver: this component shows the current receiver. It corresponds to the object selected in Reqtify UI when the evaluator is opened
- Source Code: in this text field, OTScript source code can be entered to be evaluated. Each modification of text is immediately evaluated. When F1 key is pressed, Reqtify opens *Programming Guide* on the current selection (operator, function, or method).
- Function list: list of main functions those that can be called on the current receiver.
- Result: this view can show different information:
    - Nothing when nothing is present in the source code text view.
    - Errors when the source code contains errors.
    - Text or rich text when the result of evaluation is a text.
    - List of objects when the result of evaluation is multiple. In this case a *Navigate* menu item is available to select in the main Reqtify window the selected element.
    - Image when the result of evaluation is an image.

General Syntax

Predefined Variables

THIS

Receiver of current method.

THIS.name

EACH

Current receiver.

("The" $+ "black" $+ "cat").(STRCNT(EACH)) ⇒ 3 $+ 5 $+ 3

EACHINDEX

Current receiver index.

("The" $+ "black" $+ "cat").{ STR(EACHINDEX) ; } ⇒ 1 $+ 2 $+ 3

Control Structures

TMP var { : <classe> |:= <expr> }

Defines a temporary variable. Temporary variable definition can appear anywhere like any other instruction.

TMP n:= 1

"<string1>"|"<string2>"

Defines a multiple language literal.

"The black cat"|"Le chat noir"

{ <expr1> ; <expr2> ; ... ; exprN }

Block definition returning *exprN* result.

{ TMP n:= 0 ; n:= n + 1 ; }

<expr1>.<expr2>

Evaluates expression *expr2* on *expr1*.

(1 $+ 2).{ STR(EACH) ; } ⇒ "1" $+ "2"

<expr1>.*(<expr2>)

Evaluates recursively *expr2* on *expr1*.

<expr>[<cond>]

Evaluates expression *expr* and only selects elements verifying condition *cond*.

(1 $+ 2 $+ 3 $+ 4)[EACH > 2] ⇒ 3 $+ 4

<expr> IF <val1>: <code1> IF <val2>:<code2> … ELSE <codeN>

Evaluates expression *expr* and executes corresponding code.

STR(1) IF "1": "one" IF "2": "two" ELSE: "" ⇒ "one"

<expr1> CATCH <expr2>

Evaluates expression *expr1* and evaluates expr2 on exception.

{ RAISE("Error"); } CATCH { INFO("Exception") }

Method Definition

ATTRIBUTE <class>.name <–options> : <type>

Define an attribute named *name* of type *type* on class *class*.

<-options>:

- *transient*: attribute will not be stored, in particular for connector class

METHOD <class>.<name>(<parameters>) <-options>: <code> [LABEL <label>]

Defines a method named *name* on class *class*. Returned type is type of last instruction of code *code*.

<-options>:

- *doc*: method available for report generation
- *enablewhen* [<condition>]: Enables the menu item and/or the toolbar button when <condition> result is TRUE
- *icon* ["icon name"]: Defines an icon for the menu item and/or the toolbar
- *menu*: Displays a menu item in Tools menu by default to run the method
- *menuafter/menubefore [<item hierarchy>]*: Position a menu item depending on another one referenced by its label hierarchy
- *menusep:* a separator is displayed before in menu
- *mult*: OLE collection with *item* property (Visio, Rhapsody, QC, …)
- *mult0*: OLE collection with 0-based index and *item* property

- *mult2:* OLE collection with *item* method (Word, Excel, PowerPoint, …)
- *mult3:* OLE collection with *item* method and *length* property
- *mult4:* OLE collection with *item* property
- *mult5:* OLE collection with 0-based index and *item* method (Shell)
- *mult6:* OLE collection with 0-based index and *getAt* method (EA)
- *noa1, noa2:* 1 or 2 optional arguments. *noa1* makes optional only the last argument, *noa2* makes optional the last two arguments.
- *showwhen* [<condition>]: Displays the menu item and/or the toolbar button when <condition> result is TRUE
- *tool*: Displays an icon in the toolbar to run the method

RULE <class>.<name>{(arg : Type)} [<options>] :
<rule code>
LABEL <label> [EXPLAIN <explain code>]

Defines a rule named *name* on class class. Returned type must be Boolean.

<options>:

- arg : this argument is created once using the NEW operator and is passed in parameter to each evaluation of the rule.
- -e{1|2|3|4|5}: Error level:
  - 1: Error (red),
  - 2: Warning (orange),
  - 3: Information (green),
  - 4: Information (black),
  - 5: Information (hidden)
- -elementtype ["<element type name>"]: rule applies only to specific element type
- -documenttype ["<document type name>"]: rule applies only to specific document type
- -converttool ["<convert tool>"]: rule applies only to specific convert tool
- -documentcategory ["<document category>"]: rule applies only to specific document category

<explain code>: this code is executed on error and must return a *String* that will appear near the error in Reqtify GUI.

Example

Following rule enables you to redefine the *uncovered* rule to depend on a weight defined as an attribute on cover link. When sum of weights is greater or equal to 100, the requirement is considered as covered.

```
RULE Requirement.uncovered : {
  $NO(document.coveringDocuments) OR
    $SUM(INT(coverLinks.attributeValue("Weight"))) >= 100
} LABEL "Uncovered requirement"|"Exigence non couverte";
```

OLE Access

Reqtify can access other OLE APIs.

Declaration

To access other OLE APIs, library functions and properties must be declared.

Here is an example for Word:

```
PACKAGE Word;                // Library declaration

CLASS Application: OleAutomationObject;   // Declare main class
CLASS Document: OleAutomationObject;
CLASS Selection: OleAutomationObject;

ATTRIBUTE Application.visible: Boolean;     // Declare a boolean attribute
ATTRIBUTE Application.documents -mult2: Document;      // Return a collection
ATTRIBUTE Application.selection: Selection;
ATTRIBUTE Application.activeDocument: Document;
ATTRIBUTE Document.name: String;

METHOD Application.activate(): VOID(Application);          // Function declaration
METHOD Document.save(): VOID(Document);
METHOD Document.activate(): VOID(Document);

PACKAGE DEFAULT; // End of package declaration. Return to default package
```

Usage

GETOLEAUTOMATIONOBJECT(class: CLASS, filename: String): Entity_

Searches a particular object of class *class* in Windows system and returns it or VOID if not found.

```
GETOLEAUTOMATIONOBJECT(CLASS(Word.Document), "c:\doc.doc");
```

NEW(class: CLASS): Entity_

Creates a new object of class *class*.

```
NEW(Word.Application)
```

Reqtify as a Server

Using VB/VBA

Installation

This operation is automatically realized during the installation step. But it is also possible to realize this task manually by opening an administrator CMD window and by typing the instruction:

```
reqtify –regserver
```

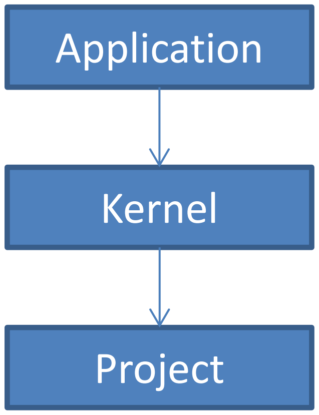
To delete the record of the server, type:

```
reqtify –unregserver
```

The reqtify.tlb file is provided next to the executable file.

Entry Points

Some Reqtify elements can be managed using VB.



Use *CreateObject* and *GetObject* functions to create or recover an object.

Specificities

All the basic functions are available and are listed in the reqtify.tlb file. This file is present in *bin.w32* directory.

Each interface is associated with another one with a 's' at the end. This second "plural" interface allows you to work on list.

All the OTScript functions are also available in VB even if there are not listed in the tlb file.

SOAP/JSON-RPC server

Reqtify can be run as a SOAP or JSON-RPC server.

Definition

To define a server, just define a method without any argument on *Kernel* class. This method shall return an instance of an OTScript class. Then on this server class, all functions are available as WebServices.

Execution

To run Reqtify as a HTTP server:

```
reqtify –http <port> -logfile <log file>
```

In this case, Reqtify has no UI and it is blocked waiting for client request.

To access Reqtify server, use following URL:

http://<host>:<port>/<kernel function>

To access Reqtify server using OTScript code use following code:

```
GETOBJECT(<Client class>, "http://<host>:<port>/<kernel function>")
```

To end Reqtify server, kill Reqtify process or use following URL:

http://<host>:<port>/quit

Example

This example defines a server. This server offers 3 functions: *test*, *increment* and *test2*.

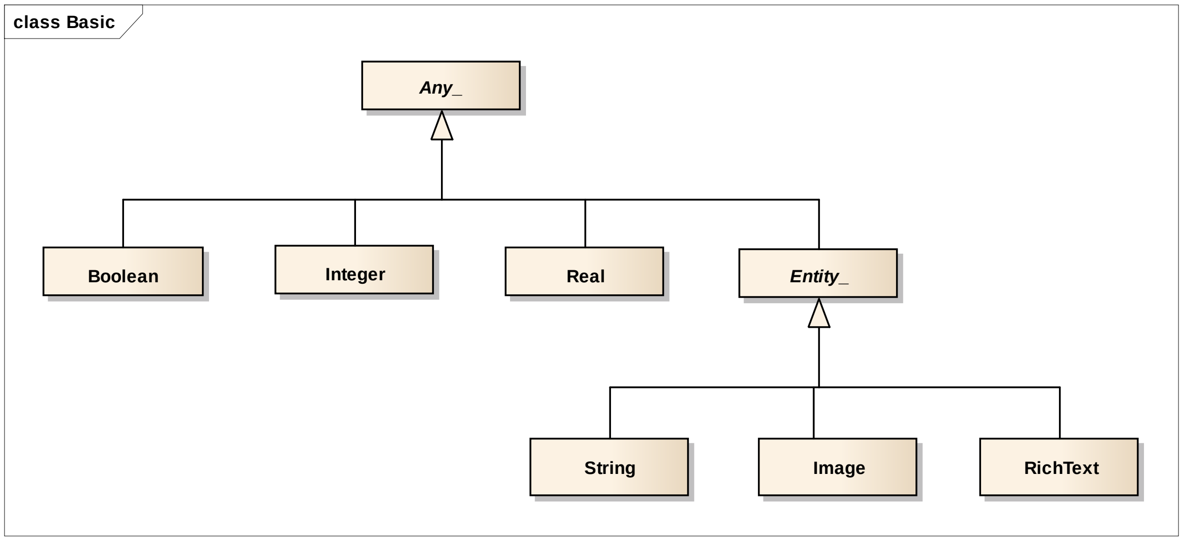
```
PACKAGE Test;


CLASS Server : Entity_;

CLASS Object : Entity_;


ATTRIBUTE Object.string : String;

ATTRIBUTE Object.object : Object;
ATTRIBUTE Server.count : Integer;

METHOD Server.test() : {
        "abc";
};

METHOD Server.increment() : {
        count := count + 1;
};

METHOD Server.test2() : {
        TMP o := NEW(Object);
        o.string := "abc";
        o.object := o;
        o;
};


PACKAGE DEFAULT;

METHOD Kernel.testServer() : {      // Server creation
        TMP server := NEW(Test.Server);
        server.count := 0;
        server;
};
```

DDE Server

By default a DDE server is launched by Reqtify. The service name is *REQTIFY*. DDE Execute messages containing OTScript code can be sent to Reqtify. The receiver of the message is the application.

Basic Classes, Operations and Functions

Schema



Any_ Class

Any_

*Any_* class is an abstract class super class of all other classes.

Boolean: Any_

Boolean class inherits from *Any_* class. Only two instances of this class are available: *TRUE* and *FALSE*.

Boolean operations

bool1 AND bool2: Boolean

Tests if *bool1* and *bool2* are true. *bool2* is not evaluated if *bool1* is false.

TRUE AND FALSE ⇒ FALSE

bool1 OR bool2: Boolean

Tests if *bool1* or *bool2* is true. *bool2* is not evaluated if *bool1* is true.

TRUE OR FALSE ⇒ TRUE

NOT(bool: Boolean): Boolean

Returns negation of *bool*.

NOT(TRUE) ⇒ FALSE

bool1 = bool2: Boolean

Tests if *bool1* = *bool2* is true.

TRUE = FALSE ⇒ FALSE

STR(bool: Boolean): String

Returns string representing Boolean *bool*.

STR(TRUE) ⇒ "true"

Integer: Any_

Integer class inherits from *Any_* class. Integer class has at least 32 bits.

0, 1, 10, -9

Integer operations

int1 { + | - | * | / } int2: Integer

Applies operation on integers *int1* and *int2*.

100 + 23 ⇒ 123

int1 { = | /= | < | > | <= | >= } int2: Boolean

Compares integers *int1* and *int2*.

100 < 23 ⇒ FALSE

REAL(int: Integer): Real

Converts integer *int* into real.

REAL(123) ⇒ 123.0

STR(int: Integer): String

Converts integer *int* into string.

STR(123) = "123"

Integer functions

$SUM(integers): Integer

Returns the sum of all integers in *integers* collection.

$SUM(100 $+ 23) ⇒ 123

Real: Any_

Real class inherits from *Any_* class.

0.0, 1.5, -0.5, 1.6e5, 2.7e-23,-1e-5

Real operations

real1 { + | - | * | / } real2: Real

Applies operation on real *real1* and *real2*.

100.5 + 22.1 ⇒ 122.6

real1 { = | /= | < | > | <= | >= } real2: Boolean

Compares real *real1* and *real2*.

100.5 < 22.1 ⇒ FALSE

STR(real: Real): String

Converts real *real* into string.

STR(122.6) ⇒ "122.6"

Entity_: Any_

Entity_: Any_

Class *Entity_* is an abstract class. All object classes inherit from *Entity_* class.

Entity_ operations

obj1 ISA class1: Boolean

Tests if object *obj1* is of class *class1*.

"The black cat" ISA String ⇒ TRUE

DESTROY(obj)

Unlinks object *obj*. Destroyed object must not be used.

INT(obj1): Integer

Computes object hash code.

INT(THIS) ⇒ 137843391

obj1 $< coll1: Boolean

Tests if object *obj1* belongs to collection *coll1*.

"cat" $< ("The" $+ "black" $+ "cat") ⇒ TRUE

STR(obj1): String

Returns *obj1* display.

STR(objCat) ⇒ "The black cat"

obj1 DEFAULT obj2: Entity_

Returns *obj1* if not null else *obj2*.

"The cat" DEFAULT "black" ⇒ "The cat"

Entity_ functions

call(method : String, [arg1 : Entity_ [, arg2 : Entity_]]): Entity_

Calls method named *method* on current receiver with given arguments. Raises an error if the method does not exist or bad number of arguments. The method can contain wildcards (*) to call several methods.

class_(): Class

Returns class of receiver.

"The black cat ".class_ ⇒ CLASS(String)

copyVariableValuesTo(obj : Entity_ [, someNames : String]): String

Copies variable values from this to *obj*. If someNames is specified, copy only corresponding variable values.

getVariableNames(): String

Returns attribute names.

getVariableValue(aName : String): String

Returns values for variable named *aName*.

print(): String

Returns display of receiver.

objCat.print ⇒ "The black cat"

setVariableValue(aName : String, someValues : String): Entity_

Sets variable value for variable name *aName*.

String: Entity_

String: Entity_

String class inherits from *Entity_* class. Special characters are escaped by character ‟:

- n: Carriage return
- t: Tabulation
- ‟: Quote

"The black cat"

"Two"nlines"

String operations

string1 + string2: String

Concats strings *string1* and *string2*.

"The black" + " cat" ⇒ "The black cat"

string1 { = | /= | < | > | <= | >= } string2: Boolean

Compares strings *string1* and *string2*.

"The black cat" < "The black dog" ⇒ TRUE

BOOL(string: String): Boolean

Converts string *string* to Boolean.

BOOL("1") ⇒ TRUE

BOOL("cat") ⇒ TRUE

BOOL("false") ⇒ FALSE

INT(string: String): Integer

Converts string *string* to integer.

INT("123") ⇒ 123

INT("cat") ⇒ 0

REAL(string: String): Real

Converts string *string* to real.

REAL("123.5") ⇒ 123.5

RICHTEXT(string: String[, format: String]): RichText

Converts string *string* to RichText. *string* can be an RTF string or an XHTML string. Format can be forced: RTF, HTML, BOOST.

String functions

MAKEUNIQUENAME(aString: String, someStrings : String, options : String): String

Return a new string from *aString* not contained in *someStrings*. The *options* parameter can be set to *"CASE_INSENSITIVE"* to compare case insensitive strings.

MAKEUNIQUENAME("A1", "A1" $+ "A2" $+ "A3") ⇒ "A4"

STRF(format: String, ...): String

Returns formatted string replacing $n variable by n th argument.

STRF("The $1 cat", "black") ⇒ "The black cat"

STRSUB(string: String, beginIndex: Integer, endIndex: Integer): String

Returns substring of *string* beginning at *beginIndex* and ending at *endIndex*.

STRSUB("The black cat", 5, 9) ⇒ "black"

STRMATCH(string1: String, string2: String): Boolean

Tests if string *string1* matches *string2*. Use * for 0-n characters and # for one character. This function is not case-sensitive.

STRMATCH("The black cat", "*BLACK*") ⇒ TRUE

STRMATCHRE(string1: String, re: String): Boolean

Tests if string *string1* matches regular expression *re*.

STRMATCHRE("The black cat", "black") ⇒ TRUE

STRSEARCHRE(string: String, re: String): String

Searches regular expression *re* in string *string* and return matched groups.

STRSEARCHRE("The black cat", "(\S+a\S+)") ⇒ "black" $+ "cat"

STRTOTIME(string: String, format: String): Integer

Converts string *string* to date using format.

Letter	**Description**	**Example**
M	Month long name	January
m	Month short name	jan
p	Month short English name	jan
J	Day of weeklong name	Monday

Letter	Description	Example
j	Day of week short name (3 chars)	mon
k	Day of week short English name	mon
K	Day of week very short name (2 chars)	mo
Y	Year with 4 digits	1997
y	Year with 2 digits	97
n	Month with 1 or 2 digits	5
d	Month day with 1 or 2 digits	3
H	Hours (24) with 1 or 2 digits	4
h	Hours (12) with 1 or 2 digits	4
u	Minutes on 1 or 2 digits	5
s	Seconds on 1 or 2 digits	7
N	Month on 2 digits	01
D	Month day on 2 digits	03
I	Hours (24) on 2 digits	04
i	Hours (12) on 2 digits	04
U	Minutes on 2 digits	05
S	Seconds on 2 digits	07
A	AM/PM	AM
a	am/pm	am

STRTOTIME("5 January 2003", "$d $M $Y") ⇒ 1041764400

STRSUBSTITUTERE(string: String, re1: String, string2: String): String

Substitutes regular expression *re1* by string *string2* within string *string*. Usage of groups is valid and can be referenced by \n where n is the group index starting at 1.

STRSUBSTITUTERE("The black cat", "black", "white") ⇒ "The white cat"

STRSUBSTITUTERE("The black cat", "black (\w+)", "white \1") ⇒ "The white cat"

STRTOKENS(string: String, someSeparators: String): String

Splits string *string* with all separators *someSeparators*. *someSeparators* is a string collection.

STRTOKENS("The black cat", " " $+ "ac") ⇒ "The" $+ "bl" $+ "k" $+ "cat"

STRTOKENSRE(string: String, separator: String): String

Splits string *string* with separator *separator*. *separator* is a regular expression.

STRTOKENSRE("The black cat", "\s") ⇒ "The" $+ "black" $+ "cat"

STRCNT(string: String): Integer

Returns string length.

STRCNT("The black cat") ⇒ 13

STRINDEX(string: String, subString: String[, startIndex: Integer]): Integer

Returns index of sub string *subString* within string beginning at index *startIndex*. Default *startIndex* value is 1. If *startIndex* is negative, starts from (size + *startIndex)*. If *subString* is not found, return 0.

STRINDEX("The black cat", "black", 1) ⇒ 5

STRLOWER(string: String): String

Converts string *string* to lower case.

STRLOWER("The black cat") ⇒ "the black cat"

STRUPPER(string: String): String

Converts string *string* to upper case.

STRUPPER("The black cat") ⇒ "THE BLACK CAT"

STRUPPER1ST(string: String): String

Converts first character of string *string* to upper case.

STRUPPER("the black cat") ⇒ "The black cat"

TIMETOSTR(time: Integer, format: String[, options: String]): Integer

Converts string *format* to date. If option is equal to “GMT”, date is not converted to local.

TIMETOSTR(1041764400, "$d $M $Y") ⇒ "5 January 2003"

Letter	Description	Example
Y	4 digits year	1997
y	2 digits year	97
M	Month long name	January
m	Month short name	jan
p	Month tiny name	jan

N	2 digits month	01
n	1 or 2 digit month	5
D	2 digits month day	03
d	1 or 2 digit month day	3
J	Week day long name	Monday
j	Week day short name	mon
K	Week day tiny name	mo
I	2 digit hours (24)	04
i	2 digits hours (12)	04
H	1 or 2 digit hours (24)	4
h	1 or 2 digit hours (12)	4
U	2 digits minutes	05
u	1 or 2 digit minutes	5
S	2 digits seconds	07
s	1 or 2 digit seconds	7

Image: Entity_

Image attributes

dpi: Integer

Number of dots per inch.

format: String

Can be:

- bmp: Pixmap
- dib: Bitmap
- wmf: Metafile
- emf: Enhanced Metafile

height: Integer

Image height.

width: Integer

Image width.

Image functions

$RTF(img: Image): String

Returns RTF display for image *img*.

Image methods

differenceFrom(img: Image): Integer

Returns 4 integers (left, top, right, bottom) corresponding to first different rectangle.
Returns nothing if no difference found.

RichText: Entity_

RichText functions

$RICHTEXT(Entity_): RichText

Creates a rich text by concatenating elements of multiple types:

- String
- Image
- Rich text

RichText methods

asRTF([options: String]): String

Converts RichText to RTF String.

asText(): String

Converts RichText to plain text String. All rich information is removed.

asXHTML([options: String]): String

Converts RichText to XHTML String.

The *options* parameter can be:

- *DO_NOT_GENERATE_IMAGES*: does not generate images
- *OBJECT_FORMAT*: can be RTF

- *IMAGE_FORMAT*: image format can be gif, png, bmp
- *OBJECT_DIRECTORY*: when external object, defines the directory
- *IMGTAG*: replaces the *object* tag by *img* tag

containsOle(): Boolean

Is rich text contains OLE objects.

isRich(): Boolean

Is rich text contains rich information.

Collections

Collection operations

coll1 $= coll2: Boolean

Returns *true* if both collections contains same elements.

"cat" $+ "The" $+ "black" $= "The" $+ "black" $+ "cat" ⇒ TRUE

coll1 $+ coll2: Entity_

Returns a new collection containing *coll1* objects and *coll2* objects.

"The black" $+ " cat" ⇒ "The black" $+ " cat"

coll1 $- coll2: Entity_

Returns a new collection containing *coll1* objects without *coll2* objects.

("The black" $+ " cat") $- "The black" ⇒ " cat"

coll1 $* coll2: Entity_

Returns a new collection containing objects which are both in *coll1* and in *coll2*.

("The" $+ "black" $+ "cat") $* ("The" $+ "white" $+ "cat") ⇒ "The" $+ "cat"

coll1 $< coll2: Boolean

Tests whether *coll2* contains all elements contained in *coll1*.

"The" $< ("The" $+ "black" $+ "cat") ⇒ TRUE

Collection functions

$FIRST(coll): Entity_

Returns first element of collection *coll*.

$FIRST("The" $+ "black" $+ "cat") ⇒ "The"

$TAIL(coll): Entity_

Returns last elements of collection *coll*.

$TAIL("The" $+ "black" $+ "cat") ⇒ "black" $+ "cat"

$LAST(coll): Entity_

Returns last element of collection *coll*.

$LAST("The" $+ "black" $+ "cat") ⇒ "cat"

$AT(coll, index: Integer): Entity_

Returns element at index *index* of collection *coll*.

$AT("The" $+ "black" $+ "cat", 2) ⇒ "black"

$ATPUT(coll, index: Integer, elem): Entity_

Replaces the element at index *index* of collection *coll* by element *elem*

$AT("The" $+ "black" $+ "cat", 2, "white") ⇒ "The white cat"

$REV(coll): Entity_

Returns the inverse of collection *coll*.

$REV("The" $+ "black" $+ "cat") ⇒ "cat" $+ "black" $+ "The"

$INDEX(coll, ent1): Integer

Returns the index of first element *ent1* within collection *coll*.

$INDEX("The" $+ "black" $+ "cat", "cat") ⇒ 3

$CNT(coll): Integer

Returns the number of elements in collection *coll*.

$CNT("The" $+ "black" $+ "cat") ⇒ 3

$SOME(coll): Boolean

Returns TRUE if collection *coll* is not empty.

$SOME("The" $+ "black" $+ "cat") ⇒ TRUE

$SUB(coll, start: Integer, stop: Integer): Entity_

Returns a subcollection beginning at start position and ending at stop position.

$SUB("The" $+ "black" $+ "cat", 1, 2) ⇒ ("The" $+ "black")

$NO(coll): Boolean

Returns TRUE if collection *coll* is empty.

$NO("The" $+ "black" $+ "cat") ⇒ FALSE

$MAX(ints: Integer): Integer

Returns the max value among the integer *ints*.

$MAX(7 $+ 3 $+ 8) ⇒ 8

$MIN(ints: Integer): Integer

Returns the min value among the integer *ints*.

$MIN(7 $+ 3 $+ 8) ⇒ 3

$INTERVAL(ints: Integer): Integer

Returns a new collection containing *ints* and all integers included in *ints* interval.

$INTERVAL(3 $+ 8) ⇒ 3 $+ 4 $+ 5 $+ 6 $+ 7 $+ 8

$SET(strings : String): coll

Removes duplicated elements from collection *strings*.

$SET("The" $+ "black" $+ "black" $+ "cat") ⇒ "The" $+ "black" $+ "cat"

coll.SELECTUNIQUE(criterion): Entity_

Returns a subcollection of *coll* with duplication based on *criterion* removed.

("The" $+ "black" $+ "cat").SELECTUNIQUE(STR(STRCNT(EACH))) ⇒ "The" $+ "black"

coll.SORTBY(someCriteria): Entity_

Sorts collection *coll* using criteria *someCriteria*. If *someCriteria* is multiple, the first element is considered as the primary key, the second one as the secondary key …

("The" $+ "black" $+ "cat").SORTBY(EACH) ⇒ "black" $+ "cat" $+ "The"

$STRSEP(strs: String, sep: String): String

Joins strings of the list *strs* using the separator *sep*.

$STRSEP("The" $+ "black" $+ "cat", ", ") ⇒ "The, black, cat"

UI Functions

CHOICE(label: String, buttonLabels: String): Integer

Opens a dialog box with label *label* and buttons labelled *buttonLabels* and returns index of selected button. Return 0 if dialog is closed.

CHOICE("Is the cat black ?", "Yes" $+ "No")

=> 1

FILEEDIT(file: String): String

Opens file *file* with associated editor.

FILEEDIT("C:\document.doc")

INFO(msg: String): String

Opens message box with simple message and OK button.

INFO("Hello Black Cat")

INPUTLINE(initialAnswer: String, label: String): String

Opens a dialog box to get a string from user. Returns VOID when click *Cancel* or close button.

INPUTLINE("White", "What is the color of the cat ?")

=> "black"

PROGRESSBAR(action : String[, args : String]): String

Displays a progress bar. *action, args* parameters can be:

- OPEN, ”title”: Open progressbar window with specified title
- COUNT, ”count”: Define number of steps
- MESSAGE, ”message”: Change progressbar message
- CLOSE: Close progressbar
- STEP, ”step count”: Progress of progress bar of *step count* steps. If *step count* is absent, it is 1, if step count is not numeric, it is 1 and display as a message.

REQUESTFILER(directory: String, types: String): String

Opens a read file browser on directory *directory* to choose a file of one of type *types*. Argument *types* is a list of strings “<Type name> .<suffix>”. Returns a list of two strings: file name, type name. Return VOID when click *Cancel* or close button.

REQUESTFILER("C:\windows", "INI file .ini")

=> "C:\windows\system.ini" $+ "INI file"

REQUESTFILEW(directory: String, types: String): String

Opens a write file browser on directory *directory* to choose a file of one of type *types*. Argument *types* is a list of strings “<Type name> .<suffix>”. Returns a list of two strings: file name, type name. Return VOID when click *Cancel* or close button.

REQUESTFILER("C:\windows", "INI file .ini")

=> “C:\windows\dummy.ini” $+ “INI file”

REQUESTDIR(directory: String): String

Opens a directory browser initialized on directory *directory*. Returns VOID when click *Cancel* or close button.

REQUESTDIR(".")

⇒ “C:\Documents and Settings\user\Mes Documents”

ASK <variable> [<-option>] [: availableValues] [LABEL “<label>“]

Allows the quick creation of dialog boxes.

<-options>:

- *multiline*: Creates a multiline text field widget.
- *multiple*: Creates a multiple selection text field widget.

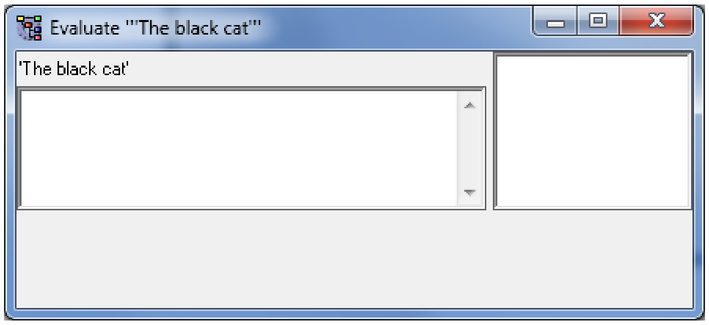
- *readonly*: Creates a read-only text field widget.
- *focus*: Gives the focus to the widget data capture.
- *list*: Creates a list widget.
- *combo*: Creates a combo-box widget.
- *duallist*: Creates a dual-list widget.
- *completion*:Creates a single line with completion.
- *radio*: Creates a radio button for Boolean.
- *radio multiple*: Creates check boxes.

INSPECT(obj: Entity_{, options : String})

Opens evaluator on an object *obj*. Useful for debugging. Options can be:

- *UNIQUE* to not open a new evaluator and reuse opened one.

```
INSPECT("The black cat")
```



WINGETIMAGE(windowHandle: Integer{, options : String}): Image

Returns window contents as an image.

options:

- *LEFT:* crops left
- *TOP:* crops top
- *RIGHT:* crops right
- *BOTTOM:* crops bottom

WINRAISE(titles: String, classNames: String{, options: String}): Integer

Finds a window by its title and class. *Titles* and *classNames* can be multiple to find a sub-window. Options can be:

- *NORAISE* in order not to raise window and just returns found window handle
- *ROOT:* to define root window.

Returns window handle.

WINSEND(windowHandle: Integer, events: String): Integer

Sends mouse and keyboard events like Windows does. If windowHandle is specified (not 0), screen coordinates are transformed. Parameter events can be:

- *MOUSE_MOVE:* move mouse pointer to specified coordinates contained in argument. *windowHandle* parameter is used to transform coordinates.
- *KEYBOARD_STRING:* enter all characters contained in argument.
- *KEYBOARD_KEY:* enter a key. Example: Ctrl+O.
- *MOUSE_SELECTBUTTONDOWN* left button down.
- *MOUSE_SELECTBUTTONUP* left button up.
- *MOUSE_SELECTBUTTONCLICK* left button down and up.
- *MOUSE_MENUBUTTONDOWN* right button down.
- *MOUSE_MENUBUTTONUP* right button up.
- *MOUSE_MENUBUTTONCLICK* right button down and up.
- *SLEEP:* wait for a delay in milliseconds

File Functions

FILECONTENTS(file: String [, options : String]): String

Returns contents of file *file*. Returns VOID on error.

Parameter *options* can be following:

- *BINARY* to get contents as binary contents. Does not transform anything.

FILECONTENTS("C:\doc1.doc") ⇒ "…"

FILECOPY(file1: String, file2: String, options: String): Boolean

Copies file *file1* into file *file2*. Works with directories since v3.1. Returns *FALSE* on error. If *options* contains *"SYSTEM"*, system copy is called.

FILECOPY("C:\doc1.doc", "C:\doc2.doc") ⇒ TRUE

FILECRC(file: String[, options : String]): Integer

Computes CRC on file *file*.

Parameter options can be following:

- *UNICODE* to indicate file is a unicode file
- *EXCLUDE_BEGIN:* defines a string as a starting of exclusion
- *EXCLUDE_END:* defines a string as an ending of exclusion

FILEDATE(file: String): Integer

Returns modification file date. Returns VOID on error.

FILEDATE("C:\document.doc") ⇒ 1059402702

FILEDELETE(file: String [, force : Boolean]): Boolean

Deletes file *file*. Returns FALSE on error. If *force* parameter is set to TRUE, delete operation occurs even if file read-only flag is set.

FILEDELETE("C:\doc1.doc") ⇒ TRUE

FILEDIRECTORY(file: String[, level : Integer]): String

Returns directory of file *file* with character / at end. Parameter *level* indicated the directory level to go. By default, *level* value is 1.

FILEDIRECTORY("C:\doc1.doc") ⇒ "C:\"

FILEEDITOR(file: String): String

Returns associated editor path. Returns VOID when not found.

FILEEDITOR("C:\document.doc") ⇒ "C:\Program Files\Microsoft Office\Office10\WINWORD.EXE"

FILEEXISTS(file: String): Boolean

Tests if file exists.

FILEEXISTS("C:\doc1.doc") ⇒ FALSE

FILEFORMATNAME(file: String, format: String[, dir: String]): String

Re-format a file name.

Letter	**Description**	**Example (on "C:\Program Files\docs\doc1.doc")**
a	Absolute filename	C:\Program Files\docs\doc1.doc
b	Name without directory neither suffix	doc1
c	Directory of directory	C:\Program Files\
d	Directory	C:\Program Files\docs\
e	Extension	doc
g	Absolute directory without final '/'	C:\Program Files\docs
h	Directory base name	docs
j	Remove last '/'	C:\Program Files\docs\doc1.doc
p	DOS conversion	C:\Progra~1\docs\doc1.doc
q	DOS conversion when not file name not compatible with locale	C:\ Program Files\docs\doc1.doc
r	Relative filename	docs\doc1.doc

Letter	Description	Example (on "C:\Program Files\docs\doc1.doc")
s	Suffix	.doc
u	UNC name	\\computer\C\Program Files\docs\doc1.doc
/	Separator	

FILEFORMATNAME("C:\doc1.doc", "bs") ⇒ "doc1.doc"

FILEISDIRECTORY(file: String): Boolean

Tests if file *file* is a directory.

FILEISDIRECTORY("C:\doc1.doc") ⇒ FALSE

FILELOADIMAGE(file: String): Image

Loads image (Metafile/Bitmap) from file *file*. Returns VOID on error.

FILELOADIMAGE("C:\test.bmp") ⇒ <Image>

FILEMOVE(file1: String, file2: String): Boolean

Moves file *file1* to file *file2*. Returns FALSE on error.

FILEMOVE("C:\doc1.doc", "C:\doc2.doc") ⇒ TRUE

FILEREADONLY(file: String [, readonly : Boolean]): Boolean

Gets or sets read-only flag.

FILEREADONLY("C:\doc1.doc") ⇒ TRUE/FALSE

FILEREADONLY("C:\doc1.doc", TRUE) ⇒ TRUE

FILESAVEIMAGE(file: String, image: Image): Boolean

Saves image (Metafile/Bitmap) in file *file*. Returns TRUE if OK.

FILESAVEIMAGE("C:\test.bmp", img) ⇒ TRUE

FILETEMP(): String

Returns a new temporary filename.

FILETEMP()⇒ "C:\DOCUME~1\user\LOCALS~1\Temp\reqtify1"

FILEZIP(zipFile: String, command: String, files: String[, zippedFiles]): String

Processes zip files named *zipFile*. *files* is an optional argument.

Cmd	Description	Example
get	Extract a file from zipFile	FILEZIP("c:\x.zip", "get", "C:\temp\x.doc", "x.doc")

Cmd	Description	Example
add	Add a new file to zipFile	FILEZIP("c:\x.zip", "add", "x.doc" $+ "y.doc")
list	List all files stored in zipFile	FILEZIP("c:\x.zip", "list") ⇒ "x.doc" $+ "y.doc"

SEARCHLINEINFILE(file: String, string: String[, startLine: Integer]): Integer

Searches first occurrence of string *string* in file *file* from line number *startLine* and returns its line number. Returns 0 on error.

SEARCHLINEINFILE("C:\doc1.txt", "test", 3) ⇒ 12

DIRSEP(): String

Returns file separator.

DIRSEP ⇒ "\"

MKDIR(dir: String): Boolean

Creates a new directory. Returns FALSE on error.

MKDIR("C:\dir") ⇒ TRUE

DIRLIST(dir: String[, options: String]): String

Returns file list contained in directory *dir*.

Parameter *options* can be following:

- *RECURSIVE* to go down into subdirectories
- *ABSOLUTE* to get absolute file names instead of relative ones.
- *ONLY_DIRECTORIES* to only get directories
- *MATCH:* <matching> to only includes specific file names
- *DONT_MATCH:* <matching> to excludes specific file names
- *MATCHRE:* <matching> to only includes file names matching specified regular expressions
- *DONT_MATCHRE:* <matching> to excludes file names matching specified regular expressions

DIRLIST("C:\") ⇒ "AUTOEXEC.BAT" $+ "CONFIG.SYS" $+ …

TOOLDIR(): String

Returns directory of tool.

TOOLDIR() ⇒ "C:\Program Files (x86)\Reqtify v2017\"

EXECSUBDIR(): String

Returns subdirectory containing program.

EXECSUBDIR()⇒ "bin.w32"

Stream Functions

CLOSE()

Closes last opened buffer.

CLOSE()

FOPEN(file: String, options : String)

Opens file *file* for read/write operations, raising exception on error. If Parameter *options* can be:

- *W* to open for writing (default). File is created if it does not exist
- *R* to open for reading
- *C* to open for writing if modified
- *A* to open for appending
- *L* to open in exclusive rights. The Lock is deleted when closing.
- *ENCODING:* to set encoding to "UTF8", "UTF16", "BASE64", "HEXA", "COMPRESS", "GZIP"
- *EOL:* to set line end convention. 1: CRLF, 2: LF, 3: CR, 4: transparent.

Calls *CLOSE* function to close file.

FOPEN("C:\document.doc", "W" $+ "ENCODING:" $+ "UTF8" $+ "TR::" $+ ""n" $+ "\n")

GET(ind : Integer, sep: String)

Gets all characters from stream index *ind* (0: current stream, 1: previous one) until separator *sep*.

OPENBUFF(options : String)

Opens a memory buffer. Argument *options* can be following:

- *ENCODING:* to set encoding to "UTF8", "UTF16", "BASE64", "HEXA", "COMPRESS", "GZIP"
- string for reading instead of writing

Calls *CLOSE* function to return buffer contents.

OPENBUFF()

PUT(obj: Entity_)

Writes *obj* on last opened buffer.

PUT("The black cat")

PUTF(format: String, ...)

Writes a text with a format string.

```
PUTF("The $1 cat"|"Le chat $1", "black"|"noir")
```

PUTFILECONTENTS(file: String)

Writes file contents on last opened buffer.

```
PUTFILECONTENTS("c:\doc1.txt")
```

PUTN(obj: Entity_)

Writes *obj* on last opened buffer with a CR at the end.

```
PUTN("The black cat")
```

System Functions

PUTHTTPCONTENTS(url : String{, options : String}): String

Gets HTTP page from *url*. *url* can be a list of URL components to be concatenated. Parameter options can be:

- *BODY:* defines request body.
- *EOL: when 1,* transforms body end of lines into CRLF.
- *HEADER:* adds a header to request.
- *PASSWORD:* defines password for authentication
- *POST* uses POST verb instead of GET.
- *USER:* defines user for authentication

Returns error code or error message.

PUTSYS(cmd: String{, options : String}): Integer

Executes command *cmd*, prints output on current stream and returns error code.

Parameter *options* can be:

- NOWAIT: does not wait for end of command execution
- CODE_PAGE: cp: code page of output
- HIDE : hides execution
- REMOVE_FINAL_CR: removes end character, in general a CR
- DISPATCH_EVENTS: windows events are processed during command execution
- DIRECTORY: defines current directory for called process

```
OPENBUFF;
PUTSYS("ls", "HIDE"$+"REMOVE_FINAL_CR");
```

CLOSE

⇒ "file1   file2   ..."

SYSINFO(key: String): String

Gets system information. Parameter *key* can be one of the following:

- HOST: host name
- LOGIN: User login
- ARGUMENTS: Reqtify command line arguments

SYSINFO("LOGIN") ⇒ "user"

SYSW(cmd: String): String

Executes command *cmd*, waits for result and return command output.

SYSW("ls") ⇒ "file1   file2   ..."

TIMENOW(): Integer

Returns the current date and time. The returned number is the number of seconds since 1 January 1970.

TIMENOW()⇒ 1069628365

WIN(): Boolean

Tests if current platform is Windows.

WIN()⇒ TRUE

UNIX(): Boolean

Tests if current platform is Unix.

UNIX()⇒ FALSE

GETENV(var: String): String

Returns environment variable value. Returns VOID when absent.

GETENV("PATH") ⇒ "C:\Windows\System32;..."

SETENV(var: String, val: String) : String

Sets environment variable value with *val*. If string *val* is null, removes variable *var*. Returns *val*.

SETENV("CAT", "The black cat")

GETCONFIG(section: String, key: String): String

Returns value of configuration file name (.INI) or registry.

GETCONFIG("Editor", "Font") ⇒ "Arial"

GETCONFIG("HKEY_CLASSES_ROOT", "Font") ⇒ "Arial"

GETCONFIG("HKEY_CLASSES_ROOT", "Word.Application\CLSID\") ⇒ "{000209FF-0000-0000-C000-000000000046}"

SETCONFIG(section: String, key: String, value: String)

Sets value in configuration file name. *section* can be a full file name.

SOCKETREQUEST(host: String, port: Integer, data: String): String

Opens a socket on computer *host* port *port* and sends string *data* and waits for result.

SOCKETREQUEST("server", 1234, "...") ⇒ "..."

APPLICATIONDATADIR(): String

Returns application dir.

APPLICATIONDATADIR() ⇒ "C:\Users\<user>\AppData\Roaming\DassaultSystemes\reqtify\"

APPLICATIONNAME(): String

Returns application name.

APPLICATIONNAME()⇒ "Reqtify"

URLENCODE(var: String): String

Encodes a string to create a valid URL.

URLENCODE("My tailor is rich.") ⇒ "My%20Tailor%20is%20rich."

URLDECODE(var: String): String

Returns a string rebuilt without URL encoded code.

URLDECODE("http://mysite//My%20Tailor%20is%20rich") ⇒ "http://mysite//My tailor is rich"

LASTERRORSTRING: String

Returns the last error message when an OTScript exception is raised.

{ RAISE("Invalid character") } CATCH { INFO("Exception: " + LASTERRORSTRING) };

Parsing Functions

LEX(file: String, regexps: String, obj1: Entity_, methodNames: String, state: String): String

Analyzes a file and call methods *methodNames* associated with each regular expression *regexps* passing state.

XMLEXTRACT(tree: XmlNode, subTree : XmlNode, callback : Callback): Integer

Scans *tree* to find a *subtree* and calls *callback* for each found occurrences. Parameters of callback are: identifier, label, text, GUID, parent and image. Returns the number of found occurrences.

CLASS Test : Entity_;

```
METHOD Test.clbk(index : Integer, depth : Integer, i : String, l : String) : {
        INFO(i);
};

{
        TMP tst := NEW(Test);
        XMLEXTRACT(
                XMLPARSE("<Root><Tag id='1'/><Tag id='2'/></Root>"),
                XMLPARSE("<Tag id='$i'/>"),
                tst.callback("clbk")
        );
}
⇒ 2
```

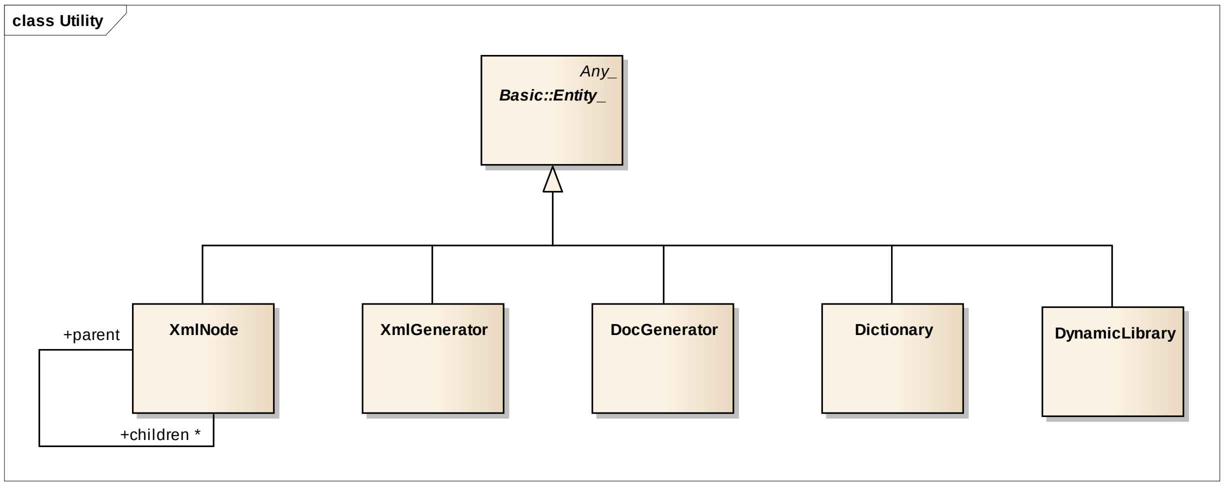
XMLPARSE(fileOrString: String{, options: String}): XmlNode

Analyzes an XML file or an XML string and returns a tree of XmlNode. Parameter *options* can be one or more of following:

- ENCODING: to define encoding. It can be one of ASCII, UTF-8, UTF16.
- MAX_NODES: to maximize number of parsed nodes.

Utility Classes

Schema



Dictionary: Entity_

This functional class named *Dictionary* is a hashed collection with external key. It allows to access quickly a value associated with a specific key.

Dictionary methods

addKeysAndValues(keysAndValues : Entity_) : Entity_

Considers *keysAndValues* as a list of couples key/value and fills the dictionary with these data.

at(key : Entity_) : Entity_

Returns value associated to a specific key.

atAdd(key : Entity_, value : Entity_) : Entity_

Adds a value for a specific key. If key is already present, value is added, if not, it is created.

atPut(key : Entity_, value : Entity_) : Entity_

Associates a value to a key in a dictionary. Returns the value.

keys() : Entity_

Returns all keys.

removeKey(key : Entity_) : Entity_

Removes values at key *key*.

stringAt(key : Entity_) : String

Returns value associated to a key if this value is a string.

stringKeys() : String

Returns all dictionary keys those are typed string.

values() : Entity_

Returns all values.

Dictionary example

```
TMP dict := NEW(Dictionary);

dict.atPut("cat", "black");

dict.atAdd("cat", "white");

dict.at("cat"); // "black" $+ "white"

dict.keys;       // "cat"
```

DocGenerator : Entity_

Functional class to produce documents.

DocGenerator methods

open(aFileName: String, aTitle: String, aTemplateFileName: String{, fields: String}): DocGenerator

Opens a new document named *aFileName*. *aTitle* will appear on first page. *aTemplateFileName* is used to define document styles. It can be an absolute file name or just a name. In this second case, template file name will be searched firstly in *<project directory>/doc_templates* and then in *config/doc_templates* directory.

Format	**Extension**	**Tool**
RTF	Rtf	Word
Text	Txt	Text editor

Format	Extension	Tool
HTML	Html	Navigator
MIF	mif	FrameMaker
PDF	pdf	Acrobat PDF
PostScript	ps	
DocBook	xml	

fields is a collection of field name/field value pairs.

close(): DocGenerator

Closes document.

beginTag(aDocBookTag: String, someAttributes: String): DocGenerator

Opens a new document section.

Tag name	Parameters	Description	Parents
article		Document	
colspec	colwidth	Table group column specification	tgroup
entry		Row entry	row
itemizedlist		Element list	article, section
link	linkend	Cross-reference	literal
listitem		Element of list	itemizedlist
para		Simple paragraph	article, section
row		Table row	thead, tbody
section		Section	article, section
table		Table	article, section
tgroup	cols	Table elements group	table
thead		Table header	tgroup
tbody		Table body	tgroup
ulink	url	URL	literal

endTag(): DocGenerator

Closes last opened tag.

text(aString: String): DocGenerator

Appends text to current document.

DocGenerator example

```
TMP dg := NEW(DocGenerator);
dg.rootClass := CLASS(Project);
dg.open("c:\out.rtf", "Doc title", "portrait");
        dg.beginTag("section", VOID(String));
                dg.beginTag("title", VOID(String));
                        dg.text("Section title");
                dg.endTag();
                dg.beginTag("para", VOID(String));
                        dg.text("Text");
                dg.endTag();
        dg.endTag();
        dg.beginTag("table", VOID(String));
                dg.beginTag("tgroup", "cols"$+"2");
                        dg.beginTag("colspec", "colwidth"$+"100pt"); dg.endTag();
                                dg.beginTag("colspec", "colwidth"$+"200pt");
                                dg.endTag();
                                dg.beginTag("thead", VOID(String));
                                        dg.beginTag("row", VOID(String));
                                                dg.beginTag("entry", VOID(String));
                                                        dg.text("Col1");
                                                dg.endTag();
                                                dg.beginTag("entry", VOID(String));
                                                        dg.text("Col2");
                                                dg.endTag();
                                        dg.endTag();
                                dg.endTag();
                                dg.beginTag("tbody", VOID(String));
                        $INTERVAL(1$+4).{
                            dg.beginTag("row", VOID(String));
                                dg.beginTag("entry", VOID(String));
                                    dg.text("Row"+STR(EACH)+",Col1");
                                dg.endTag();
                                dg.beginTag("entry", VOID(String));
                                    dg.text("Row"+STR(EACH)+",Col2");
                                dg.endTag();
                            dg.endTag();
                        };
                    dg.endTag();
                dg.endTag();
            dg.endTag();
        dg.endTag();
dg.close();
```

ReportModel : Entity_;

Represents a Report Model.

ReportModel attributes

name: String

Name of Report Model.

description: String

Description of Report Model.

ReportModel methods

generate(fileName : String, template : String, receiver : Project[, arg1 : Entity_[, arg2 : Entity]])

Generates a report. Parameter *template* can be null if the template is defined in the report.

ReportModel example

```
{
  TMP rmls := kernel.reportModelLists;
  TMP rm := rmls.reportModels[name = "Traceability Matrix"];
  rm.generate("report.rtf", "portrait", project, project.documents);
}
```

ReportModelList : Entity_;

Represent a Report Model.

ReportModelList attributes

filename: String

File name of Report Model List.

ReportModelList methods

reportModels(): ReportModel

Report models belonging to Report Model List.

XmlNode: Entity_

Represents an XML tree. This class partially supports DOM protocol.

XmlNode attributes

id: String

Attribute named *id* attribute of the node.

label: String

Attribute named *label* of the node.

childNodes: Entity_

Child nodes of the node. Can be strings or *XmlNode*.

entity: Entity_

User data. Usually set to corresponding object in data model.

image: Image

Image of the node.

parent: XmlNode

Parent of the node.

tagName: String

Tag of the node.

XmlNode methods

appendChild(aNode: Entity_): XmlNode

Appends a child to receiver children.

elementNodes([tagNames : String]): XmlNode

Returns child nodes which are *XmlNode* and optionally filters on tag names.

getAttribute(attributeName: String): String

Returns value of attribute named *attributeName*.

getBooleanAttribute(attributeName: String): Boolean

Returns Boolean value of attribute named *attributeName*. Returns TRUE if value is "true" and "false" in other cases.

getElementById(id: String): XmlNode

Returns node which id is *id*. Goes down recursively in XML tree.

getElementsByName(aName: String): XmlNode

Returns nodes which tag name is *aName*. Goes down recursively in XML tree.

getField(pseudoAttributeName: String): Entity_

Returns value of pseudo attribute named *pseudoAttributeName*.

getXmlField(pseudoAttributeName: String): Entity_

Returns XML value of pseudo attribute named *pseudoAttributeName*.

newChild(tagName: String [, args : String]): XmlNode

Creates a child of receiver with tag named *tagName*. Optional parameter *args* is a list of attribute names and values.

newChildren(tagName : String, elements: Element): XmlNode

Creates new children with tag name *tagName* for all *elements*. Instance variable entity is set with element.

removeChild(aNode: Entity_): XmlNode

Removes a child of receive.

setAttribute(attributeName: String, value: String): XmlNode

Sets or creates new attribute with value *value*.

setBooleanAttribute(attributeName: String, value: Boolean): setBooleanAttribute

Sets or creates new attribute with Boolean value *value*.

textContent(): String

Returns all contained text as one string.

XmlGenerator: Entity_

Functional class to produce XML file.

XmlGenerator attributes

formatOption: String

Changes output format:

- *withNLs*: without indentation
- *none*: all in one line
- *withIndentation*: with indentation (default mode)

XmlGenerator methods

open(filename: String[, dtdRoot: String, dtdUrl: String]): XmlGenerator

Opens an XML file for writing.

close(): XmlGenerator

Closes an XML file.

beginTag(aTagName: String, someAttributeValues: String): XmlGenerator

Opens a new tag named *aTagName* with attributes *someAttributeValues*.

emptyTag(aTagName: String, someAttributeValues: String): XmlGenerator

Generates an empty tag.

endTag(): XmlGenerator

Closes last opened tag.

text(text: String): XmlGenerator

Writes text in an XML file.

tree(node : XmlNode[, options : String]): XmlGenerator

Writes a tree of XML element nodes.

For generation without indentation:

```
tree(node, "FORMAT:" $+ "withNLs")
```

For generation on one line:

```
tree(node, "FORMAT:" $+ "none")
```

For generation on one line for specific tag:

```
tree(node, "TAG_FORMATTED_AS_NONE:" $+ "tag")
```

DynamicLibrary: Entity_

This class allows calling functions located in dynamic libraries. After defining a class inheriting from *DynamicLibrary* class, all methods declared on this new class are mapped to functions defined in the dynamic library.

Only following function signatures are supported:

- Function() : Boolean/Integer/String
- Function(Boolean/Integer/String) : Boolean/Integer/String
- Function(Boolean/Integer/String, Boolean/Integer/String) : Boolean/Integer/String
- Function(String, String, String) : Boolean/Integer/String

Example

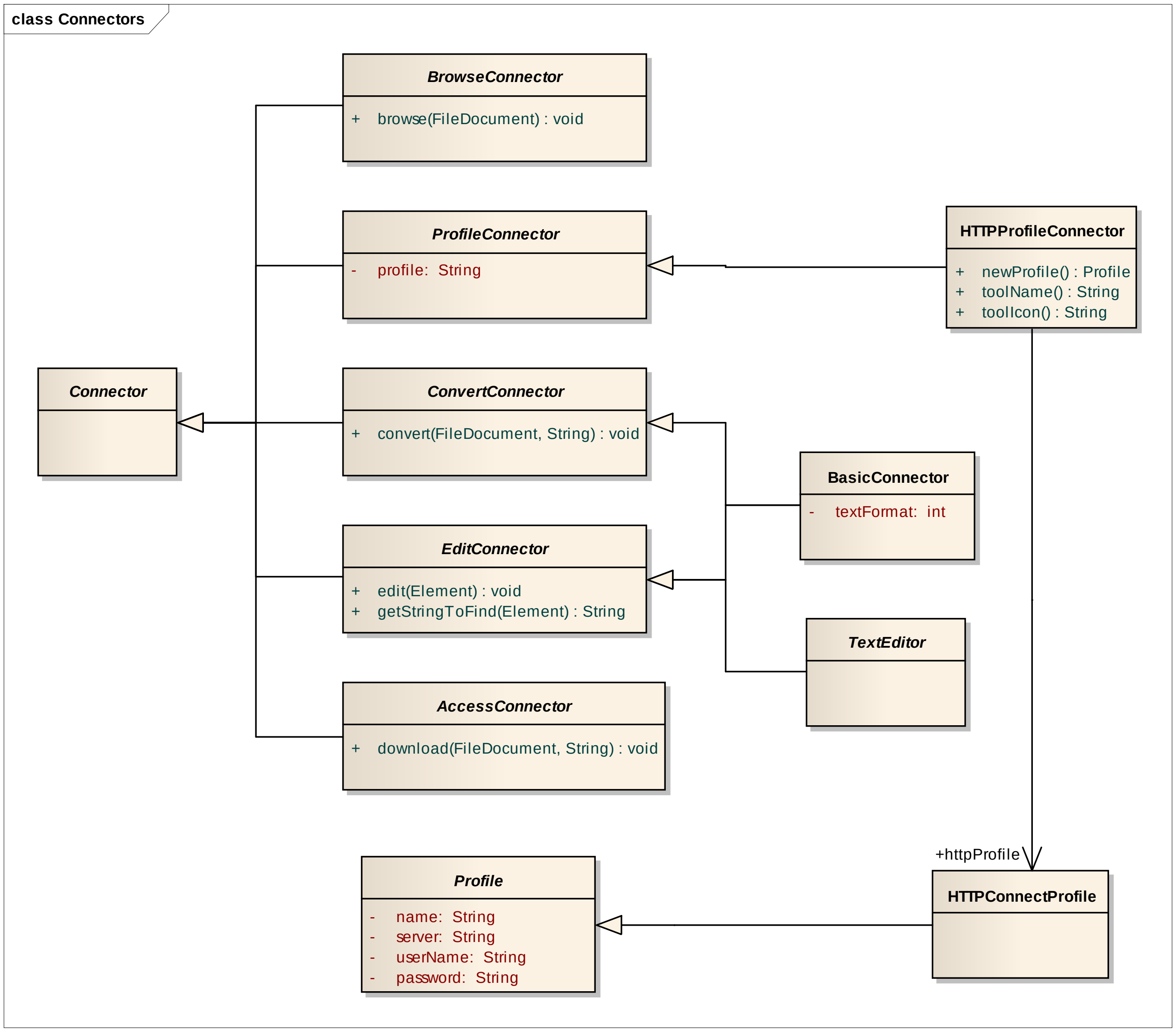
This example uses standard *kernel32.dll* dynamic library to call *Beep()* function.

```
CLASS Kernel32 : DynamicLibrary;
METHOD Kernel32.Beep(freq : Integer, duration : Integer) : VOID(Boolean);

{
	TMP lib := GETOBJECT(Kernel32, "kernel32.dll");
	lib.Beep(500, 200);
}
```

Connectors

This part explains how to create a new connector. A connector has four functions:

- Converting data to textual or XML file
- Navigating to original tool
- Downloading data from a remote place
- Browsing input data to select a document.



Data Model

Convert Connector

Convert connector is used to convert a source document into a textual or XML format.

To create a new convert connector:

- Create a class inheriting from *Connector* class.
- Add a method named *convert(FileDocument, String[, Kernel])* on this new class.
- Add some attributes if parameters are required. If an attribute has a label, this attribute is considered as public and is displayed in *Project Editor*. These attributes are initialized from *FileDocument* variable values. For a parameter

named *param*, if a method named *paramList()* exists, returned values are displayed as possible choices for *param*. If a method named *paramChange()* exists, this method is called after the parameter modification.

- A specific *textFormat* integer attribute shall be defined to indicate rich text format. One or more constants can be composed for this attribute:
  - 0: none
  - 1: RTF
  - 2: HTML
  - 3: Internal
  - 4: Boost
  - 5: RTF or HTML
  - 8: gz.b64
  - 16: ident, label and attribute can contain rich text.
- Optionally, a *getDate(FileDocument)* method can be added to compute a date when default date function computation is not sufficient. This method must return an integer.
- Optionally, a *convertAll(Project, FileDocument[, Kernel])* method can be added to prepare the conversion for several documents. Second optional parameter contains all same type documents those need to be converted.
- Optionally, a *preprocessXXX(FileDocument, String)* method can be added to the preprocess input file for connectors based on files.
- Optionally, a *postanalyze[XXX](FileDocument[, Dictionary])* method can be added to finalize object tree. This method is called just after conversion.
  XXX is defined in .types file with *ToolPostprocess=XXX.*
  In this method, links are not finalized but second optional argument can be used to access links. Keys are link objects and values are target identifiers. Links can be modified by updating this dictionary or returning a new dictionary containing new links. When a method named *postanalyze()* is defined on a connector, it is always called.
- Optionally, a *postprocess[XXX](FileDocument[, Document])* method can be added to finalize object tree and links. This method is called after link edition. Second parameter is optional and contains other reloaded documents. This postprocess method can also be written as a simple method on *FileDocument* class. Last optional argument contains all project modified documents.
  XXX is defined in .types file with *ToolPostprocess=XXX.* When a method named *postprocess()* is defined on a connector, it is always called.
- Optionally, an *initialize(FileDocument)* method can be added to initialize the connector.
- Optionally, an *afterLoad(Project)* method can be added to realize some actions after project loaded.

- Optionally, an *updateEnumerationValues(ElementType)* method can be added to update enumeration values of an element type.

Example:

```
CLASS NewConnector : ConvertConnector;

ATTRIBUTE NewConnector.parameter : String LABEL “Parameter”|”Paramètre”;

METHOD NewConnector.parameterList(doc : FileDocument) : {
        <OTScript code>      // Return a list of values
};

METHOD NewConnector.parameterChange(doc : FileDocument) : {
        <OTScript code>
};

METHOD NewConnector.convert(doc : FileDocument, outfile : String) : {
        <OTScript code>
};

METHOD NewConnector.convertAll(project : Project, docs : FileDocument) : {
        <OTScript code>
};

METHOD NewConnector.postanalyzeXXX(doc : FileDocument) : {
        <OTScript code>
};

METHOD NewConnector.postprocessXXX(doc : FileDocument, docs : FileDocument) : {
        <OTScript code>
};

METHOD FileDocument.postprocessXXX() : {
        <OTScript code>
};
```

Access Connector

Access connector is used to get file coming from a remote location. In Reqtify project, this connector is always associated to a type based on a convert connector.

To create a new access connector:

- Create a class inheriting from Access*Connector* class.
- Add some attributes if parameters are required. If an attribute has a label, this attribute is considered as public and is displayed in *Project Editor*. These attributes are initialized from *FileDocument* variable values.
- Add a method named *download(FileDocument, String, aFile : String)* on this new class.

Example:

```
CLASS NewConnector : AccessConnector;

ATTRIBUTE NewConnector.parameter : String LABEL ”Parameter”|”Paramètre”;


METHOD NewConnector.download(doc : FileDocument, outfile : String) : {

        <OTScript code>

};
```

Upload Connector

Upload connector is used to upload to a remote location.

To create a new upload connector:

- Create a class inheriting from Upload*Connector* class.
- Add a method named *browseSave(Object, name : String, someTemplates : String)* to this new class.
- Add a method named *upload(Object, localFile : String)* to this new class.

Example:

```
CLASS NewConnector : UploadConnector;

METHOD NewConnector.browseSave(obj : Object, aName : String, someTemplates : String) : {

        <OTScript code>

};

METHOD NewConnector.upload(obj : Object, localFile : String) : {

        <OTScript code>

};
```

Edit Connector

Edit connector is used to navigate from Reqtify to an element in external tool. In Reqtify project, this connector is always associated to a type based on an edit connector.

To create a new edit connector:

- Create a class inheriting from *EditConnector* class.
- Add a method named *edit(Element)* on this new class.
- Optionally, add a method named *editdoc(FileDocument)* to open document or modification document.

Example:

```
CLASS NewConnector : EditConnector;


METHOD NewConnector.edit(anElement : Element) : {
        <OTScript code>
};
```

Browse Connector

Browse connector is used to select a document.

To create a new browse connector:

- Create a class inheriting from *BrowseConnector* class.
- Add a method named *browse(FileDocument)* on this new class.

Example:

```
CLASS NewConnector : BrowseConnector;


METHOD NewConnector.browse(aDoc : FileDocument) : {
        <OTScript code>
};
```

Project Connector

Project connector is used to manage a Reqtify project files in a remote repository.

To create a new project connector:

- Create a class inheriting from *ProjectConnector* class.

- Add some attributes if parameters are required. If an attribute has a label, this attribute is considered as public and is displayed in *Project Editor*.
- Add a method named *afterSave (Project)* on this new class.
- Optionally, add a method named *beforeOpenFile(Project, Document, String)* on this new class. This method is called before opening project file. The content of this method can be a lock or a check-out for example.
- Optionally add a method named *afterCloseFile(Project, Document, String)* on this new class. This method is called after the file is closed. The content of this method can be a check-in or an unlock.

Example:

```
CLASS NewConnector : ProjectConnector;


METHOD NewConnector.afterSave(aProject : Project) : {

        <OTScript code>

};


METHOD NewConnector.beforeOpenFile(aProject : Project, aDocument : FileDocument, aFile :
String) : {

        <OTScript code>

};


METHOD NewConnector.afterCloseFile(aProject : Project, aDocument : FileDocument, aFile :
String) : {

        <OTScript code>

};
```

Snapshot Connector

The snapshot connector is used to manage snapshots in a remote repository.

To create a new snapshot connector:

- Create a class inheriting from the *SnapshotConnector* class.
- Add a method named *getSnapshots(Project)* on this new class. This method shall return a list of *Snapshot* objects.
- Add a method named *loadSnapshot(Project, Snapshot)* on this new class. This method shall return a project from a snapshot.

- Add a method named *saveSnapshot(Project, Snapshot)* on this new class. This method shall save a snapshot from a project.
- Optionally, add a method named *deleteSnapshot(Project, String)* on this new class. This method shall delete a snapshot referenced by its name.

Example:

```
CLASS NewConnector : ProjectConnector;


METHOD NewConnector.getSnapshots(aProject : Project) : {

        <OTScript code>

};


METHOD NewConnector.loadSnapshot(aProject : Project, aSnapshot : Snapshot) : {

        <OTScript code>

};


METHOD NewConnector.saveSnapshot(aProject : Project, aSnapshot : Snapshot) : {

        <OTScript code>


};

METHOD NewConnector.deleteSnapshot(aProject : Project, someSnapshotNames : String) : {

        <OTScript code>

};
```

Example

The goal of this example is to create a convert connector to …

Here are the steps:

- Create a file named *MyC.br* in *config/otscript* directory.
- Add following OTScript code to create the connector:

```
CLASS MyC : ConvertConnector;
```

- Define convert method:

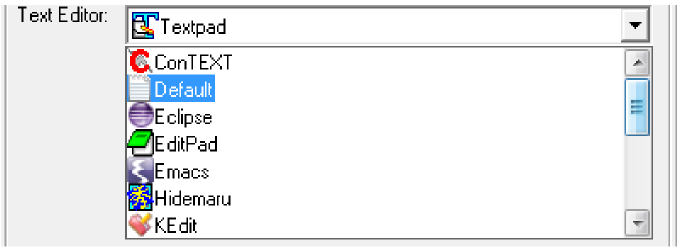
```
METHOD MyC.convert(aDocument : FileDocument, outFile : String) : {

        …
```

}

- Run Reqtify to take this new connector into account
- Create a new type based on this new connector. *MyC* appears in in converter drop listbox.

Text Editor

It is possible to add a new text editor in *Text Editor* drop list box.



To realize this operation, create a new class named *Xyz* with a method named *edit* and define an icon option.

```
CLASS Xyz -icon ["icon"] : TextEditor;

METHOD Xyz.edit(anElement : Element): {

        exploreIfDirectory(anElement) IF FALSE: { // For directories

…

        };

};
```

Command Line Interface

This chapter provides an overview of the usage of Reqtify which can be executed at command line.

To write command lines, it is better to have a few knowledge of OTScript language.

A command execution can be done as a command line execution described in the Command Line section.

A brief description of a frequently used example is given to help the user in the next chapter. It is about the Report Generation.

Command Line

To execute Reqtify in command line, one of the following commands lines have to be entered in a command window:

reqtify [-l {eng|fra}] **-batch** [-logfile file.log] **-exec** "<OTScript block>"

or

reqtify [-l {eng|fra}] **-batch** [-logfile file.log] **-execfile** scriptfile.br

-l {eng|fra|chn|jpn}

Choose the MMI language launching.

-exec <command>

Launch an OTScript command.

-execfile <command file>

Execute commands from a file.

-batch

Launch Reqtify in batch mode.

-logfile <fichier>

Set a log file. When this option is set, the debug mode is automatically set.

-hide

Launch Reqtify with hidden main window. All other windows are visible.

-http <port>

Launch Reqtify as socket server. By default, Reqtify server is only accessible from same machine.

-public

Launch Reqtify as public server.

-sync

This option allows you to run the application in X synchronous mode. The synchronous mode forces the X server to perform each X client request immediately and not use buffer optimization. It makes the program easier to debug and often much slower.

Note

This option is Linux-compatible only.

-timeout <s>

Process ends when s seconds has elapsed without any input.

Note

If you are using a batch license, this license is borrowed for you until midnight.

Generating Reports

Reports can be generated with a command line specifying the project to open and the report to produce with their corresponding parameters. The report generation is called using an OTScript method. It is necessary to precede special symbols of the operating system shell with the escape character.

reqtify [-l <eng|fra>] **-batch** [-logfile <file>] **-exec** "openProject(\"<project_path.rqtf>\").{generateReport(\"<report model name>\",\"<outfile.suf>\",\"<template>\"[,args_OTScript_expression]);}"

The file named <*template.suf*> must exist in the *doc_templates* directory in order to the report can be generated (such as *portrait.rtf*). Some reports models do not require any parameter while some others need a list of documents or requirements (or even more).

You can use OTScript expressions such as below:

- If the report needs a list of two documents:

```
documents[name = \"Spec\" OR name = \"Design\"]
```

- If the report needs a requirement:

```
documents.requirements[ident = \"REQ001\"]
```

It is also possible to run an OTScript file which is much easier to maintain and does not require escaping shell characters:

```
openProject("<project path>.rqtf").{
	generateReport("<report_name>","<outfile>.<suf>","portrait"[,args_OTScript_expression
]);
}
```

Example:

scriptfile.ots

```
openProject("<path_to_project>.rqtf").{
	generateReport("My Report1","report1.html","default");
	generateReport("My Report2","report2.html","default");
}
```

This allows the user to generate in a *report1.html* file the *My Report1* report from the *<path_to_project>.rqtf* project. The result format is default. No parameters are needed. The same thing is done for a *My Report2* report.

Note

Report generation with parameters is more complex because it needs to find the elements from the project via OTScript code.

Thereafter, to call the *scriptfile.br* file type the following command:

reqtify -batch -execfile scriptfile.ots

Note

If report model is a project report model, it will not be found because only current project report models are available. This is also true for project templates and filters. To use a project report model, you have to set project as current project as follows:

```
openProject("<path_to_project>.rqtf").{
	THIS.currentProject := EACH;
	generateReport("My Report1","report1.html","default");
}
```

XUI

Introduction

This chapter describes the XUI language (XML User Interfaces) which is a platform independent language for describing user interfaces extensions to the Reqtify GUI.

XUI provides the ability to create most elements found in modern graphical interfaces. It is generic enough that developers can create sophisticated interfaces with it. Some elements that can be created are:

- Input controls such as buttons, textboxes
- Toolbars with buttons
- Menus on a menu bar or popup menus
- Lists
- Trees for display of hierarchical information
- Tabbed panels and group-boxes

The displayed contents can be created from the contents of a XUI file (static contents) or with data resulting from some kind of computation (dynamic contents). The contents of lists, trees, text and other elements can be populated with such XML data.

The dynamic aspects of XUI are handled by code written in OTScript language.

The XUI files are by convention located in the *xui* subdirectory of the *config* directory. The suffix for XUI file is *.xml*. It is possible to amend an XUI file by creating another one with same name except an addition part before the suffix. For example, X.Y.xml is an addition XUI part for X.xml and will be loaded at the same time. To modify main XUI, have a look at *Element <update>* chapter.

Resource Locators

Resources are represented by URIs of the form: *path* or *path#id*. *path* is a sequence of slash separated elements (slash is the / character). If the *path* begins with a slash, the URI is absolute, else the URI is relative. In the process of loading a resource, relative URIs are transformed to absolute URIs by prefixed with the absolute directory path of the document specifying the relative URI.

URIs specify internal resources or external files, depending on the first element of their path:

If the path starts with /res/, the resource is loaded directly from the executable built-in resources. This gives access to a built-in collection of images.

Example

```
<button image="/res/stop.bmp/">

<button image="images/updateTree.bmp/">
```

If the path does not starts with /res/, the resource is loaded from the file system. The file is searched relatively to a root directory corresponding physically to the *config* directory. If the name is a relative filename, *config/xui* directory is the relative starting directory.

The *id* part of the URI specifies the id of an XML node to be searched.

This resource is expected to be a window description providing a menu-bar and a tool-bar definition.

Loading of Resources

When a resource identified by a URI is needed, for example when opening a window, it is loaded into the XUI object memory. The process of loading implies the following steps:

- Instantiation: A copy of the XML tree designated by the URI is instantiated in the object memory. This is to make sure that two windows sharing the same description can be open together, each window having its own XML tree representing its internal state.
- Include resolution: Every <include src="URI"> element found in the XML tree is physically replaced by its definition. The definition is loaded into the memory using the same process.
- Alias resolution: Every XML element which is an alias is replaced by its equivalent form.

The following aliases are provided for user convenience:

- *<hbox>* equivalent to *<box orient="horizontal">*
- *<vbox>* equivalent to *<box orient="vertical">*
- *<spacer>* equivalent to *<box>.*

Common

Attributes

The following attributes are common to several XUI elements. In this document, attributes are presented in three columns tables. For each attribute, the first column contains its name, the second column contains its type, the third column its default value.

The type describes the syntax and semantic of an attribute. The following types are used:

- Boolean: the value true or the value false
- string: any sequence of characters
- identifier: an XML identifier; typically a letter followed by a sequence of letter or digits
- dimension: specify a dimension (width or height) of an element; a dimension specifies a fixed part and/or a proportional (elastic) part.
- orientation: the value vertical or the value horizontal.
- uri: a reference to another resource, using the URI notation.

bkcolor: Color

The *bkcolor* attribute specify background color.

color: Color

The *color* attribute specify foreground color.

displayed: Boolean

The *displayed* attribute specifies if an element is visible. If its value is *false*, the element is hidden. This attribute can be dynamically modified.

enabled: Boolean

The *enabled* attribute specifies if an element is enabled. If its value is false, the element is greyed and unreceptive to user interaction. This attribute can be dynamically modified.

fontheight: Dimension

The *fontheight* attribute specifies the height in points of the font to be used for any text displayed in the element. This attribute cannot be dynamically modified.

fontname: String

The *fontname* attribute specifies the name of the font to be used for any text displayed in the element. This attribute cannot be dynamically modified.

height: Dimension

The *height* attribute specifies the height of an element. The *height* is a dimension. This attribute cannot be dynamically modified.

id: Identifier

The *id* uniquely identifies an element in a XUI file. All XUI widgets can use the id attribute.

Use *id* to access and manipulate XML elements in XUI. This attribute cannot be dynamically modified, i.e. its value is taken into account at the opening of the window.

image: URI

The *image* attribute specifies an icon associated to the element. This image is displayed in a manner depending on the type of the element. If the path starts with /res/, the resource is loaded directly from the executable built-in resources. This gives access to a built-in collection of images.

/res/XXX	Image	/res/eltXXX	Image	/res/flagXXX	Image
new	📄	Requirement	📌	Added	✳
open	📂	Section	📄	Modified	✎
save	💾	Entity	📄	Moved	➡
delete	❌	MacroRequirement	📌	Deleted	☒
Up	⬆	Attribute	▣	Uncovered	❗
Down	⬇	Reference	↱	DerivedRequirement	⬇
undo	↶	Link	🔗	Warning	⚠
redo	↷	Folder	📁	Unknown	❓
copy	📄	Document	📖		
cut	✂	Modificationdocument	📖		
paste	📋	AbstractDocument	📖		
back	⬅	Cover	↗		
forward	➡	Text	🅃		
find	🔍	Image	🖼		
duplicate	📄	eltTable	▦		

label: String

The *label* attribute specifies the textual label associated to the element. This text is displayed in a manner depending on the type of the element.

language: String (internal)

The *language* attribute specifies the language to use for events. By default, it is *internal* and it corresponds to kernel coding. It can be set to *otscript* for user customization.

selbkcolor: Color

The *selbkcolor* attribute specify background selection color.

selcolor: Color

The *selcolor* attribute specify foreground selection color.

tooltiptext: String

The *tooltiptext* attribute specifies a short textual help to be displayed when the user positions the cursor over the widget and waits for a short time. This attribute is intended for tool-bar buttons and menu items. The tool tip is typically displayed in the status-bar, when present. This attribute cannot be dynamically modified.

width: Dimension

The *width* attribute specifies the width of an element. A width is a *dimension*. This attribute *cannot* be dynamically modified.

A dimension is an expression having a fixed part, represented by an integer, and/or a proportional part, represented by an integer followed by a *. The two parts of a dimension, when present, are separated by a +. The * symbol represent one unit of flexibility. When displaying, the total available space is distributed to the elements according to their amount of flexibility.

For example, the attribute *width="150+2*"* specifies a width having 150 pixels as fixed part and 2 units of flexibility.

The attribute *width="32"* specifies a width with only a fixed part: the width of the element does not change when the window is resized.

The attribute *width="2*"* specifies a width with only a flexible part: the width of the element is proportional to the width of the window and changes twice as fast as an element having a width of "*1**".

Events

evdestroy()

Sent when component is destroyed.

evdrag

Sent when user start to drag. Return of callback is Dragged object.

evdragmove(draggedObject : Entity_, nodeAndParentAfter: XmlNode)

Sent when user drag. Return VOID to forbid drop.

evdrop(draggedObject: Entity_, nodeAndParentAfter: XmlNode)

Sent when user drop dragged object.

Callbacks

initcallback

Called on widget initialization.

updatecallback

Called when user interact with window of current widget.

Element <*xui*>

The <*xui*> element is the root element of a XUI description file. A XUI file can contain several definitions. A definition is typically a window, but can also be any element re-used elsewhere using a <include> directive.

Children

Any

Example

```
<?xml version='1.0' encoding='ISO-8859-1'?>

<!DOCTYPE xui SYSTEM "xui.dtd">

<xui>

<window id='win1' ...>

...

</window>

<window id='win2' ...>

...

</window>

</xui>
```

Element <include>

The <*include*> element establishes a link to another source element. When loading a resource containing a <*include*> element, it substituted by its source definition.

Children

None

Attributes

src: URI

Element <window>

The <*window*> element defines a root window.

Children

<box>, <hbox>, <vbox>, <groupbox>, <tabbox>, <spacer>, <label>, <button>, <check>, <textfield>, <listbox>, <duallistbox>, <droplistbox>, <tree>, <graphbox>, <menubar>, <statusbar>, <toolbar>, <include>

Attributes

acceptDragBorder: Boolean (false)

The *acceptDragBorder* attribute, when true, specifies that the window components are sizable.

orient: Orientation

Attribute *orient* is an optional attribute that specifies the alignment of the children of the current element. It can take one of the two values vertical or horizontal. This attribute cannot be dynamically modified.

title: String

The *title* attribute specifies the window label. This attribute can be dynamically modified.

modal: Modal (false)

The *modal* attribute, when set to true, specifies a modal window, i.e. a special kind of window commonly called dialog-box. The opening of a modal window using a call to XUIOPEN() will not return until the user has closed the window. Furthermore, the other windows of the application can't be interacted with until the modal window is closed. This attribute cannot be dynamically modified.

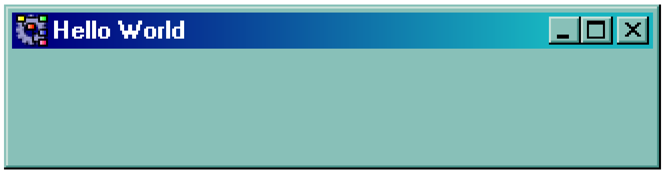
Events

evclose

Sent when a window is about to be closed. This callback can open a message box to ask user if he want to close the window. Should return VOID() when window should really be closed.

Example

```
<window id='idWnd' title='Hello World' width='320' height='55'>

</window>
```



Element <*menubar*>

The <*menubar*> element defines the menu-bar of a window. A menu-bar contains a collection of menus. The menu-bar is automatically located at the top of the window and does not require any location attribute.

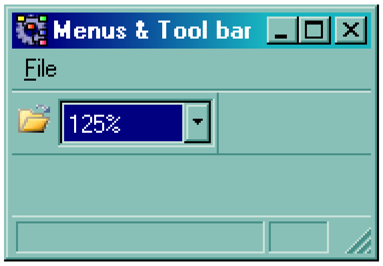
Children

<menu>, <include>

Example

```
<window id='idWnd' title='Menus &amp; Tool bar' width='200' height='155'>
        <menubar>
                <menu label='File' accessletter='F'>
                        <button id='idOpen' label='Open' image='/res/open.bmp' accessletter='p'
ooltiptext='Opens an existing document'/>
                        <button id='idClose' label='Close' accelerator='Ctrl F2'/>
                        <button id='idCloseall' label='Close All' accessletter='A' accelerator='Alt
F3'/>
                </menu>
        </menubar>
        <toolbar>
                <include src='#idOpen'/>
                <droplistbox width='80'>
                        <listitem label='50%'/>
                        <listitem label='100%'/>
                        <listitem label='125%'/>
                </droplistbox>
        </toolbar>
        <statusbar>
                <statusbarfield width='*'/>
```

```
            <statusbarfield width='30'/>
        </statusbar>
</window>
```



Element <*menu*>

The <*menu*> element defines a menu in a menu-bar, a widget or a sub-menu in a hierarchical menu.

Children

<button>, <separator>, <menu>, <include>

Attributes

Accessletter: String

The *accessletter* attribute defines a letter of the label that will act as an access letter when controlling the menu with the keyboard. This letter is usually indicated with an underline when the menu is displayed. This attribute cannot be dynamically modified.

Element <*toolbar*>

The <*toolbar*> element defines a tool-bar widget containing push buttons or drop-down list widgets.

Children

<button>, <separator>, <droplistbox>, <include>

Element <*button*>

The <*button*> element defines a push button widget that can be clicked by the user. It is also used to define menu items within a menu.

Children

None

Attributes

label: String

The *label* attribute specifies the name displayed inside the button. This attribute can be dynamically modified.

image: URI

The *image* attribute specifies an icon displayed inside the button.

checked: Boolean

When *checked* is set to true, a marker of some kind indicates the ”checked” status of the button. If the button describes a menu item, this item is checked, i.e. a tick is displayed at the left of the menu item. If the button describes a push button, the button is displayed in a ”pressed” position. This attribute can be dynamically modified.

accessletter: String

This attribute is used only by buttons describing a menu item. The *accessletter* attribute defines a letter of the label that will act as an access letter when controlling the menu with the keyboard. This letter is usually indicated with an underline when the menu is displayed. This attribute cannot be dynamically modified.

accelerator: String

This attribute is used only by buttons describing a menu item. The *accelerator* attribute defines a keyboard short-cut for a direct activation of the menu command. This attribute cannot be dynamically modified.

- A letter name, e.g. X
- One of the following function keys: F1 to F12, Del, Back, Esc
- The following modifiers can be specified: Ctrl, Alt, Shift. They must be written before the accelerator key, separated with a space.

Some examples of accelerators:

```
accelerator="Ctrl O"

accelerator="Ctrl Alt F12"

accelerator="Shift Del"
```

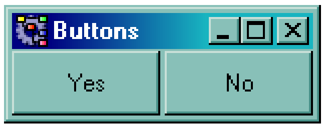
Events

evselect

Sent when button is pressed.

Example

```
<window id='idWnd' title='Buttons' width='150' height='32' orient='horizontal'>
        <button label='Yes'/>
        <button label='No'/>
</window>
```



Element <*separator*>

The <*separator*> element specifies a separation line between item of a menu or a tool-bar.

Children

None

Element <*box*>

The <*box*> widget provides a general purpose and flexible layout mechanism. Using boxes, you can specify the position and relationship of almost any combination of widgets in the UI.

The <*box*> widget allows you to divide a window into a series of boxes. Elements inside a box will orient themselves horizontally or vertically. By combining a series of boxes, spacers and elements with width and height attributes, you can control the layout of a window. A box can lay out its children in one of two orientations, either horizontally or vertically. A horizontal box lines up its elements horizontally and a vertical box orients its elements vertically.

Children

<box>, <hbox>, <vbox>, <groupbox>, <tabbox>, <spacer>, <label>, <button>, <check>, <textfield>, <listbox>, <duallistbox>, <droplistbox>, <tree>, <graphbox>, <toolbar>, <include>

Attribute	**Type**	**Default value**
Common attributes		
id	identifier	

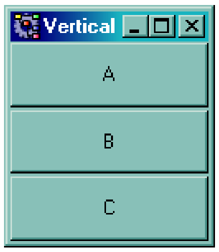
Attribute	**Type**	**Default value**
width	dimension	1*
height	dimension	1*
displayed	Boolean	true
enabled	Boolean	true
Specific attributes		
orient	orientation	vertical

orient: orientation

orient is an optional attribute that specifies the alignment of the children of the current element. It can take one of the two values vertical or horizontal. This attribute cannot be dynamically modified.

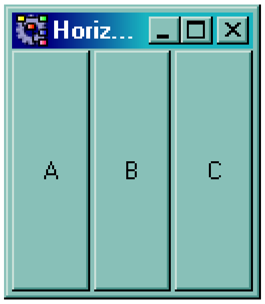
Example

```
<window id='idWnd' title='Vertical' width='120' height='120'>

        <box orient='vertical'>

                <button label='A'/>

                <button label='B'/>

                <button label='C'/>

        </box>

</window>
```



```
<window id='idWnd' title='Horizontal' width='120' height='120'>

        <box orient='horizontal'>

                <button label='A'/>

                <button label='B'/>

                <button label='C'/>

        </box>
```

```
</window>
```



Element <*hbox*>

The <*hbox*> element is an alias for <box orient='horizontal'>.

Element <*vbox*>

The <*vbox*> element is an alias for <box orient='vertical'>.

Element <*spacer*>

The <*spacer*> element provides a spacing widget. The width and height attributes can be set to specify the extent of the spacer.

The <*spacer*> element is an alias for an empty <*box*>. The <*spacer*> notation is recommended when a box has no children and is used only to provide some spacing.

Children

None

Element <*groupbox*>

The <*groupbox*> element can be used to group elements together. A border is drawn around the children elements to show that they are related. The label across the top of the *groupbox* can be specified by using the label attribute.

Children

<box>, <hbox>, <vbox>, <groupbox>, <tabbox>, <spacer>, <label>, <button>, <check>, <textfield>, <listbox>, <duallistbox>, <droplistbox>, <tree>, <graphbox>, <toolbar>, <include>

Attributes

acceptDragBorder: Boolean (false)

The *acceptDragBorder* attribute, when true, specifies that the groupbox components are sizable.

checked: Boolean

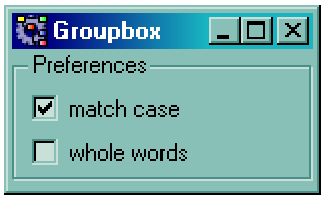
When *checked* is set to true or false, a checkbox appear on right of label. When set to *false*, all children are disabled. This attribute can be dynamically modified.

label: string

The *label* attribute specifies the name displayed across the top of the *groupbox*. This attribute can be dynamically modified.

Example

```
<window id='idWnd' title='Groupbox' width='150' height='68'>
        <groupbox label='Preferences'>
                <check label='match case' checked='true'/>
                <check label='whole words' checked='false'/>
        </groupbox>
</window>
```



Element <*statusbar*>

The <*statusbar*> element defines a status-bar widget containing several display fields, typically located at the bottom of a window.

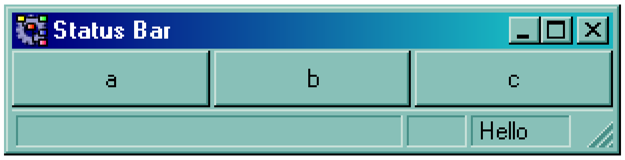
Children

<statusbarfield>, <include>

Example

```
<window id='idWnd' title='Status Bar' width='300' height='48'>
        <hbox>
```

```
            <button label='a'/>
            <button label='b'/>
            <button label='c'/>
        </hbox>
        <statusbar>
            <statusbarfield id='wndw_status_a' width='*'/>
            <statusbarfield id='wndw_status_b' width='30'/>
            <statusbarfield id='wndw_status_c' width='50' value='Hello'/>
        </statusbar>
</window>
```



Restriction: The width of the first *statusbarfield* is necessarily *. The widths of the other fields are necessarily fixed (no * allowed).

Element <*cells*>

The <*check*> element provides a table.

Children

None

Fields

editcomponent: XmlNode

Display component to edit current selection. This component must be a child of cells component.

Element <*check*>

The <*check*> element provides a special button widget that the user can check and uncheck.

Children

None

Attributes

label: String

The *label* attribute specifies the name displayed for the check item. This attribute can be dynamically modified.

checked: Boolean

When *checked* is set to true, the check is selected. This attribute can be dynamically modified.

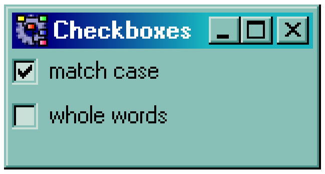
Events

evselect

Sent when button is pressed.

Example

```
<window id='idWnd' title='Checkboxes' width='150' height='55'>
        <check label='match case' checked='true'/>
        <check label='whole words' checked='false'/>
</window>
```



Element <*label*>

The <*label*> element displays an non-editable text.

Children

None

Attributes

value: String

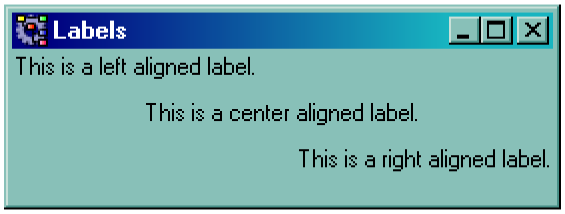
The *value* attribute specifies the textual contents of the text box. This attribute can be dynamically modified.

align: String (left)

The *align* attribute specifies the alignment of the text within the box. Possible values are *left*, *center*, *right* and *topCenter*. This attribute can be dynamically modified.

Example

```
<window id='idWnd' title='Labels' width='270' height='75'>
        <label value='This is a left aligned label.' align='left'/>
        <label value='This is a center aligned label.' align='center'/>
        <label value='This is a right aligned label.' align='right'/>
</window>
```


Labels
This is a left aligned label.
This is a center aligned label.
This is a right aligned label.

style: String

The *style* attribute specifies the style of the label. Possible value is *pushed*.

Element <*textfield*>

The <*textfield*> element creates a box in which the user can enter and modify text.

Children

<*menu*>, <*include*>

Attributes

value: String

The *value* attribute specifies the textual contents of the text box. This attribute can be dynamically modified.

multiline: Boolean (false)

The *multiline* attribute turns the widget from a single line text editor into a multi-lines text editor. This attribute cannot be dynamically modified.

type: String

The *type* attribute can be assigned to the special value password to create a text widget that hides what it types. This is usually used for password entry fields. This attribute cannot be dynamically modified.

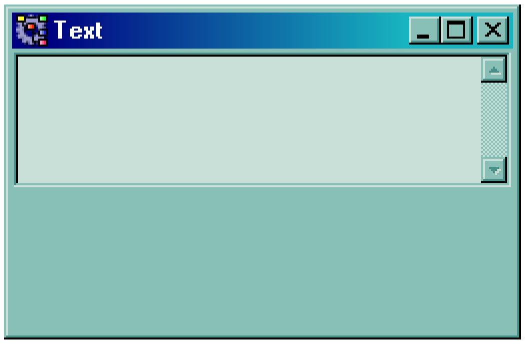
Events

evupdate

Sent when a text content has changed.

Example

```
<window id='idWnd' title='Text' width='250' height='140'>
        <textfield height='1*' multiline='true' />
        <label id='wndw_text_txt' height='1*'/>
</window>
```



Element <*listbox*>

The <*listbox*> element displays a list of items.

Children

<menu>, <listitem>, <include>

Attributes

childNodes: String

Set this attribute value to empty string in order to propagate the addition or deletion of child nodes on display.

multiple: Boolean (false)

The *multiple* attribute specifies a list widget that can have several selected items. This attribute cannot be dynamically modified.

Field

selection: XmlNode

The *selection* field allow to get or set current selection.

selections: XmlNode

The *selection* field allow to get or set current selection in case of multiple selection.

Events

evselect

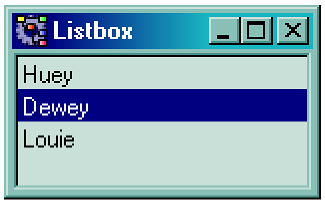
Sent when user select an item.

evdoubleclick

Sent when user double-click on an item.

Example

```
<window id='idWnd' title='Listbox' width='150' height='70'>
        <listbox>
                <listitem label='Huey'/>
                <listitem label='Dewey' selected='true'/>
                <listitem label='Louie'/>
        </listbox>
</window>
```



Element <*listitem*>

The <*listitem*> element defines a single item in a <*listbox*>.

Children

None

Attributes

align: Alignment (center, topLeft, topCenter, topRight, leftCenter, rightCenter, bottomLeft, bottomCenter, bottomRight)

The *align* attribute specifies text alignment.

bkcolor: Color

The *bkcolor* attribute specifies background color.

color: Color

The *color* attribute specifies text color. Following predefined colors can be used:

Name	**Value**	**Color**	**Name**	**Value**	**Color**
aqua	#00FFFF	🟦	navy	#000080	🟦
black	#000000	⬛	olive	#808000	🟫
blue	#0000FF	🟦	orange	#FFA000	🟧
brown	#A02820	🟫	purple	#800080	🟪
chartreuse	#80FF00	🟩	red	#FF0000	🟥
fushia	#FF00FF	🟪	silver	#C0C0C0	⬜
gray	#808080	⬜	teal	#008080	🟦
green	#008000	🟩	violet	#F080F0	🟪
lime	#00FF00	🟩	white	#FFFFFF	
maroon	#800000	🟥	yellow	#FFFF00	🟨

fontbold: Boolean (false)

The *fontbold* attribute can be used to set current font as bold.

fontitalic: Boolean (false)

The *fontitalic* attribute can be used to set current font as italic.

fontundeline: Boolean (false)

The *fontunderline* attribute can be used to set current font as underlined.

imageposition: ImagePosition (left for list, tree, droplistbox and top for matrixlist)

The *imageposition* attribute specifies image position relative to text.

label: String

The *label* attribute specifies the textual label of the item. This attribute can be dynamically modified.

image: URI

The *image* attribute specifies an icon displayed inside the item. This attribute can be dynamically modified.

selected: Boolean (false)

The *selected* attribute specifies a default value for a droplistbox, listbox, tabbox or tree. This attribute cannot be dynamically modified.

Element <*duallistbox*>

The <*duallistbox*> element displays a list of items.

Children

<menu>, <listitem>, <include>

Attributes

multiple

The *multiple* attribute specifies a list widget that can have several selected items. This attribute cannot be dynamically modified.

Field

selection: XmlNode

The *selection* field allows to get or set current selection.

selections: XmlNode

The *selections* field allows to get or set current selection in case of multiple selection.

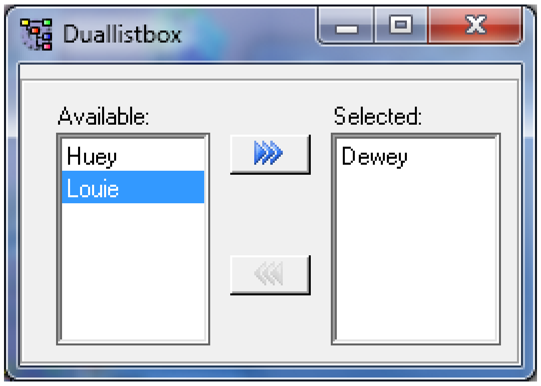
Events

evselect

Sent when user select an item.

Example

```
<window id="idWnd" title="Duallistbox" width="250" height="150">

        <duallistbox label="Choice">

                <listitem label="Huey'/>

                <listitem label="Dewey" selected="true"/>

                <listitem label="Louie "/>
        </duallistbox>
</window>
```



Element <*droplistbox*>

The <*droplistbox*> element displays a list of items in a drop-down menu and holds the selection of one of these items.

Children

<menu>, <listitem>, <include>

Attributes

childNodes: String

Set this attribute value to empty string in order to propagate the addition or deletion of child nodes on display.

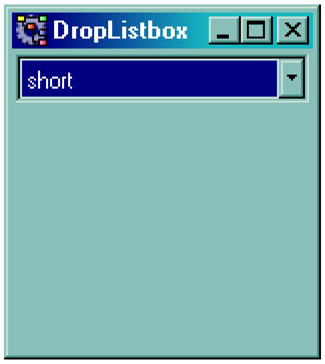
Events

evselect

Sent when user select an item.

Example

```
<window id='idWnd' title='DropListbox' width='150' height='150'>
        <droplistbox id="wndw_droplist_1">
                <listitem label='char'/>
                <listitem label='short'/>
                <listitem label='int' selected='true'/>
        <listitem label='long'/>
        <listitem label='float'/>
        <listitem label='double'/>
    </droplistbox>
    <label id="wndw_droplist_2"/>
</window>
```



Element <*mappingbox*>

The <*mappingbox*> element displays several trees with links between these tree elements and give the possibility to edit these links.

Children

<menu>, <treeitem>, <link>, <include>

Attributes

childNodes: String

Set this attribute value to empty string in order to propagate the addition or deletion of child nodes on display.

Fields

selection: XmlNode

The *selection* field allow to get or set current selection.

Events

evchange

Sent when links will be created or deleted.

evselect

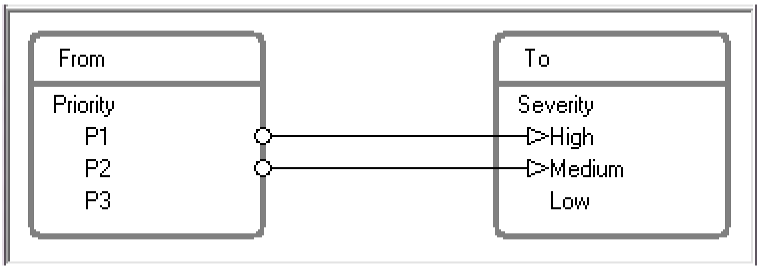
Sent when user select a treeitem or a link.

evupdate

Sent after link creation or deletion.

Example

```
<mappingbox>
        <treeitem label="From" cardinality="1">
                <treeitem label="Priority">
                        <treeitem id="P1" label="P1"/>
                        <treeitem id="P2" label="P2"/>
                        <treeitem id="P3" label="P3"/>
                </treeitem>
        </treeitem>
        <treeitem label="To">
                <treeitem label="Severity">
                        <treeitem id="High" label="High"/>
                        <treeitem id="Medium" label="Medium"/>
                        <treeitem id="Low" label="Low"/>
                </treeitem>
        </treeitem>
        <link from="P1" to="High"/>
        <link from="P2" to="Medium"/>
</mappingbox>
```



Element *<matrixlistbox>*

The *<matrixlistbox>* element displays a matrix list of items.

Children

<menu>, <listitem>, <include>, <directory>

Attributes

childNodes: String

Set this attribute value to empty string in order to propagate the addition or deletion of child nodes on display.

itemheight: Dimension (30)

Height of each element.

itemwidth: Dimension (30)

Width of each element.

type: String

The *type* attribute can be set to *imageNavigator* in order to navigate on images directory. In this case, children tagged *directory* are used to defined root directories.

Fields

selection: XmlNode

The *selection* field allow to get or set current selection.

selections: XmlNode

The *selection* field allow to get or set current selection in case of multiple selection.

Events

evselect

Sent when user select an item.

evdoubleclick

Sent when user double-click on an item.

Element <*radiobox*>

The <*radiobox*> element displays a list of radioboxes or checkboxes depending of *multiple* attribute.

Children

<menu>, <listitem>

Attributes

childNodes: String

Set this attribute value to empty string in order to propagate the addition or deletion of child nodes on display.

multiple: Boolean (false)

The *multiple* attribute can be set to *true*. In this case, radioboxes becomes checkboxes to allow multiple check.

Fields

selection: XmlNode

The *selection* field allow to get or set current selection.

selections: XmlNode

The *selection* field allow to get or set current selection in case of multiple selection.

Events

evselect

Sent when user select an item.

Example

```
<window id='idWnd' title='RadioBox' width='150' height='150'>
        <radiobox multiple='true'>
                <listitem label='One'/>
                <listitem label='Two'/>
                <listitem label='Three'/>
                <listitem label='Four'/>
                <listitem label='Five'/>
        </radiobox>
</window>
```

Element <*table*>

The <*table*> element displays a table.

Children

<titles>, <cells>

Element <*tree*>

The <*tree*> element displays a hierarchy of items.

Children

<menu>, <treeitem>, <include>

Attributes

childNodes: String

Set this attribute value to empty string in order to propagate the addition or deletion of child nodes on display.

multiple (false)

The *multiple* attribute specifies a tree widget that can have several selected items. This attribute cannot be dynamically modified.

propagatechecking (false)

The *propagatechecking* attribute indicates that item checking must be propagated to parent and children.

Field

selection: XmlNode

The *selection* field allows to get or set current selection.

selections: XmlNode

The *selections* field allows to get or set current selection in case of multiple selection.

Events

evselect

Sent when user select an item.

evdoubleclick

Sent when user double-click on an item.

Callbacks

childrencallback(treeitem : XmlNode)

Sent when *treeitem* children are requested. If this callback is not defined or Boolean *computed* attribute is set, already present *treeitem* children are taken into account.

Element <*treeitem*>

The <*treeitem*> element defines a single item in a <tree>.

Children

<*treeitem*>

Attributes

childNodes: String

Set this attribute value to empty string in order to propagate the addition or deletion of child nodes on display.

collapsed: Boolean

The *collapsed* attribute specifies that sub-tree must be opened if set to *true*.

computed: Boolean

The *computed* attribute specifies that *childrencallback* callback does not need to be called. This attribute is generally set during *childrencallback* call.

label: String

The *label* attribute specifies the textual label of the item. This attribute can be dynamically modified.

image: URI

The *image* attribute specifies an icon displayed inside the item. This attribute can be dynamically modified.

Element <*tabbox*>

The <*tabbox*> element is a container for a collection of <*tabbox*> elements.

Children

<tab>, <include>

Element <*tab*>

The <*tab*> element can be used within a <*tabbox*> element to group elements together under a selectable tab.

Children

<box>, <hbox>, <vbox>, <groupbox>, <tabbox>, <spacer>, <label>, <button>, <check>, <textfield>, <listbox>, <duallistbox>, <duallistbox>, <droplistbox>, <tree>, <graphbox>, <toolbar>, <include>

Attributes

acceptDragBorder: Boolean (**false*****)***

The *acceptDragBorder* attribute, when true, specifies that the tab components are sizable.

label: String

The *label* attribute specifies the name displayed within the tab. This attribute can be dynamically modified.

default: Boolean

The *default* attribute, when true, specifies that the tab has to be selected when opening the window. This attribute *cannot* be dynamically modified.

sortIndex: Integer

The *sortIndex* attribute specifies the sort order of the tab in the tabbox.

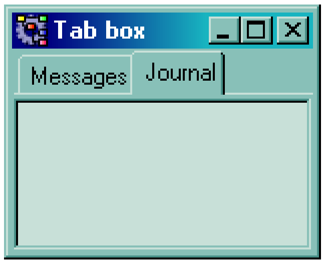
Events

evshow()

Sent when component is shown or hidden.

Example

```
<window id='idWnd' title='Tab box' width='150' height='100'>
        <tabbox>
                <tab label='Messages'>
                        <textfield value='' multiline='true' height='*'/>
                                                        </tab>
                <tab label='Journal' default='true'>
                        <listbox/>
                </tab>
        </tabbox>
</window>
```



Element <*update*>

The <*update*> element is used to modify an existing XUI tree. The goal is to modify an existing GUI without modifying existing XUI files. Here is a list of frequent XUI modifications:

- Add/remove menu item
- Add tab
- Add button
- Modify label

XUI file naming is important. To update an XUI file named xxx.xml, *update* element must appear in an XUI file xxx.yyy.xml.

Children

<box>, <hbox>, <vbox>, <groupbox>, <tabbox>, <spacer>, <label>, <button>, <check>, <textfield>, <listbox>, <duallistbox>, <duallistbox>, <droplistbox>, <tree>, <graphbox>, <tab>, <toolbar>, <include>

Attributes

add: identifier

Add XUI elements as last children of element identified by *add* attribute.

after: identifier

Add XUI elements after element identifier by *after* attribute.

delete: identifier

Delete XUI element identifier by *delete* attribute.

before: identifier

Add XUI elements before element identifier by *before* attribute.

root: identifier

Look for identifier searching from *root* attribute. The goal of this attribute is to precise the search.

set: identifier

Add/set attributes for element identified by *set* attribute. All attributes of update element except *set* attribute are used.

Example

This short example show how to modify central tree contextual menu of Reqtify main window. A command labeled My Command is added to the menu. This command is enabled when a document is selected. When launched, it opens a message box.

Here is XUI contents to be saved in *config/xui/main-menus.myCommand.xml* file:

```
<xui>

        <update add="idFullCentralTreeMenu">

                <button language="otscript" evselect="myCommand"
updatecallback="enableMyCommand" label="My Command"/>

        </update>

</xui>
```

Here is corresponding OTScript code.

```
METHOD Editor.myCommand() : {

        INFO("My Command is not yet implemented");

};


METHOD Editor.enableMyCommand(button : XmlNode) : {

        button.setBooleanAttribute("enabled",selection ISA Document);
```

};

Instructions

Two OTScript instructions are available to process XUI elements:

XUILOAD(anURI: String): XmlNode

Load an XUI file and return associated node.

XUIOPEN(aNode: XmlNode, receiver: Entity_): XmlNode

Display an XUI window node.

DTD of XUI

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ENTITY % XUI.common_attributes ’
id ID #IMPLIED
width CDATA #IMPLIED
height CDATA #IMPLIED
label CDATA #IMPLIED
image CDATA #IMPLIED
tooltiptext CDATA #IMPLIED
annotation CDATA #IMPLIED
displayed CDATA #IMPLIED
enabled CDATA #IMPLIED
oncommand CDATA #IMPLIED
onclick CDATA #IMPLIED ’>
<!ENTITY % XUI.item_attributes ’
id ID #IMPLIED
label CDATA #IMPLIED
image CDATA #IMPLIED
annotation CDATA #IMPLIED ’>
<!ENTITY % XUI.component ’statusbar|toolbar|box|hbox|vbox|
groupbox|tabbox|spacer|label|button|check|
textfield|listbox|duallistbox|droplistbox|tree|graphbox|include’>
```

```
<!ELEMENT xui (window|menubar|%XUI.component;)*>
<!ELEMENT window (menubar?, (%XUI.component;)*)>
<!ATTLIST window
%XUI.common_attributes;
modal CDATA #IMPLIED
title CDATA #IMPLIED
orient CDATA #IMPLIED
>
<!ELEMENT box (%XUI.component;)*>
<!ATTLIST box
%XUI.common_attributes;
orient CDATA #IMPLIED
>
<!ELEMENT vbox (%XUI.component;)*>
<!ATTLIST vbox
%XUI.common_attributes;
orient CDATA #IMPLIED
>
<!ELEMENT hbox (%XUI.component;)*>
<!ATTLIST hbox
%XUI.common_attributes;
orient CDATA #IMPLIED
>
<!ELEMENT spacer EMPTY>
<!ATTLIST spacer
%XUI.common_attributes;
orient CDATA #IMPLIED
>
<!ELEMENT groupbox (%XUI.component;)*>
<!ATTLIST groupbox
%XUI.common_attributes;
orient CDATA #IMPLIED
```

```
>
<!ELEMENT label EMPTY>
<!ATTLIST label
%XUI.common_attributes;
value CDATA #IMPLIED
align (left|center|right) #IMPLIED
>
<!ELEMENT button EMPTY>
<!ATTLIST button
%XUI.common_attributes;
checked CDATA #IMPLIED
accessletter CDATA #IMPLIED
accelerator CDATA #IMPLIED
>
<!ELEMENT check EMPTY>
<!ATTLIST check
%XUI.common_attributes;
checked CDATA #IMPLIED
>
<!ELEMENT textfield (menu?)>
<!ATTLIST textfield
%XUI.common_attributes;
value CDATA #IMPLIED
multiline CDATA #IMPLIED
onedit CDATA #IMPLIED
>
<!ELEMENT listbox (menu?, listitem*)>
<!ATTLIST listbox
%XUI.common_attributes;
multiple CDATA #IMPLIED
>
<!ELEMENT duallistbox (menu?, listitem*)>
```

```
<!ATTLIST duallistbox
%XUI.common_attributes;
multiple CDATA #IMPLIED
>
<!ELEMENT droplistbox (menu?, listitem*)>
<!ATTLIST droplistbox
%XUI.common_attributes;
>
<!ELEMENT listitem EMPTY>
<!ATTLIST listitem
%XUI.item_attributes;
>
<!ELEMENT tree (menu?, treeitem*)>
<!ATTLIST tree
%XUI.common_attributes;
multiple CDATA #IMPLIED
>
<!ELEMENT treeitem (treeitem*)>
<!ATTLIST treeitem
%XUI.item_attributes;
>
<!ELEMENT graphbox (menu?, (node|edge)*)>
<!ATTLIST graphbox
%XUI.common_attributes;
multiple CDATA #IMPLIED
>
<!ELEMENT node EMPTY>
<!ATTLIST node
%XUI.item_attributes;
>
<!ELEMENT edge EMPTY>
<!ATTLIST edge
```

```
%XUI.item_attributes;
from CDATA #REQUIRED
to CDATA #REQUIRED
>
<!ELEMENT tabbox (tab*)>
<!ATTLIST tabbox
%XUI.common_attributes;
>
<!ELEMENT tab (%XUI.component;)*>
<!ATTLIST tab
%XUI.common_attributes;
default CDATA #IMPLIED
>
<!ELEMENT statusbar (statusbarfield*)>
<!ATTLIST statusbar
%XUI.common_attributes;
>
<!ELEMENT statusbarfield EMPTY>
<!ATTLIST statusbarfield
%XUI.common_attributes;
value CDATA #IMPLIED
>
<!ELEMENT menubar (menu|include)*>
<!ATTLIST menubar
%XUI.common_attributes;
>
<!ELEMENT menu (button|separator|menu)*>
<!ATTLIST menu
%XUI.common_attributes;
accessletter CDATA #IMPLIED
>
<!ELEMENT toolbar (button|droplistbox|separator|include)*>
```

```
<!ELEMENT separator EMPTY>
<!ELEMENT include EMPTY>
<!ATTLIST include
src CDATA #REQUIRED
>
```

Presentations

Introduction

A Presentation is a graphical display. The goal of presentation is to present dashboards. For example, Reqtify Management View has two presentations.

A presentation is defined as a tree of presentation structures (label, bar, project) and data coming from Reqtify data model.

Presentations are saved in XML files. There is no specific editor to create presentations. These XML files are located in config/presentations directory.

Element <PresentationList>

The <*PresentationList*> element is the root element.

Children

<*Presentation*>

Attributes

style: uri

CSS style sheet.

Element <Presentation>

The <*Presentation*> element corresponds to a graphical display.

Children

<DataParameter>, <Structure>

Attributes

name: *String*

Name of presentation.

type: String

Type of Presentation. It is always equal to *sheet*.

receiverClass: Class

Class name of receiver.

receiverName: String

Name of receiver.

wishedWidth: Integer

Wished width.

Element <DataParameter>

A *<DataParameter>* element allows to parameterize its parent.

Children

<DataTerminal>, <DataMethodCallText>

Attributes

name: *String*

Name of parameter.

class: Class

Class of parameter. It can be *String*, *Color*.

Element <DataTerminal>

A *<DataTerminal>* element defines a constant. It can be a text, a string, an integer, a color or an OTScript code

Attributes

class: Class

Return class. This attribute must be define for code. For other types, it is automatically deduced.

code: Code

OTScript code. This OTScript code must return a value matching attribute class.

color: *Color*

Color. Refer to color table in this document to have more information about possibilities.

integer: String

Integer contant.

string: String

String contant.

text1: String

English constant text.

text2: String

French constant text.

text3: String

Chinese constant text.

text4: String

Japanese constant text.

Element <Structure>

A <*Structure*> element allows you to display a graphical part.

Children

<DataParameter>, <Structure>

Attributes

type: *String*

Type of structure. It can be *barSection*, *barLabel*, *button*, *horizontalBar*, *label*, *project*, *projectCover*, *table*.

class: *String*

Class coming from style sheet.

Element <DataMethodCallText>

A *<DataMethodCallText>* element allows to call a function returning a terminal (string, integer, real).

Attributes

method: *String*

Method name.

receiver: *String*

Variable name containing the receiver of the method.

Element <DataMethodCallLoop>

A *<DataMethodCallLoop>* element allows to call a function returning an object list.

Attributes

condition: *Code*

OTScript condition.

method: *String*

Method name.

receiver: *String*

Variable name containing the receiver of the method.

variable: *String*

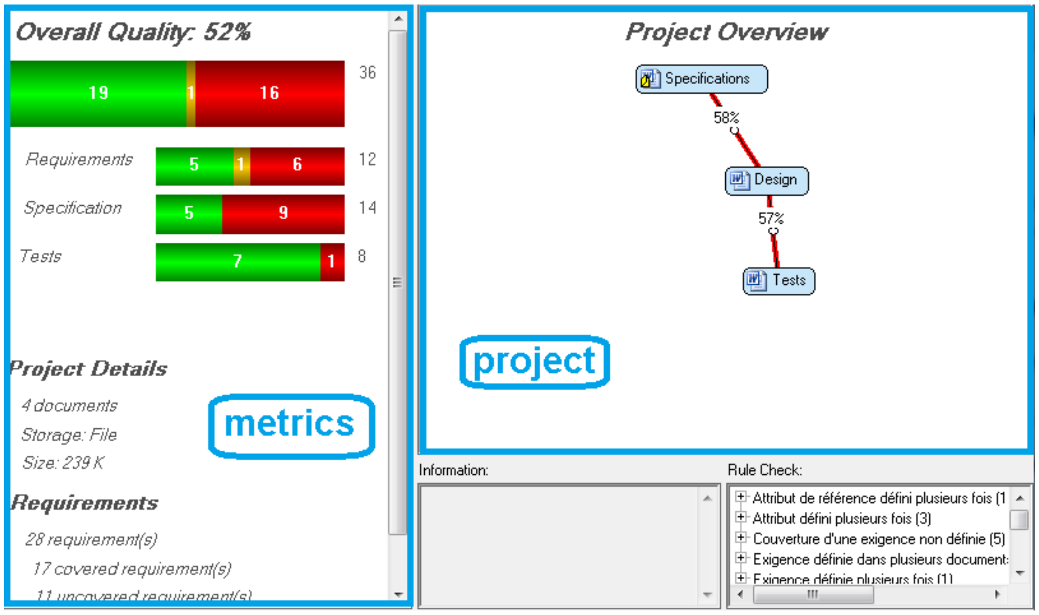
Variable name identifying the object.

Instructions

An OTScript instructions is available to process presentations:

GETPRESENTATION(name: String): Presentation_

Find a presentation by name. For example, in Management View, 2 presentations are present : *metrics* and *project*.



Attribut de référence défini plusieurs fois (1
Attribut défini plusieurs fois (3)
Couverture d'une exigence non définie (5)
Exigence définie dans plusieurs document:
Exigence définie plusieurs fois (1)

Example

Create an XML file with following content:

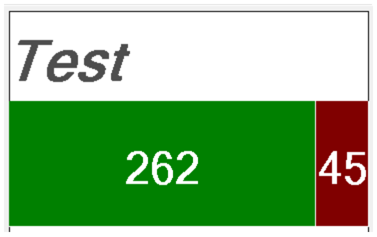
```
<PresentationList style="default.css">
         <Presentation name="myPres" type="sheet" receiverClass="Project"
receiverName="project" wishedWidth="4000">
           <DataParameter name="drawColor" class="Color">
                   <DataTerminal color="black" />
           </DataParameter>
           <Structure type="label" class="title">
                   <DataParameter name="string" class="String">
                           <DataTerminal text1="Test" />
                   </DataParameter>
           </Structure>
                   <Structure type="horizontalBar" class="mainBar">
                           <Structure type="barSection">
                                   <DataParameter name="size" class="Integer">
                                           <DataTerminal
code="numberOfCoveredLeafRequirements" class="Integer" />
                                   </DataParameter>
                                   <DataParameter name="fillColor" class="Color">
                                           <DataTerminal
color="gradient(verticalCentered,green,lime)" />
                                   </DataParameter>
                                   <DataParameter name="string" class="String">
                                           <DataTerminal
code="numberOfCoveredLeafRequirements" class="Integer" />
                                   </DataParameter>
                           </Structure>
                           <Structure type="barSection">
                                   <DataParameter name="size" class="Integer">
                                           <DataMethodCallText
method="numberOfUncoveredLeafRequirements" receiver="project" />
```

```
                                        </DataParameter>
                                        <DataParameter name="fillColor" class="Color">
                                               <DataTerminal
color="gradient(verticalCentered,maroon,red)" />
                                        </DataParameter>
                                        <DataParameter name="string" class="String">
                                               <DataMethodCallText
method="numberOfUncoveredLeafRequirements" receiver="project" />
                                        </DataParameter>
                                 </Structure>
                          </Structure>
                 </Presentation>
</PresentationList>
```

In Reqtify, select a document and open evaluator to enter following OTScript code:

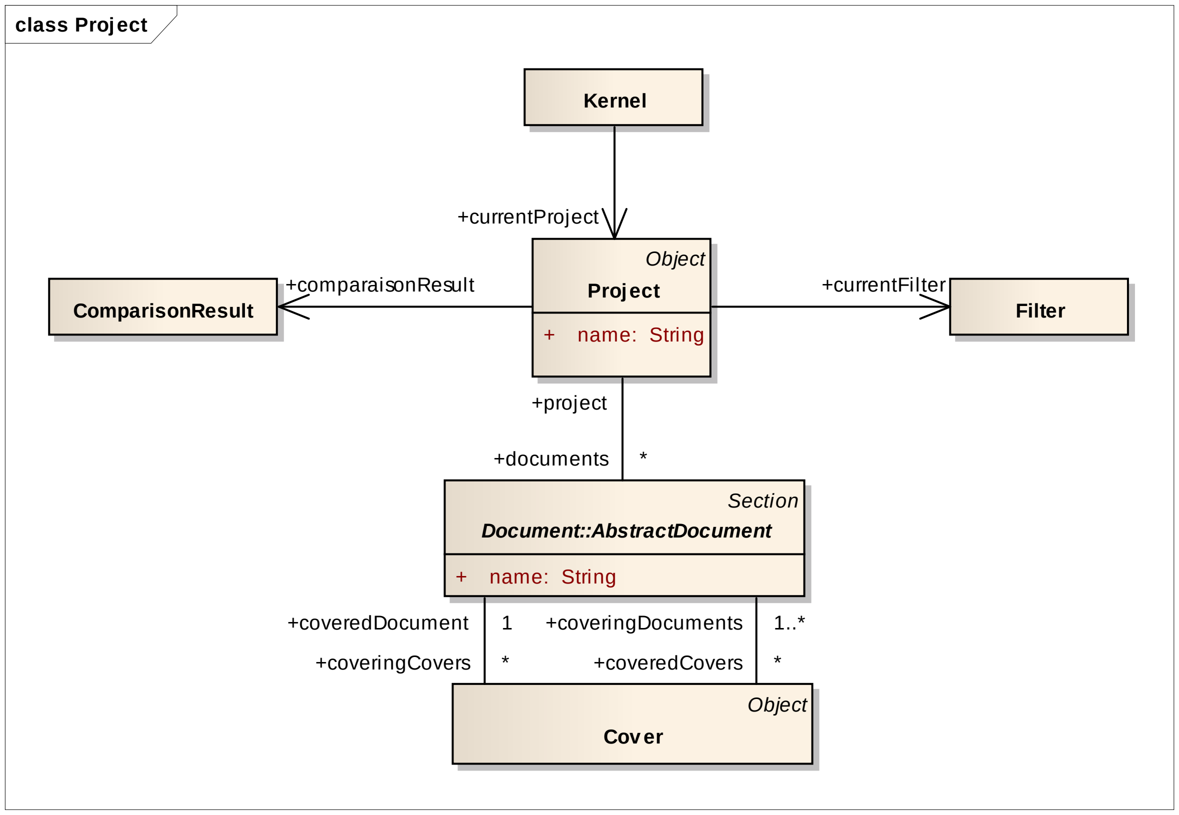
```
GETPRESENTATION("myPres").asImage(project)
```

Display will be following:

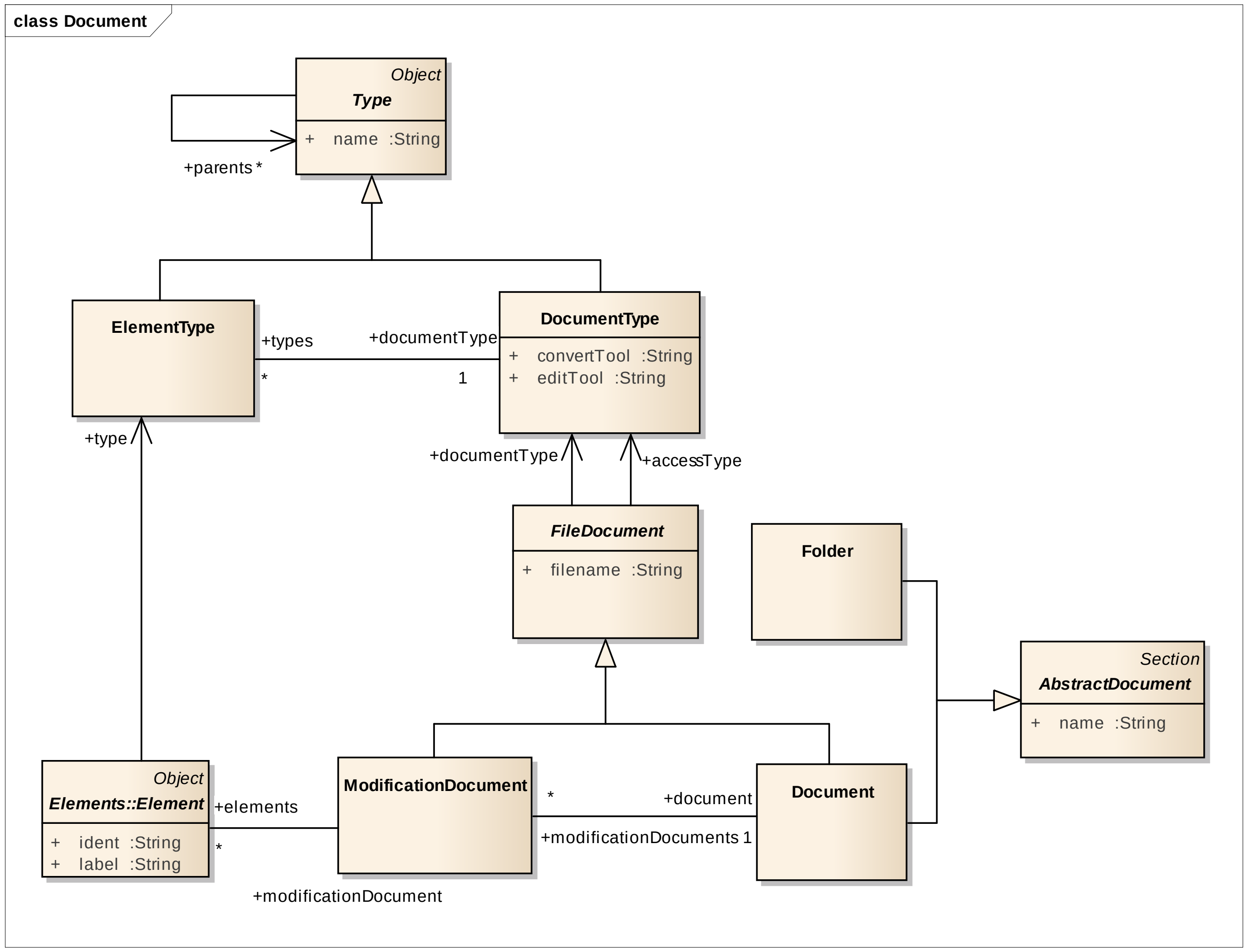


Reqtify Data Classes

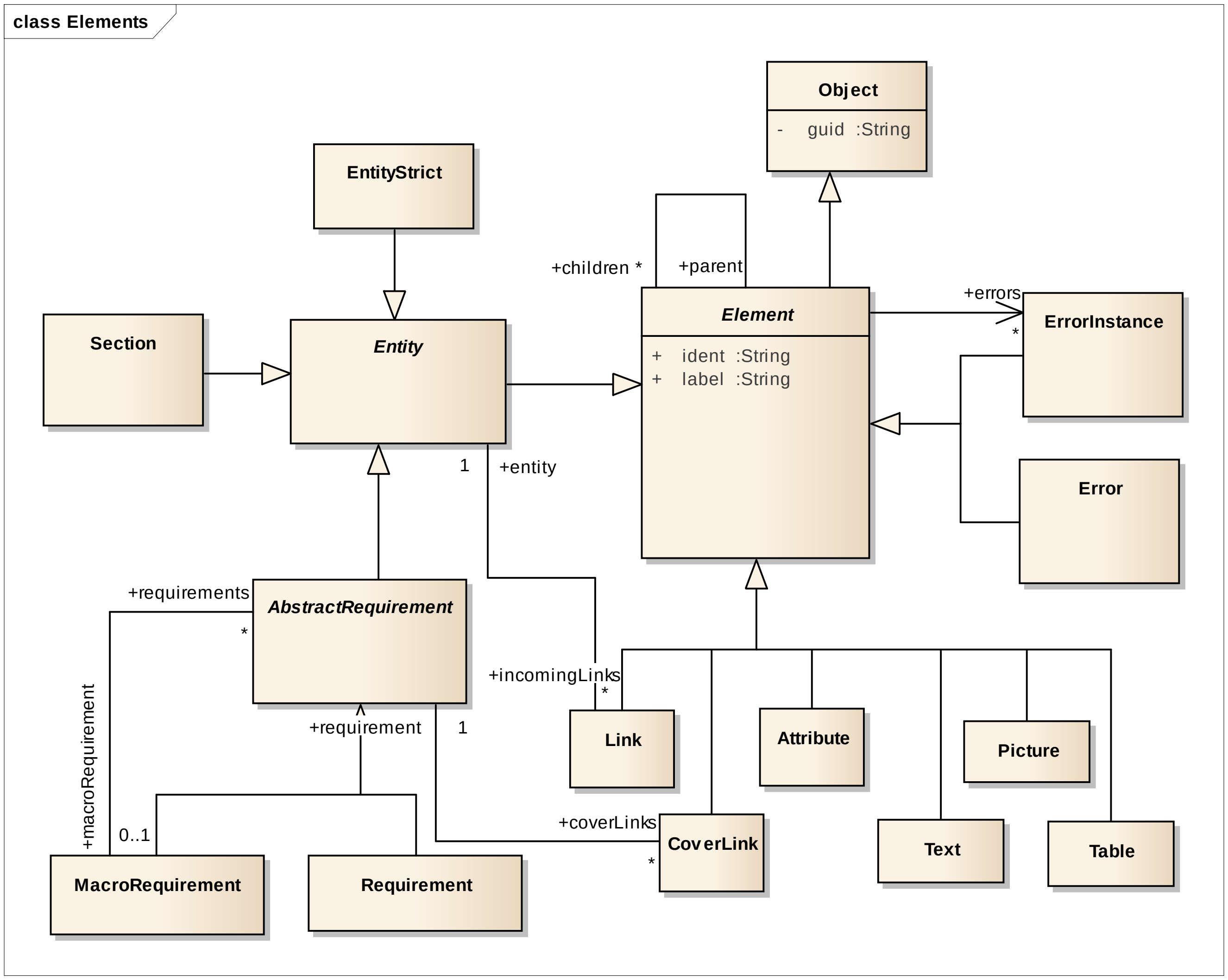
Schema



Project



Documents and types



Elements

AbstractDocument: Section

Super-class of documents and folders.

AbstractDocument attributes

category: String

Category of the document.

coveredCovers: Cover

Covers covered by the document.

coveringCovers: Cover

Covers covering the document.

editorPositionX: Integer

Abscissa of document box center.

editorPositionY: Integer

Ordinate of document box center.

filename: String

Filename of document.

project: Project

Project containing the document.

publicFilename: String

Filename as it is entered by user.

AbstractDocument methods

allVisibleAttributes: Attribute

Return all visible attributes contained in the document.

allVisibleCoverLinks: Attribute

Return all visible cover links contained in the document.

allVisibleLinkAttributes: Attribute

Return all visible link attributes contained in the document.

allVisibleEntities: Entity

Return all visible entities contained in the document.

computeGlobalRatio(): Integer

Compute global coverage ratio for all requirements present in current document.

computeRatio(someCoveringDocuments : AbstractDocument): Integer

Compute coverage ratio for leaf requirements covered by entities present in *someCoveringDocuments*.

computeRequirementRatio(someCoveringDocuments : AbstractDocument): Integer

Compute coverage ratio for requirements covered by entities present in *someCoveringDocuments*.

newCoveredCover(aDoc : AbstractDocument): Cover

Create a new cover to document *aDoc*.

AbstractLink class

Cover link or link. Parent is covering entity.

AbstractLink attributes

entity: Entity

Target entity.

AbstractLink methods

attributes({typeNames : String}): LinkAttribute

Return link attributes of abstract link. If *typeNames* is defined, return only attributes of type named *typeNames*.

newAttribute(type : ElementType{, value : String}): LinkAttribute

Create a new link attribute. If value is not defined, create a Boolean link attribute.

AbstractRequirement: Entity

Super-class of Requirement and Macro-Requirement classes.

AbstractRequirement attributes

coverLinks: CoverLink

Return links covering this requirement.

macroRequirement: MacroRequirement

Macro-requirement containing the requirement. Macro-requirements can contain other macro-requirements.

AbstractRequirement methods

attributes(): Attribute

Returns attributes plus macroRequirement attributes.

isBasicallyCovered(): Boolean

Test whether requirement is completely covered or not. This method is called by *uncovered* rule.

isBasicallyDerived(): Boolean

Test whether requirement is derived or not. This method is called by *derived* rule.

isCovered(): Boolean

Test whether requirement is completely covered or not. This test is based on absence of *uncovered* errors. If *uncovered* errors are present, *isCovered* return false.

isDerived(): Boolean

Test whether requirement is derived. This test is based on presence of *derived* errors.

isUnknown(): Boolean

Test whether requirement is undefined.

mirroredRequirement(): AbstractRequirement

Return mirrored requirement.

mirroringRequirements(): AbstractRequirement

Return mirroring requirements.

requirementReferencers (): AbstractRequirement

Returns referencers of type requirement.

ComparisonResult Class

Result of comparison between two projects or two documents.

ComparisonResult methods

isEmpty(): Boolean

If true, there are no differences.

newAbstractEntities(): Entity

Return new entities.

oldAbstractEntities(): Entity

Return deleted entities.

modifiedAbstractEntities(): Entity

Return modified entities.

movedAbstractEntities(): Entity

Return moved entities.

newChildren(anEntity: Entity): Element

Return new children for a modified entity.

oldChildren(anEntity: Entity): Element

Return deleted children for a modified entity.

modifiedChildren(anEntity: Entity): Element

Return modified children for a modified entity.

CoverLink: AbstractLink

Cover link. Parent is covering entity.

CoverLink attributes

requirement: AbstractRequirement

Requirement covered by the link.

CoverLink methods

associatedCovers: Cover

Return covers associated to cover link.

Cover: Object

A cover is covering an abstract document and is covered by one or more other documents.

Cover attributes

category: String

Cover category.

coveredDocument: AbstractDocument

Covered document.

coveringDocuments: AbstractDocument

List of documents covering *coveredDocument* document.

editorPositionX: Integer

Abscissa of cover center.

editorPositionY: Integer

Ordinate of cover center.

Document: AbstractDocument, FileDocument

Document attributes

timeStamp: Integer

Time stamp of the document.

modificationDocuments: ModificationDocument

Modification documents associated with document.

Document methods

beUpToDate(): Document

Set document as up to date.

getUsedElementTypes(aClass : Class_): ElementType

Get element types used in document by elements not filtered.

updateFromXml(aNode: XmlNode): Document

Replace document contents with XML node contents.

newModificationDocument(aType: DocumentType): newModificationDocument

Add a modification document to a document. If *aType* is null, document type will be used.

Document events

afterRename(String)

Called after document is renamed. Argument is old document name.

DocumentType: Type

DocumentType attributes

convertTool: String

Return convert tool name.

editTool: String

Return edit tool name.

types: Type

Types contained in document type.

intermediateFileEncoding: String

Intermediate file encoding: default (ANSI), UTF8, UCS2.

DocumentType methods

findType(name: String): Type

Find the type having the specified name.

isDirectory(): Boolean

Check whether document type is a directory type. A directory type is a type that needs a directory selection.

newType(class: Class): Type

Create a new type for elements of class *class*.

```
dt.newType(CLASS(Requirement)
```

save(): DocumentType

Save file associated with document type.

Element: Object

This is the root class of all element classes.

Element attributes

actualDocument: FileDocument

Current document of element. Get or set document or modification document.

ident: String

Element identifier.

label: String

Element short label.

parent: Element

Parent of element.

document: Document

Document of element.

root: AbstractDocument

Root of current element tree. Can be a document or a folder.

type: Type

Type of element.

typeName: String

Type name of element.

children: Element

Element children.

errors: ErrorInstance

Errors associated with element.

image: Image

Small icon associated with element.

Element methods

actualFilename(): String

Current file name of element.

addErrorMessage(identifier: String, message: String, level: Integer, parameter: String): ErrorInstance

Adds an error identified with *identifier* and message *message* to receiver. *level* parameter can be set to 1 for errors or 2 for warnings. This error message will appear in main Reqtify window under the *Rule check* section.

edit(): Element

Navigate action as done with menu item *Navigate*.

image(): Image

Returns icon associated with element.

imageFilename(): String

Returns icon filename associated with the element. If the icon is internal, a temporary file with right image extension is created.

imageName(): String

Returns icon name.

isCreatedByApplication(): Boolean

Is element created from Reqtify GUI.

isVisible(): Boolean

Tests whether element is visible in GUI.

ElementType: Type

Type of elements.

ElementType attributes

enumerationValues:String

Returns enumerate values for an attribute type.

identFormat: String

Format of identifiers.

isBoolean: Boolean

Tests whether type is Boolean.

isMultiple: Boolean

Tests whether type is multiple.

analysisTree: XmlNode

For XML type, analysis tree.

printExpression: String

Printing expression.

printTextExpression: String

Prints expression for texts.

ElementType methods

newEnumerationValue(name : String [, label : String]): EnumerationValue

Creates a new enumeration value.

Entity: Element

Class of objects those can support attributes, text, and reference requirements.

Entity attributes

incomingLinks: Link

Links for those the entity is the target.

Entity methods

allReferences(): AbstractRequirement

Return all requirements directly or indirectly covered by the entity.

attributes(): Attribute

Return attributes.

getURL(action : String): String

Computes a Reqtify URL. When selecting this URL in a navigator, Reqtify opens entity project and select the entity. Parameter action can be one of following:

- *display*: select element in Reqtify main window (default)
- *edit*: open element in original document

newAttribute(type : ElementType{, value : String}): Attribute

Creates a child attribute.

newCoverLink(type : ElementType, target : Requirement): CoverLink

Creates a child cover link.

newLink(type : ElementType, target : Entity): Link

Creates a child link.

newPicture(image : Image): Picture

Create a child picture.

newRequirement(type : ElementType, ident : String{, label : String}): Requirement

Create a child requirement.

newText(string : String): Text

Creates a child text.

pictures(): Image

Returns pictures.

references(): AbstractRequirement

Returns covered requirements.

visibleParents(): Entity

Returns visible parents. If its parents are not visible, its grandparents are returned and so on.

EnumerationValue: Entity

Class of enumeration values.

EnumerationValue attributes

name: String

Name of enumeration value.

label: String

Label of enumeration value.

EntityStrict: Entity

Class of entities.

Error Class

Class of error types.

ErrorInstance: Element

Class of errors.

ErrorInstance attributes

label: String

Explanation of rule.

FileDocument Class

Document or modification document.

FileDocument attributes

filename: String

Absolute file name of file document.

intermediateAccessFilename: String

Name of intermediate access file.

intermediateFilename: String

Name of intermediate file. When intermediate checkbox is not checked, return a temporary filename.

publicFilename: String

File name entered by user.

documentType: DocumentType

Type of document.

accessType: DocumentType

Access type of document.

project: Project

Project containing document.

FileDocument methods

browse(): FileDocument

Open a navigator to choose a document.

currentFilename(): String

Absolute file name of file document.

getVariableValue(aVariableName: String): String

Return value of variable *aVariableName*.

getBooleanVariableValue(aVariableName: String): Boolean

Return:

- VOID if *aVariableName* is not defined;
- FALSE: value is equal to '0' or 'false'
- TRUE: all other cases

setVariableValue(aVariableName: String, aValue: String): FileDocument

Set value of variable *aVariableName*.

Filter Class

A display filter.

Filter attributes

name: String

Filter name.

isAnalysis: Boolean

Test whether filter is an analysis one.

description: String

Return description of filter.

comment: String

Return comment of filter.

fullDescription: String

Get complete filter description.

showLinksToHiddenEntity: Boolean

Show links to hidden entities. If this flag is set to *FALSE*, target entities are transformed as undefined requirements.

showPictures: Boolean

Show pictures or not.

showTexts: Boolean

Show texts or not.

showUndefined: Boolean

Show undefined requirements or not.

Filter methods

addCondition(classA : Class_, classR : Class_, docTypeName : String, eltTypeName : String, operatorName : String{, value : String}): Filter

Add a condition to a filter. This condition is applicable on object of class *classA*. The receiver is of class *classR* and of type with name *eltTypeName* and document type name *docTypeName*. The operator is named *operatorName*.

```
{
  TMP filter := project.newFilter("My Filter");
  filter.addCondition(CLASS(AbstractRequirement), CLASS(Attribute),
"Word", "Allocation", "isAbsent");
}
```

cloneStandalone(): Filter

Duplicate a filter without adding it to project.

getParameterValue(name : String) : String

Get value of a filter parameter.

setParameterValue(name : String, value : String) : Filter

Set value of a filter parameter.

showAbstractDocument(aDocument: AbstractDocument, isVisible: Boolean): Filter

Set visibility of documents or folder.

showType(aType: Type, isVisible: Boolean): Filter

Set visibility of element typed *aType* to *isVisible*.

showEntity(anEntity: Entity): Filter

Force display of entity *anEntity*.

project: Project

Project containing document.

Folder: AbstractDocument

A folder.

Folder methods

newDocument(aType: DocumentType): Document

Add a new to document to current folder.

Kernel Class

Functional kernel. Allow to load projects.

Kernel attributes

currentProject: Project

Get or set current project.

temporaryDirectories: String

Directories considered as temporary by Reqtify. A project located in one of these directories is not added to the recent project list. This attribute shall be modified by using += operator.

Kernel methods

getProjectFilenamesUsingDocumentFilename(aFilename: String): String

Get file names of project referencing document file name.

getProjectProperties(aFilename: String): Dictionary

Get project properties.

newProject(aFilename: String): Project

Create a new project.

openProject(aFilename: String[, someOptions: String]): Project

Open a project without making it current one.

someOptions can be following:

- RELOAD_ALL: reload all documents
- CONVERT_ALL: convert all documents
- RELOAD: <doc name>: reload specific document name
- CONVERT: <doc name>: convert specific document name

- FORCE_YES: answer automatically 'yes' to all questions
- FORCE_NO: answer automatically 'no' to all questions
- DO_NOT_ADD_IN_RECENT_LIST: do not add in recent file list in file menu.
- DO_NOT_SHOW_STOP_BUTTON: prevent from user interruption
- PASSWORD: <password>: open project with specific password

quit(): Kernel

Quit application.

reloadProject([someOptions: String]): Project

Reload project.

reportModelLists(): ReportModelList

Return available Report Model Lists.

setCurrentProject(project : Project, [someOptions: String]): Project

Set current project with optional options:

- DO_NOT_ADD_IN_RECENT_LIST: do not add in recent file list in file menu.

Kernel events

afterCrash(String)

Called after Reqtify crash.

autoExec()

Called when starting.

autoExit()

Called when exiting. GUI is not available when autoExit is called. So, GUI instruction like *INFO()*, *REQUESTFILER()* are not available.

afterProjectChange()

Called after current project change.

afterSaveReports(ReportModelList)

Called after reports are written.

beforeSaveTypes(DocumentType)

Called before types are written.

beforeProjectChange(Project)

Called before current project change.

Link: AbstractLink

Links other than cover links.

MacroRequirement: AbstractRequirement

Macro-requirement class.

MacroRequirement attributes

requirements: AbstractRequirement

Requirements/macro-requirements included in macro-requirement.

Object class

This is the root class of all object classes.

Object attributes

guid: String

Object GUID.

Object methods

print(): String

Display of element.

Mark: Entity_

Class of marks.

Mark attributes

name: String

Mark name.

text: String

Text of mark.

elements: Element

Elements marked by the mark.

ModificationDocument: FileDocument

Class of modification documents.

ModificationDocument attributes

document: Document

Document to which the modification document is attached.

elements: Element

Elements belonging to modification document.

ModificationDocument methods

newAttribute(entity : Entity, type : ElementType): Attribute

Create a new attribute on entity *entity*.

newCoverLink(entity : Entity, target : Requirement): CoverLink

Create a new cover link on entity *entity*.

newLink(entity : Entity, target : Entity): Link

Create a new link on entity *entity*.

newLinkAttribute(entity : Entity, coverLink : CoverLink): LinkAttribute

Create a new link attribute on cover link *coverLink*.

newText(entity : Entity, string : String): Text

Create a new text on entity *entity*.

Picture: Element

Picture class.

Picture attributes

picture: Image

Real picture.

Project: Object

Project class.

Project attributes

applicationVersion: String

Return current version of tool.

canEdit: Boolean

If true, project can be modified.

comparisonResult: ComparisonResult

Return comparison result produced during reloading.

currentFilterName: String

Get or set filter name.

currentFilter: String

Get or set filter.

diagram: Image

Return display of project.

directory: String

Project directory.

documents: Document

Documents included in project.

documentTypes: DocumentType

Document types included in project.

documentsAndFolders: AbstractDocument

All documents and folders included in project.

filename: String

Project filename. Extension of filename is .rqtfimage.

name: String

Project name.

projectFilename: String

Project filename. Extension of filename is .rqtf.

rootAbstractDocuments: AbstractDocument

Root documents of project.

ruleNamesToIgnore: String

Names of rule not evaluated during rules evaluation.

versionName: String

Version name.

versionDescription: String

Version description.

Project methods

copyFilter(aFilter: Filter): Filter

Create a new filter from an existing one.

deleteFilter(aFilter: Filter): Filter

Delete a filter.

errors(): ErrorInstance

Project error instances.

errorTypes(): ErrorType

Project error types.

filters(): Filter

Project filters.

findElement(anIdent: String): Element

Find an element by its name in project tree.

getOrCreateUnknownRequirement (aRequirementName: String): Requirement

Get or create a new requirement named *aRequirementName*.

generateReport(reportName: String, filename: String, templateName: String, [arg1: Entity_, [arg2: Entity_]]): Project

Generate a report in file named filename. filename suffix is used to determine output format. If template name is already set in ReportModel, *templateName* can be *VOID(String)*.

```
myProject.generateReport("Traceability Matrix", "c:\matrix.rtf", "portrait")
```

generateReport2(reportName: String, dg: DocGenerator, [arg1: Entity_, [arg2: Entity_]]): Project

Generate a report using report generator dg.

```
myProject.generateReport("Traceability Matrix", dg)
```

getDocumentTypeFromFileExtension(extension: String): DocumentType

Find a document type having extension as one of associated extension.

```
getDocumentTypeFromFileExtension("docx") ⇒ DocumentType(Word)
```

getOrCreateMark(name : String, text : String[, color : Integer])

Get a mark by its name or create it with specified *name*, *text* and *color*.

graphicalView()

Get graphical view as an image.

marks()

Get marks associated to the project.

newDocument(aType: DocumentType): Document

Add a new to document to current project.

newDocumentType(aType: Integer): DocumentType

Create a new document type of type aType (1: textual, 2: XML, 3: added elements).

newFilter(aName: String): Filter

Create a new filter.

replaceDocuments(someDocuments : AbstractDocument, aDocument : AbstractDocument): Filter

With a composite project, replace one or more documents by another one.

save({aFile : String{, options : String}}): Project

Save a project. if *aFile* is set, save as a new file name. *options* parameter can be:

- *PURGE_TYPES* to remove unused types from project. This option only works on .rqtfimage project.
- *VERSION: "CURRENT"|"PRESERVE"|"<v>"* to save as current project version (default), to preserve .rqtfimage version or to save as a particular Reqtify project version:

v	46	51	61	64	67	69	70	73	75	80	81
internal	4.0	4.1	4.2	4.3	4.4	4.5	4.6	5.0	5.1	5.2	5.3
public	2009-1	2010-1	2011-1	2012-1	2012-2	2013-1	2013-2	2015	2015 FD01	2016	2016 FD01

saveSnapshot(aVersionName: String, aVersionDescription: String): String

Save a snapshot of current project.

snapshotNames(): String

Return all associated snapshot file path.

snapshots(): Project

Return all associated snapshots. This method call can be very costly because of snapshot contents loading.

unknownRequirements(): Requirement

Return all undefined requirements.

Project events

afterCopy(String)

Called after project is copied. Argument is old file name.

afterFilterChange()

Called when current filter is changed.

afterLoad(Kernel)

Called after project is loaded. This event is called after *afterNew* or *afterOpen* event.

afterNew(Kernel)

Called after project is created.

afterOpen(Kernel)

Called after project is opened.

afterRename(String)

Called after project is renamed. Argument is old project name.

afterSave()

Called after project is saved.

afterReport(ReportModel, String, Kernel)

Called after report is generated. Second argument is file produced by report.

afterSnapshot(String, Kernel)

Called after snapshot is created. Argument is snapshot name.

beforeClose(Kernel)

Called just before project is closed.

beforeConvert()

Called before project is converted.

beforeFilterChange()

Called before filter change. In this event, isPresent attribute can be set to FALSE and it will remain FALSE during filter application.

beforeImageSave(Kernel)

Called before .rqtfimage is saved.

beforeReport(Report)

Called before report is generated.

beforeSave(Kernel)

Called before project is saved.

Requirement: AbstractRequirement

Requirement class.

Section: Entity

Section class.

Section methods

compare(anOldSection: Section): ComparisonResult

Compare two sections and return comparison result.

Text: Element

Text class.

Type: Object

Type of elements.

Type attributes

name: String

Type name.

elementClass: Class_

Class of elements of this type.

regularExpression: String

For textual type, regular expression of the type.

endExpression: String

For textual type, end regular expression of the type.

subExpression: String

For textual type, sub regular expression of the type.

documentType: DocumentType

Type of document containing this type.

parents: Type

Parents of type.

children: Type

Children of type.

imageName: String

Image name.

image: Image

Type icon.

Type methods

isEnum(): Boolean

Test whether type is enumerated.

isInverse(): Boolean

Test whether link type is inverted.

isModifiable(): Boolean

Test if type can be used to create elements from Reqtify. It can be an automatic type, an internal type, or a type having a modified connector.

isRich(): Boolean

Test whether text is rich or not.

isXml(): Boolean

Test whether type is XML.

richText(): RichText

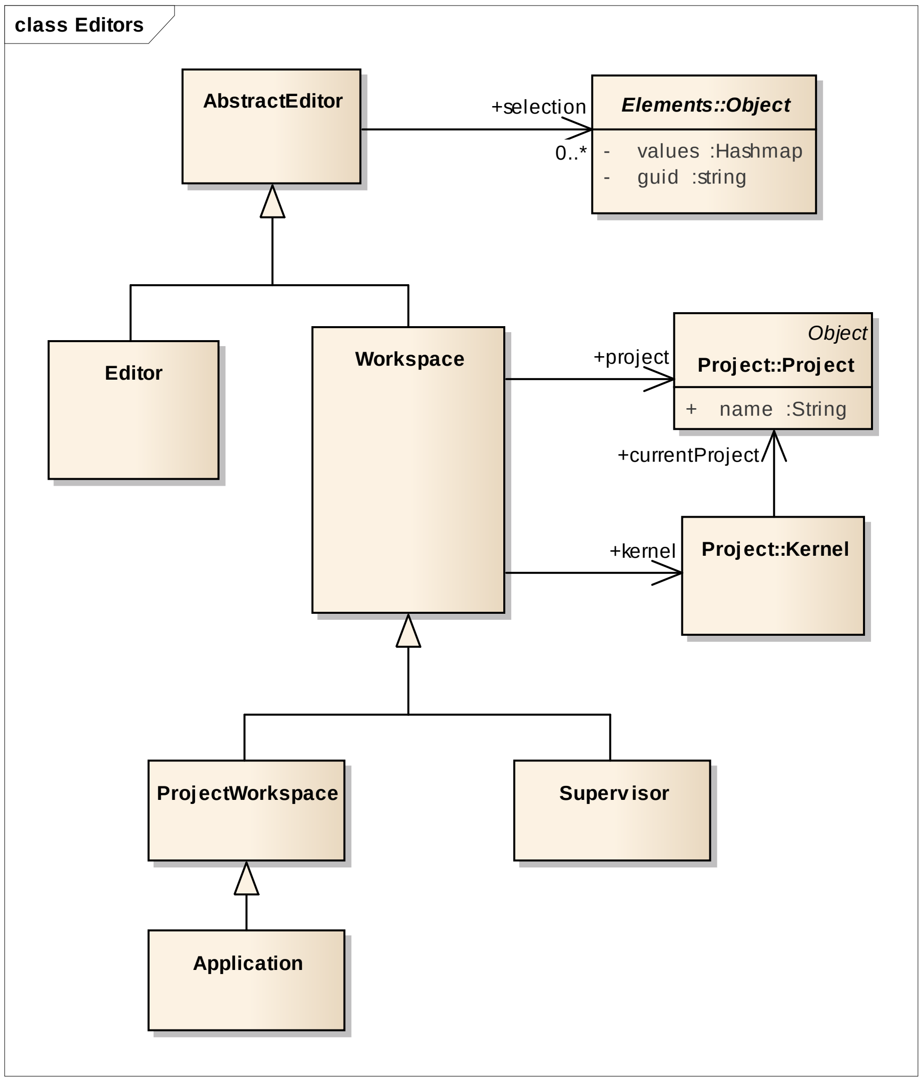
Return rich text.

setFieldIndexKind(aFieldIndex: Integer, aTypeIndex: Integer): Type

Set group index for a component. aTypeIndex: 1: identifier, 2: label, 3: text, 4: GUID.

Reqtify Editor Classes

Schema



AbstractEditor

AbstractEditor class. Super classes of all editors.

AbstractEditor attributes

selection: Element

Elements currently selected in window.

AbstractEditor methods

activate()

Raise window.

refresh()

Refresh editor display.

Editor

Editor class.

Application: ProjectWorkspace

Application class. Correspond to main Window. Does not exist if application is launched in batch mode.

Application attributes

recentProjectFilenames: String

Recent project filenames.

Application methods

openProject(aFilename: String[, someOptions: String]): Project

Open a project and set this project as current one.

Application events

afterFilterChange()

Called after filter change.

afterProjectChange()

Called after project change.

afterSave()

Called after project is saved.

afterSelectionChange()

Called when current selection change.

ProjectWorkspace: Workspace

Workspace class.

Supervisor: Workspace

Supervisor attributes

managedOpenFiles: String

Managed files currently opened.

Workspace: Editor

Workspace attributes

kernel: Kernel

Kernel.

project: Project

Edited project.

Workspace methods

newEditor(): Editor

Open a new editor.

Workspace events

beforeActivatingEditor(XmlNode)

Called after editor is activated. Parameter is corresponding XUI node.

Examples

Example 1: List and Count Requirements for each Document

OTScript script:

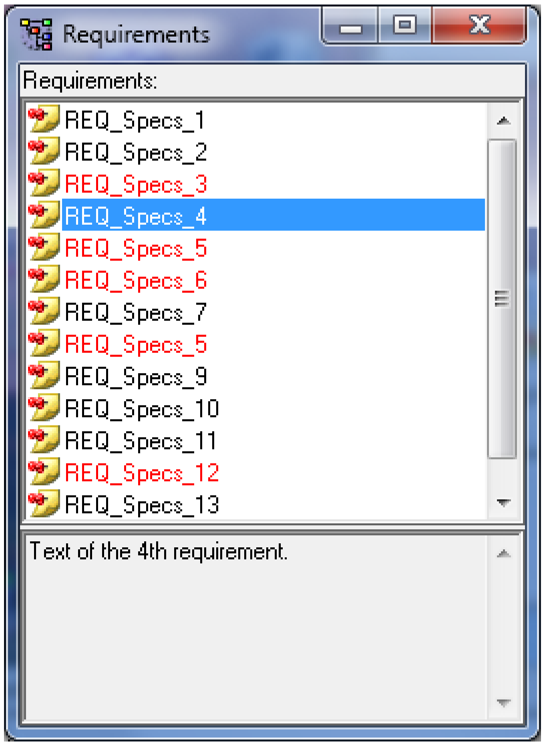
```
{
        TMP prj:= kernel.currentProject;
        OPENBUFF();
        prj.documents.{
                PUTF("$1 ($2): $3"n", name, $CNT(requirements), requirements.ident);
        };
        CLOSE();
}
```

VBS script:

```
set appli = GetObject(, "Reqtify.Application")
set prj = appli.kernel.currentProject
set docs = prj.documents
result = ""
for each doc in docs
        result = result & doc.ident & "(" & doc.requirements.count & "):"
        for each req in doc.requirements
                result = result & " " & req.ident
        next
next
MsgBox result
```

Example 2: Display all Requirements in a Window

Following source code add an *Open MyGUI* menu item in *Tools* menu and open following window:



Text of the 4th requirement.

XUI (config/xui/reqs.xml):

```
<xui>

        <window id="idReqs" title="Requirements" language="otscript" width="200"
height="300">

                <label value="Requirements:"/>

                <listbox id="idList" initcallback="initList" evselect="selectReq"/>

                <textfield id="idText" multiline="true" readonly="true" height="100"/>

        </window>

</xui>
```

OTScript (config/otscript/reqs.br):

```
CLASS MyGui : Gui;


ATTRIBUTE MyGui.document : Document;
```

```
METHOD MyGui.initialize(aDocument : Document) : {
        document := aDocument;
        rootNode := XUILOAD("/reqs.xml#idReqs");
};

METHOD MyGui.open() : {
        XUIOPEN(rootNode, THIS);
};

METHOD MyGui.initList(aList : XmlNode) : {
        aList.newChildren("listitem", document.requirements);
};
METHOD MyGui.selectReq(aList : XmlNode) : {
        TMP sel := aList.getField("selection")[EACH ISA XmlNode];
        TMP wText := rootNode.getElementById("idText");
        TMP str := sel.entity[EACH ISA Requirement].text;
        wText.setAttribute("value", str) ;
};

METHOD Document.openMyGui() -menu : {
        TMP gui := NEW(MyGui);
        gui.initialize(THIS);
        gui.open();
} LABEL "Open MyGUI";
```

Glossary

Functions

Attributes

Methods

Events