The CARA/Lua Programmers Manual

Rochus Keller, PhD
rkeller@nmr.ch

Document NMR.014
Version 1.9.1
2015-10-01

compatible with CARA 1.9.1

http://www.cara.nmr.ch

Introduction

CARA (Computer Aided Resonance Assignment) is the application for the analysis of NMR spectra and computer aided resonance assignment developed at the group of Prof. Dr. Kurt Wüthrich, Institute of Molecular Biology and Biophysics, ETH Zürich. CARA is completely platform independent, has a sophisticated graphical user interface and a build-in scripting language based on Lua 5.1. In this manual we will focus on scripting and the CARA/Lua Application Programming Interface (API) which allows custom scripts to access the CARA Object Model.

For a detailed description of the Lua programming language used by CARA, please refer to the "Lua 5.1 Reference Manual" (LRM, [1]) or to [2]. The conceptual model of CARA is described in [3], the user interface and its application in [4]. The CARA Wiki [5] offers many scripts for different use cases.

Referenced Documents

[1] Ierusalimschy, R.; de Figueiredo, , L. H.; Celes, W.: Lua 5.1 Reference Manual. August 2006. ISBN 85-903798-3-3, Lua.org.

[2] Ierusalimschy, R.: Programming in Lua. 2nd Edition, March 2006. ISBN 8590379825. Lua.org.

[3] Keller, R.: Optimizing the Process of Nuclear Magnetic Resonance Spectrum Analysis and Computer Aided Resonance Assignment. 2004. Dissertation ETH Nr. 15947.

[4] Keller, R.: The Computer Aided Resonance Assignment Tutorial. 2004. ISBN 3-85600-112-3. CANTINA Verlag, Goldau.

[5] Damberger, F.; et al.: The CARA Wiki. http://wiki.cara.nmr.ch/FrontPage.

[6] Keller, R.; Grace, C.R.; Riek, R.: Fast multidimensional NMR spectroscopy by spin-state selective off-resonance decoupling (SITAR). Magnetic Resonance in Chemistry. 2006 Jul 6;44(S1):196ff.

[7] Wüthrich, K.: NMR of Proteins and Nucleic Acids. 1986. ISBN 0471828939. Wiley, New York.

[8] Trolltech AB: Qt Reference Documentation (Open Source Edition). Qt 4.3.5. 2008. URL: https://doc.qt.io/archives/4.3/index.html (Retrieved September 2015)

A brief Lua overview

In this section I give a short overview over the most important concepts of the language and its libraries. For a full description please refer to the LRM [1].

Lua has straight forward syntax and semantics. It is a language with dynamic typing, which means you dont have to specify a data type for your variable as you might be used to from Pascal or Fortran. You create a variable by just assigning a value to a name.

```
valueA = 3
valueB = 3.5       -- valueB gets a floating point number
valueC = 314.16e-2
valueD = "this is a string"
valueE = true      -- a boolean value
valueF = nil
valueB = valueD    -- valueB now also has a string value

a, b, c = 10, 20, 30  -- multiple assignments can be done at once
                      -- a is now 10, b is 20 and c is 30
```

These variables, as soon they receive their value, are globally accessible to all scripts you might run (unless the variable is prefixed with the `local` keyword). The value might be changed by later assignments and even receive a value of another data type. Variables carry their data type with them, transparent to the user. The example shows *basic types*, which are **number** (integer or floating point), **string**, **boolean** or **nil**.

Comments start with "--" and go to the end of the line. You can also write multi line comments using the following syntax:

```
--[[ this is a long comment
     going over more than one line ]]--
```

Don't hesitate to write extensive comments, since they don't cost any performance and because experience shows that programs are written once but read many more times.

Besides handling scalar values and strings (i.e. basic types), Lua allows you to create data structures, as you might know them e.g. from Pascal as records and arrays. Lua offers so called *tables* as a central concept of data structuring. You can use tables to represent both records and arrays. In contrast to basic types, tables have to be constructed before they can be used. The variable then contains a *reference* to the table (not the table itself).

```
tableA = {}              -- construct an empty table
tableA[ 1 ] = 3          -- index by number, like an array
tableA[ 2 ] = "a text"
tableA[ valueA ] = 3.4   -- index by value of a variable (=3)
tableA[ "abc" ] = 3e-10  -- index by string, like a record
tableA.abc = "hello"     -- short form looks even more record like

tableB = {[1] = 3,       -- a constructor with initialization
          [2] = "a text",
          abc = "hello" }
tableA.sub = tableB      -- tables can be nested
tableA.sub.abc = "test"  -- accessing the nested table
```

In the above example tableB points to the same table as tableA.sub. So equality `tableB.abc == tableA.sub.abc` always holds.

Lua supports the usual arithmetic operators like "**+**" for addition, "**-**" for subtraction and negation, "*****" for multiplication, "**/**" for division and also "**^**" for exponentiation. These operators work with numbers but also with strings, as far they can automatically be converted to numbers. Lua obeys the usual operator precedence (first "*" and "/", then "+" and "-", etc.), which can be changed by using brackets.

```
res = "3" + 4 * 2       -- res becomes 11
res = 2 ^ 3             -- res becomes 8
res = ( "3" + 4 ) * 2   -- res becomes 14
```

Lua offers the following relational operators: "**==**" for equality comparison and "**~=**" for unequality. There are also the usual "**<**", "**<=**", "**>**" and "**>=**" with the expected meaning. These operators all produce a boolean value (**true** or **false**). Basic types are compared in the usual way. Tables are compared by their reference, not by the values they contain. If you want to compare tables by their contents, you have to write a comparison function iterating over the elements of the table yourself (see below).

Boolean values can be related using the logical operators "**and**", "**or**" and "**not**". Logical operators can also be applied to nil (with the meaning of false). The values of all other data types are considered true (the number 0 is also considered true!). The "or" operator returns the value of the first operand being true which is quite handy to select values.

```
res = 10 or "abc"       -- res becomes 10
res = nil or "abc"      -- res becomes "abc"
res = 10 and 20         -- res becomes 20
res = false and 20      -- res becomes false
```

In Lua you can define your own functions or use existing functions from a library (e.g. the math library specified in LRM). Functions are called as follows:

```
res = math.cos( math.pi )      -- res becomes -1
res = math.max( 1, 3, 4, 2 )   -- res becomes 4

-- functions can return multiple values:
l, o, s = spec.decomposeLabel( "CA-1" )    -- l becomes "CA"
                                           -- o becomes -1
                                           -- s becomes 1
```

Functions are objects themselves and can be assigned to variables or passed as arguments. That's why the following to variants of function declarations are equal:

```
add = function( a, b ) return a + b end    -- variant 1

function add( a, b ) return a + b end      -- variant 2

res = add( 2, 3 )       -- res becomes 5
add2 = add
res = add2( 2, 3 )      -- res also becomes 5, it's the same function

function swap( a, b )   -- another function with a local variable
     local temp = a;    -- local variables and parameter (a, b
     a = b              -- and temp) are only "alive" during
     b = temp           -- the execution of the function.
     return a, b        -- more than one return value is allowed
end
```

Since functions can be assigned to arbitrary variables, you can equip Lua tables with functions. With this possibility you can do "object oriented" programming. See the following examples:

```
bill = {}               -- create an empty table called "bill"

bill.sayHi = function()  -- associate a function with field "sayHi"
     print( "Hi!" )      -- of table "bill", see variant 1
end

function bill.sayHi()    -- this is syntactic sugar for doing
     print( "Hi!" )      -- the same definition, see variant 2
end

bill.sayHi()             -- prints "Hi!" to the terminal
```

```
bill.age = 30            -- give bill an age

function bill.printAge( me )      -- define another function
     print( "My age: "..me.age )  -- ".." does string concatenation
end

bill.printAge( bill )    -- prints "My age: 30" to the terminal

-- Lua offers syntactic sugar for the above case
function bill:printAge()
     print( "My age: "..self.age )   -- self is an automaic variable
end                                  -- provided by Lua

bill:printAge()          -- prints "My age: 30" to the terminal also
                         -- notice the use of the ":" instead of "."
```

The CARA/Lua API makes extensive use of the ":" syntax, since resulting code becomes much more readable.

Remember that variables keep their values until they go out of scope (in case of local variables) or you explicitly assign nil to them. This is especially important if you assign a huge table (with tousands of elements) to a global variable. The garbage collector of Lua will keep this huge table in memory as long as a variable is referenicing it. It is therefore a good habit to either use functions and local variables or to explicitly assign nil to all global variables when they are no longer used. You can ease your life if you create your own scope, i.e. a single global table, and assign everything you create to elements of this table (also your functions). With this convention you reduce the risk of accidentally overwriting a Lua or CARA object with the same name, and finally you can simply assign nil to your scope table and everything will be garbage collected.

```
myScope = {}
myScope.valueA = 3
myScope.valueB = 3.5
myScope.valueC = 314.16e-2
myScope.valueD = "this is a string"
myScope.add = function( a, b ) return a + b end
...
myScope = nil      -- garbage collector goes to work

collectgarbage()   -- you can even start the collector yourself
```

Lua offers the usual control and loop statements like **if**, **while**, **repeat** and **for**. The for statement is a bit special, since it exists in two forms, one specially suited for table iteration. Let's have some examples.

```
-- an if condition. elseif and else are optional
if weather == "rainy" then
     print( "Don't forget the umbrella" )
elseif weather == "sunny" then
     print( "Keep smiling" )
else
     print( "Don't care" )
end
```

```
-- a while loop
while success == false do
     success = keepTrying()
end

-- a repeat loop
repeat
     done = doSomething()
until done == true

-- an infinite while loop with a break statement
while true do
     done = doSomething()
     if done then
          break
     end
end

-- a for loop which runs ten times
for i = 1, 10 do print( i ) end

-- iterating through a table
table = { a = "HA", [2] = "HB", c = "HG" }
for key, value in pairs( table ) do
     print( key.."="..value )
end
-- prints three lines: a=HA c=HG 2=HB
```

For further examples and advanced features like meta-tables please refer to the LRM.

The Lua/Qt Binding

A Bit of History

The development of CARA and its predecessors (Spectroscope, Backbone, Sidechain, see [3] page 127) started in 2000. From the beginning the software was envisaged to be platform independent, since at that time Silicon Graphics and Sun machines were very common in the NMR community, and also Linux and Macintosh were becoming more widespread (Windows was mostly used in the office).

NMR software at that time was usually written in Fortran and C. Xeasy was fully written in C and had a real GUI based on Xlib and Motif. It was a no-brainer that CARA would use C++ since that language was state of the art, very well established, compatible with most relevant libraries and much more efficient than other object oriented languages. At that time Qt was already a very promising C++ framework, but not the only one (e.g. also Gtk+, WxWindows, Fltk or libTK were an option, just to name a few). I eventually decided for Qt (version 2.1 at that time) because it was fully C++, statically buildable and had an advanced architecture.

But Qt was mainly used as a platform abstraction of the windowing system and the basic services. Only a small fraction of the CARA/NAF source code was directly dependent on Qt. The throughout dependability of Xeasy on Motif turned out to be quite a migration obstacle. From that experience I wanted to avoid to spread third-party APIs all over the CARA/NAF source code; instead Qt was capsuled in a fundamental layer. The window frames, menues and basic dialog boxes were rendered by Qt; all window contents and interactivity was provided by the NMR Application Framework (NAF), which I specifically wrote for that purpose (the Lexi module was inspired by the Design Pattern book by Gamma et al. and the InterViews framework on which the book was based, among others). The resulting architecture is shown in figure 1.

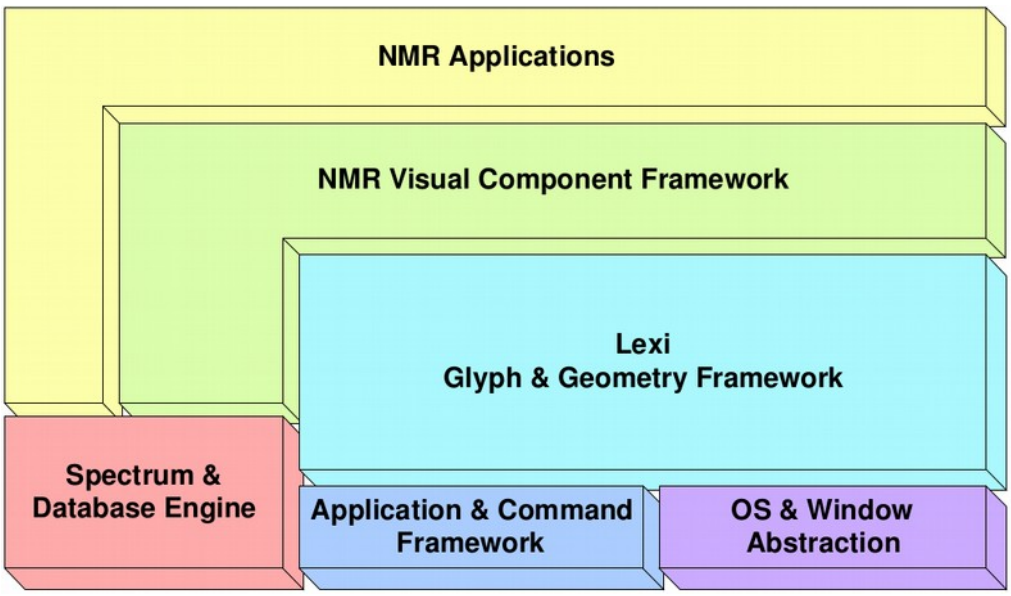


Figure 1: CARA Software Architecture (Source: [3] Figure 70)

CARA/NAF did well many years with Qt 2.x; only in 2006 I eventually migrated to Qt 4.1 which was quite an effort. The latest Qt version used so far is 4.4.3, but CARA also compiles well under Qt 4.8 (but not yet under Qt 5 due to the missing Qt3 backward compatibility layer still in use in CARA/NAF).

When I started the CARA 1.9 series in 2009 Qt has already become part of the "world cultural heritage" like the C++ standard template libraries. It therefore did no longer make sense to strictly capsule it. Qt has become even so powerful in meantime, that it seemed worthwhile to replace and rewrite parts of the NAF to make use of the new Qt features. I consequently also added Qt classes to the Lua binding; this turned out to be quite straightforward due to the powerful meta object system of Qt which allows for exposing slots and properties to Lua with little effort. The major effort I spent in writing the Lua binding for the required value classes as well as the methods of QObject descendants which are neither slots nor properties. Within the scope of this effort I also rewrote most of the GUI components accessible through the `gui` library(see page 163 and figures 3, 4 and 5). These classes are now based on the Qt/Lua binding described in the following sections; because of this you sometimes get Qt class names when requesting the type of an object (instead the ones shown in the referenced figures); the big advantage is of course that now also the slots and properties of the Qt classes are available to the GUI components.

The following three sections address people interested in the Lua/Qt binding specifics. You can still use the CARA Object Model even if you're not familiar with these specifics. You therefore may directly jump to page 24 and ignore the Lua/Qt binding until you really need it.

General Features of the Binding

This section applies to the Lua/Qt binding as well as to the CARA Object Model (see page 24). For each class represented in the binding there exists a Lua table which is accessible by the name of the class in the global namespace. This table contains the methods of the class. The following example iterates through the class "QPoint".

```
Lua> for name, func in pairs( QPoint ) do print( name ) end
setY
manhattanLength
setX
y
x
isNull
init
new
__call
```

You can replace methods by simply assigning another function to the corresponding name in the table. This even works with inheritance as we will see later. Of course the new function must have the same parameters like the replaced one; otherwise you will most likely cause some runtime errors.

In the above example the functions `init`, `new` and `__call` belong to the general features of the binding. You will see them in each creatable class when executing the above example.

The `init` method is automatically called by the binding whenever a new instance of the class is created; it is used to set the initial values of the instance; most classes have a built-in `init` method. If the class is creatable at all, an instance can be created using the following syntax:

```
Lua> p = QPoint( 2, 3 )
Lua> print( p:x() )
2
Lua> print( p.class )
QPoint
Lua> p.data.test = 12345
Lua> print( p.test )
12345
```

The arguments - "2, 3" in the above example - are handed over to the `init` method; but since `init` is a method, the first parameter is alway a reference to the instance itself (the reader might know the "this" keyword in C++ or Java, which does the same thing). The actual call of the binding is therefore `QPoint.init( p, 2, 3 )` in the above example; the notation with parentheses is only "syntactic sugar". The function `__call` is only needed to make this work and can be ignored otherwise. __call just hands over to `new` which creates the instance and calls `init`. You could likewise write `p = QPoint.new( 2, 3 )` with the same result as `p = QPoint( 2, 3 )`.

The example shows yet two other features of the binding: the `class` and `data` attributes. The `class` attribute is supported by each instance created by the binding; it returns the name of the class of which the instance was created. As an alternative to `class` you can also write `type` which does the same thing. In addition the `data` attribute is also supported by each instance; it gives the user access to the data table of the instance; the data table is an ordinary Lua table associated with the instance; it can be used to store arbitrary Lua values associated with the instance. When the name doesn't collide with a built-in attribute or method, "`.data`" can be left out; in the above example the "`.test`" attribute is the same, regardless whether `p.data.test` or only `p.test` is written.

The Binding of Qt Value Classes

Qt includes a lot of classes with value semantics such as QPoint or QString. CARA currently supports a binding for the following Qt value classes (in alphabetical order):

QBitArray, QBitmap, QBrush, QChar, QColor, QDate, QDateTime, QFont, QFontInfo, QFontMetrics, QIcon, QImage, QKeySequence, QLine, QMatrix, QPainterPath, QPalette, QPen, QPixmap, QPoint, QPolygon, QRect, QRegion, QSize, QSizePolicy, QString, QStringList, QTime and QUrl.

These are essentially the classes which can be stored in a QVariant. Please refer to [8] for documentation. CARA has implemented a subset of the methods of these classes. You can iterate over the table accessible by the class name to find out which are available.

Whenever a Qt function or method expects a QString as parameter, you can also pass an ordinary Lua string. CARA will interpret this string as Latin-1. If a Qt function or method returns a QString, you can either operate directly on the QString instance or use the Lua function `tostring()` to convert the return value to an ordinary Lua string. The conversion results in a Latin-1 string.

Note that QPoint, QPolygon, QRect und QSize are actually implemented using their floating point equivalents (e.g. QPointF, etc.). The binding automatically manages the conversion where necessary. Also note that there is no binding for QVariant. This is unnecessary since Lua values are already "variants" by themselves.

Keep in mind that the classes mentioned above are treated like tables by Lua, i.e. a variable doesn't store the instance itself, but only a reference to the instance. Therefore the following example applies:

```
Lua> p1 = QPoint( 2, 3 )
Lua> p2 = p1
Lua> p2:setX( 10 )
Lua> print( p1:x() )
10
```

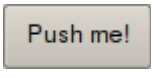
The Lua binding for value classes has the built-in method `clone()` to create a new copy of an instance (but note that the data table is not automatically copied). So you can write the following:

```
Lua> p1 = QPoint( 2, 3 )
Lua> p2 = p1:clone()
Lua> p2:setX( 10 )
Lua> print( p1:x() )
2
Lua> print( p2:x() )
10
```

The Binding of QObject Descendants

QObject and its decentants have properties, signals and slots; for an explication of the meaning of these terms and the documentation of the classes please refer to [8]. Qt has its own meta object system which allows to query the methods of a class and call them by name at runtime. The Lua binding for QObject descentants maps the slots and properties to attributes, similar to tables (remember that in Lua `table["abc"]` means the same as `table.abc`, see page 16). Let's make an example:

```
Lua> p = QPushButton()
Lua> p.text = "Push me!" -- QAbstractButton has the property "text"
Lua> p:show() -- according to [8] QWidget has the slot "show()"
```

After executing these lines a window should open which looks like this: 

CARA offers a binding for a large number (but by far not all) of QObject descendants. For each biniding at least the slots and properties are accessible as shown above. For quite many of these classes the binding also supports a subset of the other methods (which are no slots or property accessors). To find out whether the binding supports a class and which of its methods are accessible simply try to iterate over the method table of the class as shown in the example on page 20. If the class is not supported, you will see an error message, otherwise the list of available methods.

Even if properties are mapped to attributes you can still use the attributes `class`, `type` and `data` as described on page 20. These names therefore hide Qt properties with the same name. Refering to the above example the expression `p.data` would still return the data table of the QPushButton, even if the class or its superclasses would define a slot or property named "data". But keep in mind that the instance `p` of QPushButton only represents a pointer to a C++ QPushButton object, not the object itself; if another Qt method returns a pointer to this object, a new instance is created, pointing to the same QPushButton, but having a new data table. If you therefore need to store data in QPushButton, store it in properties, not in the data table.

In Qt the signals of an object can be connected to the slots of another object. If we e.g. would like the button of the above example to do something when pushed, we can write:

```
Lua> p:connect( "clicked()", function() print( "Pushed" ) end )
Pushed
Pushed
```

Now whenever you click on the button, "Pushed" is printed to the terminal. "clicked()" is a signal of the Qt class QAbstractButton which is the superclass of QPushButton. As you can see in the documentation [8] the signal has the optional argument "checked" we simply left out in the example. We could also connect the signal like this:

```
Lua> p:disconnect("clicked()")
Lua> function onClick( this, checked ) print( "Clicked "..tostring(
         this.text).." = "..tostring(checked) ) end
Lua> p:connect( "clicked(bool)", onClick )
Clicked Push me! = false
```

The first line shows yet another feature of the binding: `disconnect()` is the inverse operation of `connect()`. If you didn't call it, then the function of the previous example would remain connected and executed as well when you clicked the button.

The example shows, that you can also connect to a named function ("onClick" in this case). This time the function has two arguments: "this" and "checked". Whenever the binding makes a callback to a function, the first argument is the instance which caused the callback, in our case the QPushButton `p`; the successive arguments are the ones passed from the signal, in our case "bool checked". Please note that the signal was specified as "clicked(bool)", not as "clicked(bool checked)"; this is a rule set by Qt; you have to leave out the parameter names, otherwise you get an error message. Also note that "clicked()" and "clicked(bool)" are considered to be different signals; if you therefore executed `p:disconnect("clicked()")`, the connections to "clicked(bool)" would not be affected.

Since you connected to a named function, you could also call `p:disconnect("clicked(bool)", onClick )` to disconnect the given function and leave all other connections to "clicked(bool)" intact (the version without a function argument removes all connections to the given signal).

There are still to other versions of connect and disconnect. You can also use these methods to connect a signal of a Qt object to a slot of another Qt object (which is the original intention of Qt). Here is an example in continuation of the one above:

```
Lua> c = QCheckBox("Check me!")
Lua> c:show()
Lua> p:connect( "clicked()", c, "toggle()" )
```

When you now click on the button, the check mark is toggled on and off. If you want to disconnect the slot again from the signal, simply call `p:disconnect( "clicked()", c, "toggle()" )`.

Since an instance only represents a pointer to a Qt object the question arises who will delete the Qt object itself if it is no longer needed. For example the QPushButton in the above example would still exist and remain visible even if you wrote `p = nil`. If you want the Qt object to be deleted when the binding instance goes out of scope, set its owner attribute to true (e.g. `p.owner = true`). Here is the continuation of the above example:

```
Lua> p.owner = true
Lua> p:deleteLater() -- a slot of QObject; here the button vanishes
Lua> p:show()
[string "#Terminal"]:1: dereferencing null pointer!
Lua> print( p:isNull() )
true
```

It is no problem if you set the owner attribute of more than one instance pointing to the same QObject descendant to `true`. The binding is implemented using QPointer and therefore aware of whether the Qt object is still alive or not. For this reason the second call to `p:show()` in the above example generates an error message and not a program crash. To avoid the error message you can use the method `isNull()` which is a built-in feature of the binding.

The CARA Object Model

The *CARA Object Model* (CARM) consists of all entities CARA has to manage toghether with their relations. You as a CARA user can see and manipulate these objects from within the many CARA scope windows. But most of these objects can also be accessed by Lua scripts. In the current release of CARA there are about 80% of the core object model accessible by the CARA/Lua API (enough to do impressive things, see [5]). The rest of the model and an adapter to all CARA scope windows will be available in near future.

The following diagramm (Figure 2) gives an overview of the core classes of CARM. Each class is represented by a box containing its name. The lines between the boxes signify (static) relations between the classes, where a diamond means "ownership" and the numbers give the cardinality of the relation (i.e. a Repository owns zero or more Projects, which themselfs can own zero or more Spins). You can find a description of each object and its methods in the reference part of this manual.

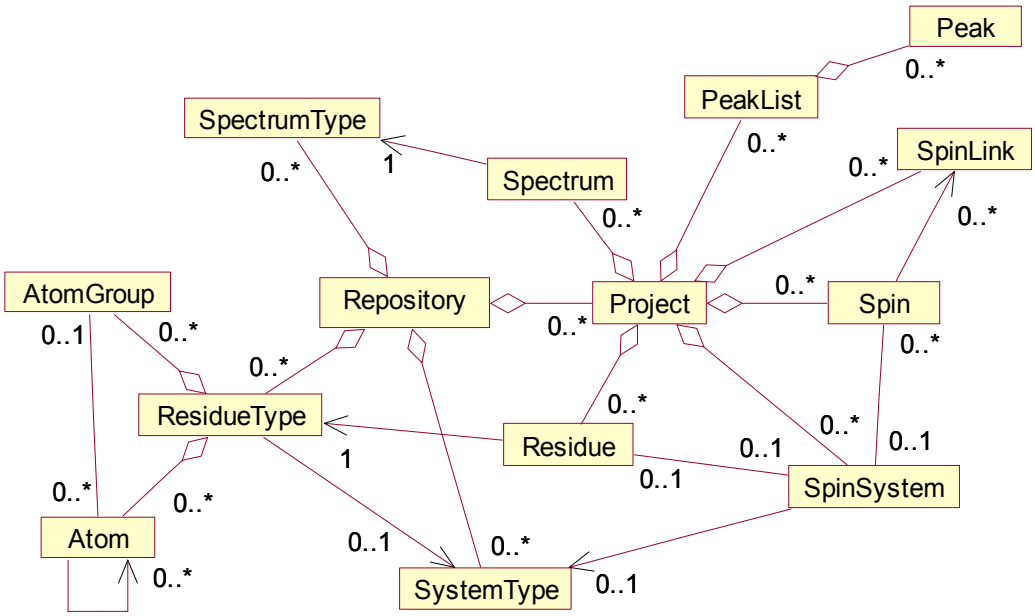


Figure 2: CARM Core Classes

Additionally CARA/Lua offerst a comprehensive library of interactive user interface components, which can be used to build custom editors and tools. The yet available components are generic. NMR specific components from the *NMR Application Framework* (NAF) will be accessible by Lua scripts in future. With the current release, you can already display planes and slices within your windows, but you have to take care of interactivity by yourself.

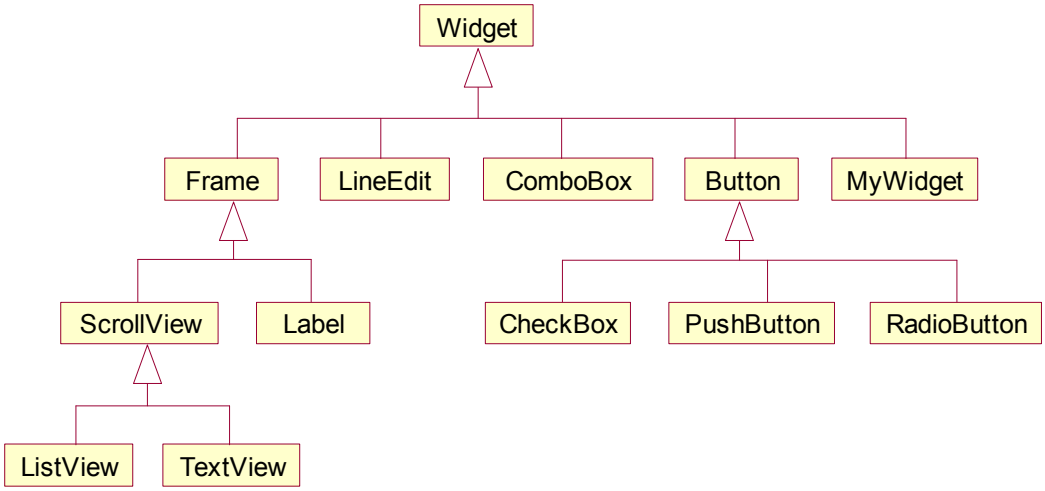


Figure 3: User Interface Controls

Figure 3 gives an overview of the user interface controls. The boxes again represent classes, whereas the lines show a class/sub-class relationship. The sub-classes point to their super-classes, from which they *inherit* all methods.

Some of the classes are abstract and cannot be instantiated (e.g. Widget and Button). Widget is the ancestor of all other controls, organizers and windows. MyWidget is the base for custom controls implemented with Lua scripts (by providing callback functions for the relevant events).

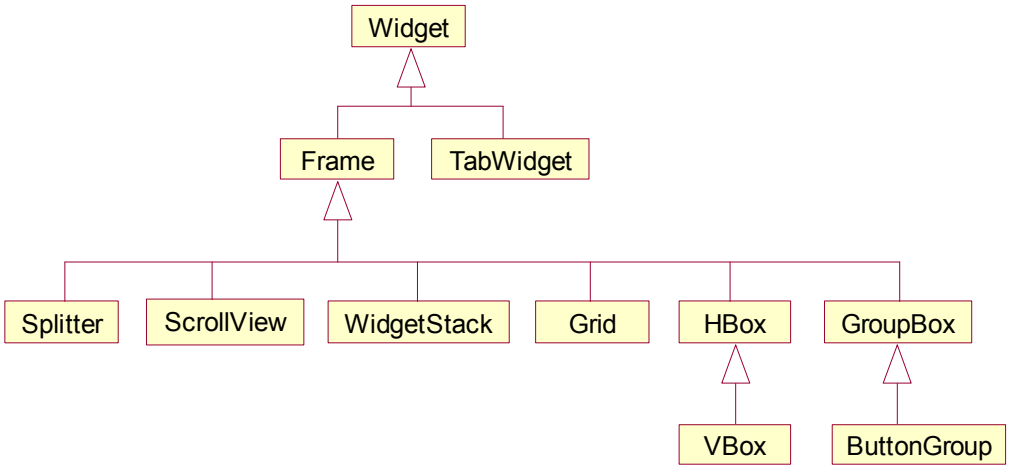


Figure 4: Layout Organizers

Layout organizers are used to organize controls within a window. Grid and Boxes automatically layout their contents according to their preferred sizes and the given space within the window.

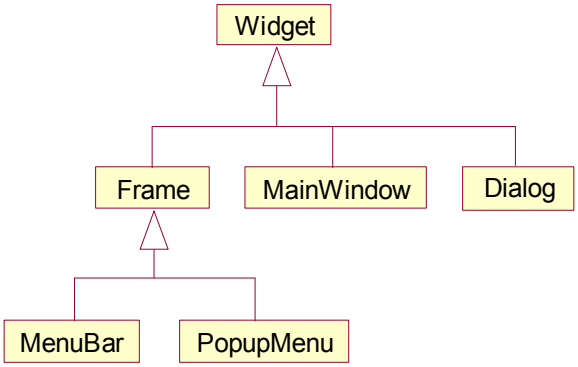


Figure 5: Top-Level Windows and Menues

In contrary to the usual Lua tables, the objects of the CARA/Lua API cannot simply be created by a table constructor (see Lua tutorial on page 16). Instead there are pre-defined objects (e.g. of class Repository or Project) which play the role of an object factory, i.e. they create an object of the requested class when calling a specific factory method (e.g. spec.openSpectrum).

The CARA/Lua API Reference

Global Variables

`cara`

Lua can access the CARA Object Model by means of the global variable named "cara", which is automatically updated to point to the currently open Repository. The following example prints the name and creation date of the current repository.

```
print( cara:getAttr( "Author" ) )
print( cara:getAttr( "Creation Date" ) )
```

gui.explorer

Lua can access the CARA Explorer by menas of the global variable "gui.explorer". The following example adds a PopupMenu to the MenuBar of the Explorer.

```
local test = gui.explorer:getMenuBar():addMenu("Test")
test:addAction("Say Hello", function() print("Hello!") end )
```

gui.selected

When one of the following object types is selected in the CARA Explorer it can be accessed by this variable: Spectrum, Spin, SpinSystem, SpectrumType, ResidueType, SystemType, Project, Residue, BioSample, PeakList and SpinLink.

Class Atom

Inherits Class Object

The Atom class represents the atoms of a molecule. Their main attributes are the atom type (a string containing a mnemonic from the periodic table of elements) and an arbitrary name (i.e. a code or atom label) specifying the role of the atom within the molecule. The atom is recognized within the molecule by means of its name. Additionally each atom can have a random coil shift.

Atoms are owned by a ResidueType, reference their neighgours (this relation represents covalent bonds) and can be part of an AtomGroup.

Atom:addIsotope

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	int > 0		ID of Labeling Scheme
2	string		Isotope code such as "C13" or "N15"

Return values

None.

Atom:getAtomType

Parameters

number (optional, schema ID, default 0)

Returns

Nr.	*Type*	*Opt.*	*Description*
1	string		Atom symbol according to periodic system of elements.

Atom:getGroup

Parameters

None

Returns

Nr.	Type	Opt.	Description
1	AtomGroup or nil		The group this atom belongs to if any, e.g. QB for HB2 or HB3

Atom:getIsotope

Parameters: number (optional, schema ID, default 0)

Returns: string (isotope label)

Atom:getMagnitude

Parameters

None.

Returns

Nr.	Type	Opt.	Description
1	number		The number of real atoms represented by this object, i.e. number 3 for the H in CH3.

Atom:getName

Parameters

None.

Returns

Nr.	Type	Opt.	Description
1	string		The name of the atom, unique within its molecule, z.B. CA

Atom:getNeighbours

Parameters

None.

Returns

Nr.	Type	Opt.	Description
1	Table[string, Atom]		A table with all other atoms connected to this atom over a bond. The key of the table is the name of the atom in the value.

Atom:getValue

Parameters

None.

Returns

Nr.	Type	Opt.	Description
1	number or nil		Mean value, if available, otherwise nil.
2	number or nil		Standard deviation, if value is available

Atom:removeIsotope

Parameters

Nr.	Type	Opt.	Description
1	int > 0		ID of Isotope

Returns
None

Atom:setMagnitude

Parameters

Nr.	Type	Opt.	Description
1	number >= 0		Number of real Atoms represented by this object

Returns
None

Atom:setValue

Parameters

Nr.	Type	Opt.	Description
1	number or nil	x	Mean value
2	number or nil	x	Standard deviation

If the parameters are missing, the value is cleared.

Returns
None

Class AtomGroup

Inherits Class Object

An AtomGroup groups Atoms, between which can normally not be distinguished during the assignment process (i.e. the atoms HB1 and HB2 belong to the atom group QB). Most CARA algorithms accept an AtomGroup as a substitute for its contained Atoms. AtomGroups are recogniced by their name (i.e. QB), which must be unique within a ResidueType.

AtomGroup:getAtoms

Parameters
None.

Returns

Nr.	Type	Opt.	Description
1	Table[string, Atom]		A table with all atoms belonging to this group. The key of the table is the name of the atom in the value.

AtomGroup:getName

Parameters
None.

Returns

Nr.	Type	Opt.	Description
1	string		The name of the atom group, unique within its molecule, z.B. QB

Class BioSample

Inherits Class Object

A BioSample represents the test sample (i.e. bio material) used for a measurement. Spectra can be assigned to the BioSample from which they were measured. The BioSample Class is also used to associate ranges of the Sequence with labeling schemes.

BioSample:getEnd

Parameters: number (start index of sequence range)

Returns: number (correpsonding end index of sequence range)

BioSample:getId

Parameters: none

Returns: number (ID of the BioSample)

BioSample:getName

Parameters: none

Returns: string (name of the BioSample)

BioSample:getRanges

Parameters: none

Returns: Table[number (start of range), number (end of range)]

BioSample:getSchema

Parameters: number (sequence index)

Returns: number (labeling scheme ID corresponding to range enclosing the given index)

Class Buffer

A Buffer is the result of a call to Spectrum:getPlanePpm or getSlicePpm. It contains a vector or matrix of sample values, associated with their corresponding ppm values. The sample values of Buffer can also be changed, which has no influence on the Spectrum it came from. You could e.g. process it with image processing or peak finding algorithms and display the resulting Buffer in a ContourPlot.

Buffer:calculate

This function takes another Buffer, executes the operation and returns a new Buffer. The following operations are supported:

Code	*Name*	*Description*
1	Correlate	Calculates the cross-correlation of this and the other Buffer
2	Convolute	Calculaties the convolution of this and the other Buffer
3	Add	Simple addition
4	Subtract	Simple subtraction
5	Desitar	Calculates the SITAR reconstruction as described in [6]

Parameters

Nr.	Type	Opt.	Description
1	Buffer		Second Buffer
2	int		Operation

Return Value

Buffer

Buffer:clone

This function creates a copy of the given Buffer.

Parameters

None.

Returns

Nr.	Type	Opt.	Description
1	Buffer		Creates a new Buffer as a full copy of this Buffer.

Buffer:getAt

This function returns the amplitude at the given index position. The Buffer can have one or two dimensions (depending on how it was created). When the position is out of valid range, the script is aborted with an error.

Parameters

Nr.	Type	Opt.	Description
1	number		Sample index of dimension D1 of the Buffer.
n	number		Sample index of dimension Dn of the Buffer.

Returns

Nr.	Type	Opt.	Description
1	number		Amplitude at the given sample index.

See Also

Spectrum:getPlanePpm, Spectrum:getSlicePpm, Buffer:getSampleCount

Buffer:getAtomType

Parameters

Nr.	Type	Opt.	Description
1	number		Dimension 1 to 2, depending on getDimCount

Returns

Nr.	Type	Opt.	Description
1	string		Atom symbol according to periodic system of elements.

See Also

Buffer:getDimCount

Buffer:getAtPpm

This function returns the amplitude at the given PPM position. Since the Buffer can have one or two dimensions (depending on how it was created). When the position is out of valid PPM range, zero is returned.

Parameters

Nr.	Type	Opt.	Description
1..n	number		PPM value for each dimension of the Buffer.

Returns

Nr.	Type	Opt.	Description
1	number		Amplitude at the given PPM point.

See Also

Spectrum:getPlanePpm, Spectrum:getSlicePpm

Buffer:getConvolution

Parameters

Nr.	Type	Opt.	Description
1	Buffer		The other Buffer used for calculating the convolution

Return Value

Buffer: the Buffer with the result of the calculation

See Also

Buffer:calculate

Buffer:getCorrelation

Parameters

Nr.	Type	Opt.	Description
1	Buffer		The other Buffer used for calculating the correlation

Return Value

Buffer: the Buffer with the result of the calculation

See Also

Buffer:calculate

Buffer:getDimCount

Returns

Nr.	Type	Opt.	Description
1	number		Number of dimensions of this Buffer: 1 or 2

Buffer:getFolding

Parameters

Nr.	Type	Opt.	Description
1	Dimension		1 or 2; only 1 in case of 1D Buffer

Returns

Nr.	Type	Opt.	Description
1	string		Kind of folding: "None", "RSH", "TPPI"

Buffer:getFreq

Parameters

Nr.	Type	Opt.	Description
1	number		Dimension 1 to 2, depending on getDimCount
2	number		Sample index 1..n

Returns

Nr.	Type	Opt.	Description
1	number		PPM position corresponding to the given sample index

See Also

Buffer:getDimCount

Buffer:getIndex

Parameters

Nr.	Type	Opt.	Description
1	number		Dimension 1 to 2, depending on getDimCount
2	number		PPM position

Returns

Nr.	Type	Opt.	Description
1	number		Sample index corresponding to the given PPM position

See Also

Buffer:getDimCount

Buffer:getIsotope

Parameters

Nr.	Type	Opt.	Description
1	Dimension		1 or 2; only 1 in case of 1D Buffer

Returns

Nr.	Type	Opt.	Description
1	string		Isotope label: "H1", "C13", etc.

Buffer:getPpmRange

Parameters

Nr.	Type	Opt.	Description
1	number		Dimension 1 to 2, depending on getDimCount

Returns

Nr.	Type	Opt.	Description
1	number		PPM value at sample index 1 (middle of sample)
2	number		PPM value at sample index n (middle of sample)

See Also

Buffer:getDimCount, Buffer:getSampleCount

Buffer:getSampleCount

Parameters

Nr.	Type	Opt.	Description
1	number		Dimension 1 to 2, depending on getDimCount

Returns

Nr.	Type	Opt.	Description
1	number		Number of samples in the given dimensions.

See Also

Buffer:getDimCount

Buffer:resample

Use this function to reduce the resolution of the buffer, i.e. to set a new number of samples for the width and height independently. The buffer is "smoothly scaled" using an image processing algorithm (scale reduction).

Parameters

Nr.	Type	Opt.	Description
1	number		Width (sample points)
2	number		Height (sample points)

Returns

None

Buffer:saveToFile

This function saves the Buffer to a CARA or XEASY format spectrum.

Parameters

Nr.	Type	Opt.	Description
1	string		File path
2	string	x	Type: "XEASY" (default) or one of the following: "Comp8", "Uncomp8", "Uncomp16", "Uncomp32", "Comp8Gauss" or "Comp8Adapt"; see [3] section 7.5 for an explanation.
3	double	x	maximum amplitude to be used for CARA format
4	double	x	minimum amplitude to be used for CARA format

Returns

None

Buffer:setAt

This function sets the amplitude at the given index position. The Buffer can have one or two dimensions (depending on how it was created). When the position is out of valid range, the script is aborted with an error.

Parameters

Nr.	Type	Opt.	Description
1	number		Sample index of dimension D1
n	number		Sample index of dimension Dn
n+1	number		Amplitude to be set at the sample point

Returns

None

See Also
Spectrum:getPlanePpm, Spectrum:getSlicePpm, Buffer:getSampleCount

Class Button

Inherits Class Widget

This is a sub-class of Widget and an abstract super-class of PushButton, CheckBox and RadioButton. A button can call a Lua function, when the user presses it.

Button:getText

Parameters: none

Returns: string

Button:isOn (also isChecked)

Parameters: none

Returns: boolean

Button:isToggleButton

Parameters: none

Returns: boolean

Button:setAccel

Parameters: string (acceleration code, e.g. "Ctrl+S")

Returns: none

Button:setCallback

See Widget:setCallback(). The following events and callback functions are supported:

gui.event.Clicked	function(self)	Parameters: Button Returns: none
gui.event.Toggled	function(self, checked)	Parameters: Button, boolean Returns: none

Button:setChecked

Parameters: boolean (optional, default true)

Returns: none

Button:setIcon

Parameters: Icon

Returns: none

Button:setText

Parameters: string

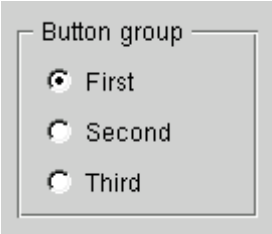
Returns: none

The text of a Button is its label, on top or next to it, depending on the button type. The text can contain shortcuts, e.g. "&Press me". The "P" is underlined as a shortcut. The user can trigger the shortcut pressing ALT+P.

Class ButtonGroup

Inherits Class GroupBox

A ButtonGroup inherits all methods of GroupBox. It is used to group RadioButtons and to assure, that only one button is checked.



ButtonGroup:addButton

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Button		
2	number	x	Member ID

Returns

number (member ID)

ButtonGroup:setCallback

See Widget:setCallback(). The following events and callback functions are supported:

gui.event.Clicked	function(self, memberID)	Parameters: ButtonGroup, number Returns: none

ButtonGroup:setExclusive

Parameters: boolean (optional, default true)

Returns: none

Class Canvas

Inherits QMainWindow

A Canvas is an independent window with automatic scrollbars, in which you can draw using Lua scripts. It is created by a call to dlg.createCanvas and accessible by Lua as long as the user doesn't close ist. There is no way to programmatically close a Canvas. All the methods of this class abort the script with an error, if the user has closed the canvas.

All coordinates are in points (1/72 inch per point). The coordinate system has its origin in the upper left corner of the canvas. On screen a point usually corresponds to one pixel.

The user can then save and reload the drawings in pix-files or print them to any printer (e.g. PostScript). It is also possible to write the PostScript code to a file (e.g. to use it in Adobe Illustrator).

The following example draws an array (`data`) as a curve:

```
local data = { 3, 4, 8, 9, 11, 4, 13, 5, 2, 1 }
local width = 500
local height = 400
canvas = dlg.createCanvas()
canvas:begin()
local x = 0
canvas:setPen( "darkGray", 1 )
canvas:drawLine( 0, 0, width, 0 ) -- draw base line
local n = table.getn( data )
canvas:setPen( "black", 1 )
canvas:moveTo( 0, 0 )
for i= 1, n do
     x = ( i - 1 ) * width / ( n - 1 )
     canvas:lineTo( x, data[ i ] * 10  ) -- draw curve
end
canvas:commit()
```

Canvas:begin

Parameters: none

Returns: none

Starts a new painting transaction, overwriting the former drawing on the canvas. It is an error to call this function more than once without calling commit().

Canvas:commit

Parameters: none

Returns: none

Ends a painting transaction started by begin(). When committing the drawing becomes visible. It is an error to call this method without a previous call to begin().

Canvas:drawContour

Draws the given ContourPlot at the given point position. The plot scales into the area given by width and height.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	ContourPlot		
2	number		X position
3	number		Y position
4	number		Width
5	number		Height

Returns

None

Canvas:drawEllipse

Draws an ellipse with the current pen and brush at center point x/y with extension width and height.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position
3	number		Width
4	number		Height

Returns

None

Canvas:drawIcon

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Icon		
2	number		X position
3	number		Y position

Returns

None

Canvas:drawImage

Draws the given image at the given point position. If width and height are left out, the image is painted with its original dimension. You can stretch the image to width and height if needed.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Image		
2	number		X position
3	number		Y position
4	number	x	Width
5	number	x	Height

Returns

None

Canvas:drawLine

Draws a line from point x1/y1 to x2/y2 using the current pen.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X1 position
2	number		Y1 position
3	number		X2 position
4	number		Y2 position

Returns

None

Canvas:drawPoint

Draws a single point at x/y using the current pen.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position

Returns

None

Canvas:drawRect

Draws a rectangle with the current pen and brush, whose left upper edge is at point x/y, extending width to the right and height to the bottom.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position
3	number		Width
4	number		Height

Returns

None

Canvas:drawSlice

Draws the given SlicePlot at the given point position. The plot scales into the area given by width and height.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	SlicePlot		
2	number		X position
3	number		Y position
4	number		Width
5	number		Height

Returns

None

Canvas:drawText

Draws the text string with the current pen and font at point x/y, where y is at the base line and x at the left edge of the first glyph.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position
3	string		Text to draw

Returns

None

Canvas:fillRect

Draws a rectangle with the given brush, whose left upper edge is at point x/y, extending width to the right and height to the bottom. Only fills the area without drawing a border. The brush is defined by the text string (format "#RRGGBB" or color-names).

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position
3	number		Width
4	number		Height
5	string		Fill color name

Returns

None

Canvas:getBounding

This method accepts a text string and returns the width and height it will have when drawn on the screen using the current font.

Parameters

string

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		Width
2	number		Height

Canvas:lineTo

This function draws a line with the current pen, starting from the last position set by a former lineTo or moveTo.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position

Returns

None

Canvas:moveTo

This function sets the current positon to x/y without drawing anything. This is useful when calling lineTo afterwards.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position

Returns

None

Canvas:setBgColor

Parameters: string

Returns: none

This function sets the background color of the Canvas to the color specified with a text string (format "#RRGGBB" with R, G and B being hexadecimal numbers, or alternatively by naming an X-Window color like "white", "green", etc.).

Canvas:setBrush

Parameters: string (optional)

Returns: none

This function sets the current brush color. All following drawing operations depending on brush are immediately affected. The color is coded as usual. Omitting the parameter sets the brush to transparent.

Canvas:setCaption

Parameters: string

Returns: none

This function sets the caption of the Canvas window to string.

Canvas:setFont

This function sets the current font, which affects the following calls to drawText.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		Font name
2	number		Point size
3	boolean		Bold
4	boolean		Italic

Returns

None

Canvas:setPen

This function sets the current pen color and width. All following drawing operations depending on pen are immediately affected. The color is coded as usual.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		Color name
2	number		Width

Returns

None

Canvas:setSize

This function sets the minimum drawing size of the canvas. When set to a larger value than the window size, scroll bars are displayed on the canvas window.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Width
2	number		Height

Returns

None

Canvas:setWindowSize

This function sets the size of the canvas window. The drawing surface can be bigger than the window in which case scrollbars appear.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Width
2	number		Height

Returns

None

Canvas:setZoomFactor

Parameters: number

Returns: none

This method sets a factor by which the x and y coordinates are stretched when drawing.

Class CheckBox

Implemented by QCheckBox

CheckBox conceptually inherits all features of Class Button.

isChecked

Parameters: none

Returns: boolean

setChecked

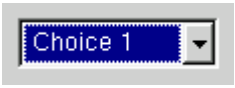
Parameters: boolean (default true)

Returns: none

Class ComboBox

Implemented by QComboBox

ComboBox inherits all features of Widget. Use it to present the user a list of items to choose from



ComboBox:addItem

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		Label text
2	Icon	x	

Returns

number (index of new item)

ComboBox:clear

Parameters: none

Returns: none

ComboBox:getCurrentItem

Parameters: none

Returns: number (index of selected item, starting with 1)

ComboBox:getCurrentText

Parameters: none

Returns: string (text of selected item)

ComboBox:setCallback

See Widget:setCallback(). The following events and callback functions are supported:

gui.event.Changed	function(self, text)	Parameters: ComboBox, string Returns: none
gui.event.Activated	function(self, index)	Parameters: ComboBox, number Returns: none

ComboBox:setCurrentItem

Parameters: number (index to select, starting with 1)

Returns: none

ComboBox:setEditable

Parameters: boolean (optional, default true)

Returns: none

If set to true, you can either choose an item from the popup or write your own text into the edit part of the control.

ComboBox:setEditText

Parameters: string

Returns: none

Use this function to preset the text of the edit part of the control.

Class Conformer

Inherits Class Object

A Conformer is a calculated version of a Structure. Usually many versions of a structure are calculated and then consolidated into a single Structure.

Conformer:getCoord

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		ID of spatial position (usually corresponds to the Spin ID)

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		Coordinate in dimension x
2	number		Coordinate in dimension y
3	number		Coordinate in dimension z

Conformer:getCoords

Parameters: none

Returns: Table[number (array index), number (ID of coordinate)]

Conformer:getId

Parameters: none

Returns: number (the ID of the Conformer)

Conformer:getNr

Parameters: none

Return: number

This is an arbitrary number which can be assigned to the Conformer (in contrast the ID number is automatically generated and immutable).

Conformer:setCoord

Parameters

Nr.	Type	Opt.	Description
1	number		ID of coordinate (usually corresponds to the Spin ID)
2	number		Coordinate in dimension x
3	number		Coordinate in dimension y
4	number		Coordinate in dimension z

Returns

None

Class ContourPlot

A ContourPlot is used to draw contour lines according to the given Buffer to a Canvas or Painter. It is created by dlg.createContour. It is - like an Image - an autonomous object which can be drawn on a Canvas or Painter with the drawContour method. The object can be reused for different drawings. Changes made to ContourPlot do not affect already commited drawings (see Canvas:commit).

```
spec = cara:getProject():getSpectrum( 3 )
buf = spec:getPlanePpm( 1, 2, 10, 7, 128, 116 )
buf:resample( 50, 50 )
c = dlg.createContour( buf )
c:setParams( 1.5, 300, "+" )
c:setPosColor( "black" )
cv = dlg.createCanvas()
cv:begin()
cv:drawContour( c, 30, 40, 200, 200 )
cv:setPen( "blue", 1 )
cv:drawRect( 30, 40, 200, 200 )
cv:commit()
```

ContourPlot:setBuffer

Parameters: Buffer

Returns: none

This function sets the sample buffer to be contoured. The buffer must be two dimensional.

ContourPlot:setLineWidth

Parameters: number (line width in points)

Returns: none

Set the line width of the pen used to draw the contour lines.

ContourPlot:setNegColor

Parameters: string (color code)

Returns: none

Set the color used to draw the contour lines of negative amplitudes. The code has the usual format ("#RRGGBB", etc.).

ContourPlot:setParams

Sets the contour parameters for this object. To see the effect, the plot has to be redrawn e.g. with Canvas:drawContour. Leave out the third parameter, if you want to draw positive as well as negative amplitudes.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Contour factor
2	number		Contour threshold
3	string	x	"+"..only positive, "-"..only negative

Returns

None

ContourPlot:setPosColor

Parameters: string (color code)

Returns: none

Set the color used to draw the contour lines of positive amplitudes. The code has the usual format ("#RRGGBB", etc.).

ContourPlot:toPoint

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Dimension 1 or 2
2	number		Origin (in points)
3	number		Width (in points)
4	number		PPM position

Returns

number (point position of given ppm value)

ContourPlot:toPpm

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Dimension 1 or 2
2	number		Origin (in points)
3	number		Width (in points)
4	number		Position (in points)

Returns

number (ppm value of given point position)

Class Dialog

Implemented by QDialog

This class inherits all the features of Widget. A Dialog is a modal top-level-window, which you can use to ask the user for input. While the dialog is displayed, the rest of the application blocks.

Dialog:accept

Parameters: none

Returns: none

Use this method to close the modal dialog, returning Dialog:exec() with a true value. The funcion has no effect when the dialog is not shown.

Dialog:exec

Parameters: none

Returns: boolean

This method shows the modal dialog on screen. As long as the dialog is shown, CARA blocks any other interactivity (that's why it's called "modal"). You can place buttons on the Dialog which call other methods when the user presses them. Within these methods you can call Dialog:accept() (in which case exec returns true) or Dialog:reject() (in which case exec returns false). The function also retuns false if the user closes the dialog by other means (e.g. the "x" button in the title bar on windows).

Dialog:reject

Parameters: none

Returns: none

Use this method to close the modal dialog, returning exec with a false value. The funcion has no effect when the dialog is not shown.

Dialog:setCentralWidget

Parameters: QWidget

Returns: none

This method accepts a Widget and uses it so it covers the whole visible area of the Dialog.

Class DomDocument

This class represents XML files loaded into memory. It is created by xml.createDocument or xml.openDocument. Use it to load, process and write XML data. DomDocument, DomElement and DomText partly follow the Document Object Model (DOM) standardized by the W3C (www.w3c.org). The following diagramm shows the objects with their relations to each other.

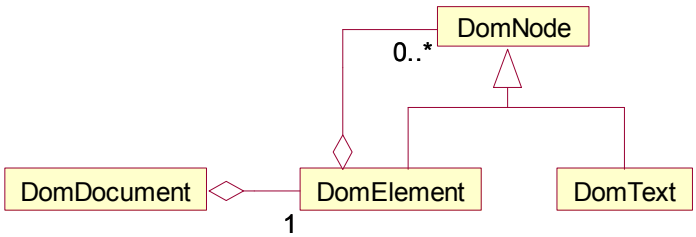


Figure 6: DOM Object Relations

DomNode is a virtual object which is only necessary to show the polymorphic nature of DomElement and DomText (DomElement is a DomNode and DomText is a DomNode). The DomElement is owner of zero or more subordinate DomText and DomElement, represented by the relation to DomNode.

```
doc = xml.openDocument( dlg.getOpenFileName() )
print( doc:getXml() )
root = doc:getDocumentElement()
e = root:getFirstChild()
while e ~= nil do
     print( e:getName() )
     e = e:getNextSibling()
end
```

DomDoc:getDocumentElement

Parameters: none

Returns: DomElement

Each DomDocument has exactly one root element which you have to access using this function. When a new DomDocument is created, this element is automatically created.

DomDoc:getXml

Parameters: none

Returns: string

This function returns a string containing the whole XML text of this DomDocument. Use it for building your own saving function or for debugging purpose.

DomDoc:saveToFile

Parameters: string (file path)

Returns: none

This function tries to save the DomDocument as an XML file to the given path. The function is aborted with an error, if the file cannot be opened for writing. Note that an existing file with the same path will be overwritten.

Class DomElement

A DomDocument consists of a tree of DomElements. DomElements are recursively contained by itself and thus span a tree structure.

According to the W3C standard DomElements can contain CDATA sections, which are supposed to be separate element types in principle. In CARA/Lua the CDATA sections are automatically converted to text elements and do not appear as separate classes.

DomElements consist of attributes and child elements, the latter beeing DomElements and DomTexts polymorphically. The attribute names are unique within a DomElement, where'as the child DomElements can have an arbitrary name (the order of the child elements is relevant).

DomElem:createElement

Parameters: string (tag name)

Returns: DomElement

This function creates a new element with the given name and appends it to the end of the list of children of the element this function is called with.

DomElem:createText

Parameters: string (optional, the text)

Returns: DomText

This function creates a new text and appends it to the end of the list of children of the element this function is called with. If the parameter is omitted, an empty text element is created. You can combine all successing text elements to one text using the function `normalize`.

DomElem:getAttribute

Parameters: string (name)

Returns: string or nil

Use this function to access an attribute of this element.

DomElem:getAttributes

Parameters: none

Returns: Table[string (name), string (value)]

This function returns the list of all attributes of this element.

DomElem:getChildren

Parameters: none

Returns: Table[number (array index), DomElement or DomText]

This function returns all the children of this element in an array (in the original order). Note that an element can have either texts or other elements as children.

DomElem:getFirstChild

Parameters: none

Returns: DomElement or DomText or nil

This function returns the first child of this element (equivalent to `DomElement:getChildren()[1]`).

DomElem:getLastChild

Parameters: none

Returns: DomElement or DomText or nil

This function returns the last child of this element (equivalent to `DomElement:getChildren()[n]`, where n is the number of children).

DomElem:getName

Parameters: none

Returns: string (the tag name of this element)

DomElem:getNextSibling

Parameters: none

Returns: DomElement or DomText or nil

This function returns the next element in the list of children of the parent element. Use it to iterate over all children of the parent.

DomElem:getPrevSibling

Parameters: none

Returns: DomElement or DomText or nil

This function returns the previous element in the list of children of the parent element. Use it to iterate over all children of the parent.

DomElem:hasAttribute

Parameters: string (attribute name)

Returns: boolean

This function returns true, if an attribute with the given name is defined in this element.

DomElem:isElement

Parameters: none

Returns: boolean

This function returns always true for DomElements. Use it to differ between DomElement and DomText members, because the list of children of an element is polymorphic.

DomElem:isText

Parameters: none

Returns: boolean

This function returns always false for DomElements. Use it to differ between DomElement and DomText members, because the list of children of an element is polymorphic.

DomElem:normalize

Parameters: none

Returns: none

Use this function to combine all succesive DomText elements of the children list of this element into one DomText.

DomElem:removeAttribute

Parameters: string (attribute name)

Returns: none

Use this function to remove the attribute with the given name from this element. Nothing happens if the attribute didn't exist.

DomElem:removeThis

Parameters: none

Returns: none

Use this function to remove this element from its parents list of children. The element still exists after removal, but is useless. If you call this function for the root element, the DomDocument becomes useless.

DomElem:setAttribute

This function sets the attribute of the given name to the given value. Only text values can be set (Lua tries to automatically convert other value types to text). If the attribute didn't exist, it is automatically created.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		Attribute name
2	string		Attribute value

Returns

None

DomElem:setName

Parameters: string (name)

Returns: none

Sets the tag name of this element. This is the name visible in the XML file, i.e. as `<name/>`.

Class DomText

DomText represents a text between to XML tags (see example). A DomElement (see above) can contain more than one DomText. The second example shows element2 containing

```
<element1>This is a text between two tags</element1>

<element2>This is part A<subelement/>This is part B</element2>
```

two DomText and a DomElement. The following diagram shows the corresponding instance relationship.

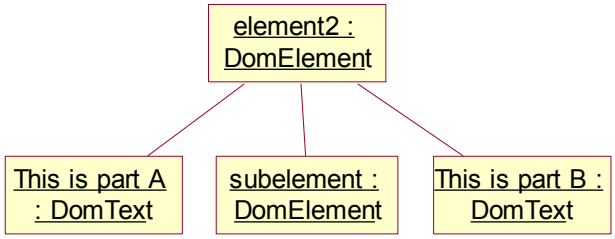


Figure 7: DOM Text Decomposition

If the two DomText were immediately following each other (without the intermediate DomElement), a call to `element2:normalize()` would combine them to a single DomText with contents "This is part A This is part B".

DomText:getNextSibling

Parameters: none

Returns: DomElement or DomText or nil

This function returns the next element in the list of children of the parent element. Use it to iterate over all children of the parent.

DomText:getPrevSibling

Parameters: none

Returns: DomElement or DomText or nil

This function returns the previous element in the list of children of the parent element. Use it to iterate over all children of the parent.

DomText:getText

Parameters: none

Returns: string (the text)

DomText:isElement

Parameters: none

Returns: boolean

This function returns always false for DomText. Use it to differ between DomElement and DomText members, because the list of children of an element is polymorphic.

DomText:isText

Parameters: none

Returns: boolean

This function returns always true for DomText. Use it to differ between DomElement and DomText members, because the list of children of an element is polymorphic.

DomText:removeThis

Parameters: none

Returns: none

Use this function to remove this text element from its parents list of children. The element still exists after removal, but is useless.

DomText:setText

Parameters: string

Returns: none

Class Experiment

Inherits Class Object

This class can be used to calculate all possible magnetization path ways through a given molecule (i.e. ResidueType) in a given NMR experiment (i.e. SpectrumType). All paths are kept in a table of arbitrary order.

```
nmr = spec.createExperiment(
     cara:getSpectrumType( "HNCA" ),
     cara:getResidueType( "ARG" ) )
print( nmr:toString() )
t = nmr:getPath( 3 )
for i, j in pairs( t ) do print( j ) end
```

Experiment:getCount

Parameters: none

Returns: number (number of rows in the path table)

Experiment:getPath

Parameters: number (row index)

Returns: Table[number (index), string (atom label)]

Experiment:getResidueType

Parameters: none

Returns: ResidueType or nil

Experiment:getSpectrumType

Parameters: none

Returns: SpectrumType or nil

Experiment:setResidueType

Parameters: ResidueType

Returns: none

Experiment:setSpectrumType

Parameters: SpectrumType

Returns: none

Experiment:toString

Parameters: none

Returns: string (a pretty printed text of the path table)

Generates a human-readable text representation of the calculated paths.

Class Explorer

Inherits QSplitter

This class is a Lua binding to the CARA Explorer. It can be used to extend the MenuBar and PopupMenues of the Explorer.

Explorer:getMenuBar

Parameters: none

Returns: MenuBar

Explorer:getNaviPopup

Parameters: none

Returns: PopupMenu

Get access to the context menu of the tree on the left side of the CARA Explorer. This is a shortcut for getPopup("navi").

Explorer:getPopup

Parameters: string or nil (section name)

Returns: PopupMenu

Get access to the context menu of the specific section. The PopupMenus of the following sections are accessible: "Spectrum Types", "Scripts", "Attribute Definitions", "Residue Types", "Spin System Types", "Labeling Schemes", "Sequence", "Samples", "Spectra", "Peak Lists", "Spins", "Systems" and "Spin Links", as well as "navi" or "navigator".

See gui.selected for a way to access the currently selected object of the section.

Here is an example which adds a new commad to the context menu of the Spectra section:

```
local m = gui.explorer:getPopup("Spectra")
m:addAction("Test", function() print("Test") end )
```

Class Frame

Inherits QFrame

This class conceptually inherits all the features of Class Widget. Frame is the super-class of many other classes, but can also be used by itself (e.g. to frame certain areas on a form or to separate areas by lines). The following diagram gives an overview of the different frame and shadow styles.

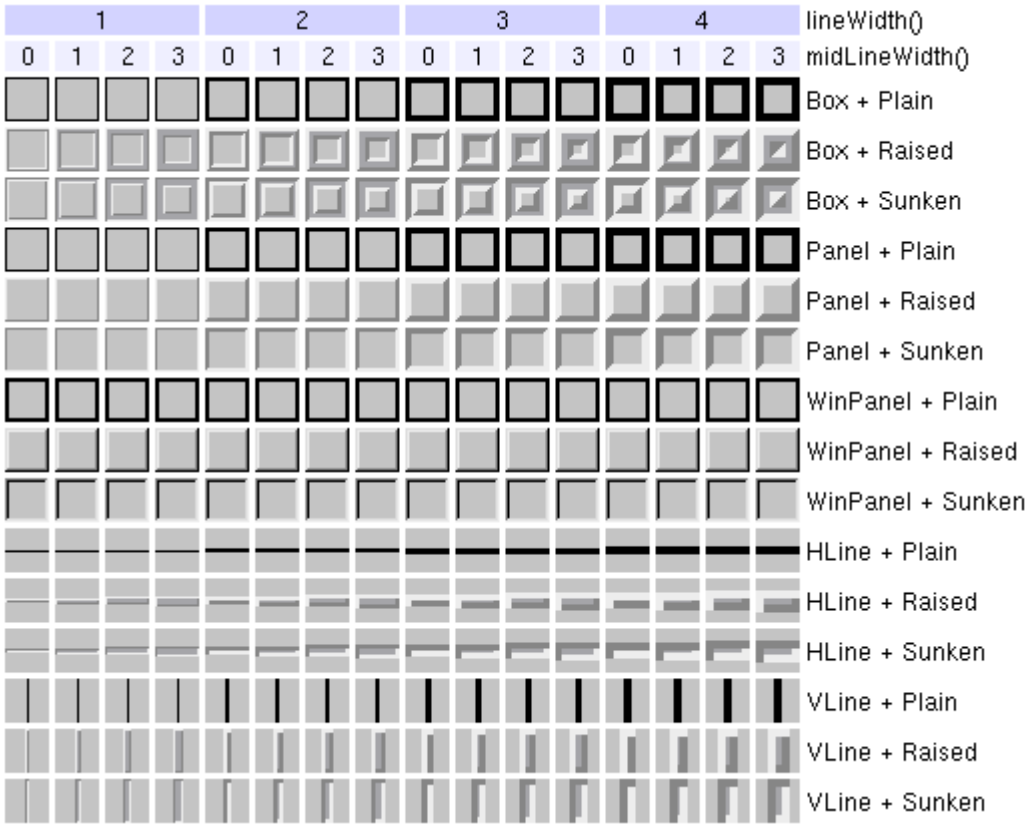


Figure 8: Frame & Shadow Styles

`Frame:getContentsRect`

This is the rectangle within the frame where the contens resides.

Parameters

None

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position
3	number		Width
4	number		Height

`Frame:getLineWidth`

Parameters: none

Retunrs: number (points)

Frame:getMargin

Parameters: none

Returns: number (points)

Frame:getMidLineWidth

Parameters: none

Returns: number (points)

Frame:setFrameStyle

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Frame style code. The following codes are available: Frame.style.NoFrame, Frame.style.Box, Frame.style.Panel, Frame.style.StyledPanel, Frame.style.PopupPanel, Frame.style.WinPanel, Frame.style.HLine and Frame.style.VLine (see Figure 8: Frame & Shadow Styles).
2	number		Optional, shadow style code, default Plain. The following codes are available: Frame.style.Plane, Frame.style.Raised and Frame.style.Sunken (see Figure 8: Frame & Shadow Styles).

Returns

none

Frame:setLineWidth

Parameters: number (points)

Returns: none

Frame:setMargin

Parameters: number (points)

Returns: none

Frame:setMidLineWidth

Parameters: number (points)

Returns: none

Class Guess

Inherits Class Object

A Guess is used to document which Spins are possibly represented by each dimension of a Peak. Each Guess has an ID and also a probability.

Guess:getAssig

Parameter

None.

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		ID of tentative Spin in dimension D1
n	number		ID of tentative Spin in dimension Dn

Guess:getId

Parameter

None.

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		ID of Guess

Guess:getProb

Parameter

None.

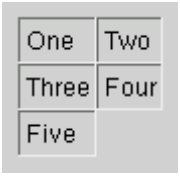
Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		Probability of the Guess

Class Grid

Inherits Class Frame

Grid inherits all features of Frame. Use it to organize the widgets it contains in a tabular fashion.



Grid:setSpacing

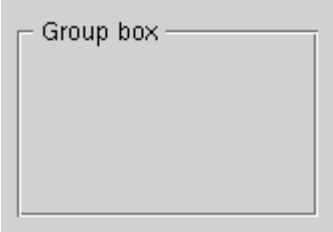
Parameters: number (points)

Returns: none

Class GroupBox

Inherits Class Frame

GroupBox inherits all the features of Frame. It is similar to Grid, but draws a frame border with a title around the widgets it contains. This is useful to separate different named areas on a form.



GroupBox:addSpace

Parameters: number (points)

Returns: none

GroupBox:setAlignment

Parameters: number (alignment code)

Returns: none

Sets the alignment of the title. Use the values gui.align.Left, gui.align.HCenter or gui.align.Right.

GroupBox:setColumns

Parameters: number (column count)

Returns: none

Use it to set the number of columns. The box is set to one column after creation. Avoid calling this function when the box contains child widgets.

GroupBox:setOrientation

Parameters: number (gui.Horizontal or gui.Vertical)

Returns: none

GroupBox:setTitle

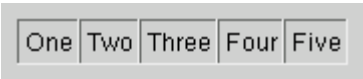
Parameters: string (new title)

Returns: none

Class HBox

Inherits Class Frame

Use this class to horizontally align its child Widgets.



Hbox:setSpacing

Parameters: number (points)

Returns: none

Class Icon

Inherits QPixmap

An Icon is a tiny image and can be used to enhance other widgets (e.g. PopupMenu, ListView, etc.). Icons can be loaded from a file or created from a XPM string, which makes it straight forward to incorporate graphical information into scripts. See also dlg.createIcon and dlg.loadIcon.

Icon:getSize

Parameters

None

Returns

Nr.	Type	Opt.	Description
1	number		Width (points)
2	number		Height (points)

Class Image

Inherits QImage

Objects of this class are created by a call to dlg.loadImage. It is used to hold a bitmap image in memory, which was loaded from a file when creating the object. An Image can be drawn to a Canvas using the method drawImage.

```
img = dlg.loadImage( dlg.getOpenFileName( "Open Image", "" ) )
print( "Size: "..img:getSize() )
cv = dlg.createCanvas()
cv:begin()
cv:drawImage( img, 30, 40, 20, 20 )
cv:commit()
```

Image:getSize

Parameters

None

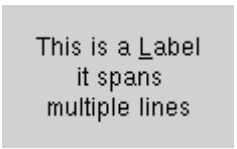
Returns

Nr.	Type	Opt.	Description
1	number		Width (points)
2	number		Height (points)

Class Label

Inherits Class Frame

Label inherits all features of Frame. Use labels to describe the purpose of other widgets (e.g. LineEdits).



Label:setAlignment

Parameters

Nr.	Type	Opt.	Description
1	number		Horizontal alignment; one of gui.align.Left, gui.align.Right and gui.align.HCenter
2	number		Vertical alignment; one of gui.align.Top, gui.align.Bottom and gui.align.VCenter
3	boolean	x	Expand tabs, default false
4	boolean	x	Word break, default false

Returns

None

Label:setBuddy

Parameters: Widget

Returns: none

Use this function to associate a label with another widget. I you activate a shortcut of the label text, the focus goes to the buddy widget.

Label:setIndent

Parameters: number (points)

Returns: none

Label:setText

Parameters: string (new text)

Returns: none

The text can contain shortcuts, e.g. "This is a &Label". The "L" is underlined as a shortcut. The user can trigger the shortcut pressing ALT+L.

Class LineEdit

Implemented by QLineEdit

LineEdit inherits all features of Widget. Use it to enter small texts into forms.

LineEdit:getText

Parameters: none

Returns: string

LineEdit:isEdited

Parameters: none

Returns: boolean

This flag becomes true when the text is changed either by the user or by setText. See also setEdited().

LineEdit:setCallback

See Widget:setCallback(). The following events and callback functions are supported:

gui.event.Changed	function(self, text)	Parameters: LineEdit, string Returns: none
gui.event.Return	function(self)	Parameters: LineEdit Returns: none

LineEdit:setEdited

Parameters: boolean

Returns: none

Sets the edited flag of the widget.

LineEdit:setReadOnly

Parameters: boolean

Returns: none

LineEdit:setText

Parameters: string

Returns: none

Class ListItem

Inherits QTreeWidgetItem

A ListItem is an item of a ListView. Technically it is a QTreeWidgetItem with additional features. It is created by ListView:createItem() or ListItem:createItem().

ListItem:createItem

Parameters: none

Returns: ListItem

Create a new ListItem as a child of the item this function was called. The new item is appended to the end of the list of child items, but this order can arbitrarily change by sorting.

ListItem:destroy

Parameters: none

Returns: none

Remove this item from the list. The item vanishes immediately from screen. The Lua reference becomes useless.

ListItem:getFirstChild

Parameters: none

Returns: ListItem or nil

Use this function to iterate over the child items.

ListItem:getListView

Parameters: none

Returns: ListView (the ListView this item belongs to)

ListItem:getNextSibling

Parameters: none

Returns: ListItem or nil

Use this function to iterate over all items of a hierarchy level.

ListItem:getParent

Parameters: none

Returns: ListItem or nil

Get the parent ListItem or nil, if it is a top-level item. See ListItem:getListView to access the Widget instead of the item.

ListItem:isOpen

Parameters: none

Returns: boolean

Returns true, if item is in open state, i.e. the subitems are visible.

ListItem:isSelected

Parameters: none

Returns: boolean

ListItem:setIcon

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Column index, starting with 1
2	Icon		New icon to be displayed

Returns

None

ListItem:setOpen

Parameters: boolean (optional, default true)

Returns: none

ListItem:setSelected

Parameters: boolean (optional, default true)

Returns: none

ListItem:setText

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Column index, starting with 1
2	string		New text

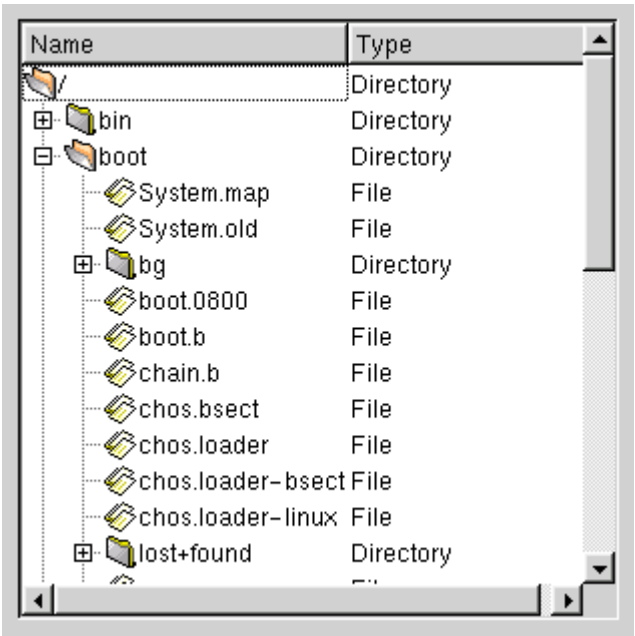
Returns

None

Class ListView

Inherits QTreeWidget

This class technically inherits QTreeWidget, but conceptually inherits Class ScrollView. You can use ListView to present tables, lists or trees to the user.



ListView:addColumn

Parameters: string (column title)

Returns: number (index of new column, starting with 1)

ListView:clear

Parameters: none

Returns: none

Remove all items from the list.

ListView:clearSelection

Parameters: none

Returns: none

Deselect all items.

ListView:createItem

Parameters: none

Returns: ListItem

Create a new top-level item and append it to the end of the list. The order of the items can change arbitrarily because of sorting.

ListView:ensureVisible

Parameters: ListItem

Returns: none

Scroll the list so the given item becomes visible.

ListView:getColumnCount

Parameters: none

Returns: number

ListView:getFirstChild

Parameters: none

Returns: ListItem or nil

ListView:getSelected

Parameters: none

Returns: ListItem or nil

This function returns the currently selected item. Works only in single-selection-mode.

ListView:removeColumn

Parameters: number (column index, starting with 1)

Returns: none

ListView:selectAll

Parameters: boolean (default true)

Returns: none

Use this function to either select (true) or unselect (false) all items of the list.

ListView:setAllColsMarked

Parameters: boolean (default true)

Returns: none

If true, the selection mark covers all columns. If false, only the first column is covered.

ListView:setCallback

See Widget:setCallback(). The following events and callback functions are supported:

gui.event.RightPressed	function(self, item, x, y, column)	Parameters: ListView, ListItem or nil, number, number, number Returns: none
gui.event.Selection	function(self)	Parameters: ListView Returns: none
gui.event.Clicked	function(self, item, x, y, column)	Parameters: ListView, ListItem or nil, number, number, number Returns: none
gui.event.DblClicked	function(self, item)	Parameters: ListView, ListItem or nil Returns: none
gui.event.Return	function(self, item)	Parameters: ListView, ListItem or nil Returns: none
gui.event.Expanded	function(self, item)	Parameters: ListView, ListItem or nil Returns: none
gui.event.Collapsed	function(self, item)	Parameters: ListView, ListItem or nil Returns: none

ListView:setColumnTitle

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Column index, starting with 1
2	string		New title

Returns

None

ListView:setItemMargin

Parameters: number (points)

Returns: none

ListView:setMultiSelection

Parameters: boolean (default true)

Returns: none

If true, the user is allowed to select more than one item in the list.

ListView:setRootDecorated

Parameters: boolean (default true)

Returns: none

If true, the open/closed handles are also shown for the top-level items.

ListView:setSorting

Explicitly sort the given column.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Column index, starting with 1
2	boolean	x	Ascending, default true

Returns

None

ListView:setSortIndicated

Parameters: boolean (optional, default true)

Returns: none

If true, a little triangle next to the column title shows the sort direction.

ListView:setStepSize

Parameters: number (points)

Returns: none

ListView:sort

Parameters: none

Returns: none

Re-sort the whole list.

Class LuaEdit

Inherits QPlainTextEdit

This class represents an editor featuring syntax coloring for Lua.

LuaEdit:getText

Parameters: none

Returns: string (the text of the editor)

LuaEdit:setText

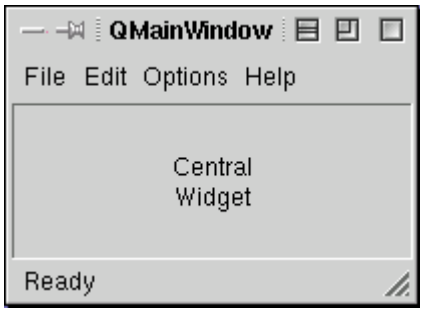
Parameters: string

Returns: none

Class MainWindow

Inherits QMainWindow

This class is a direct descendant of QMainWindow and also inherits all features of Widget. MainWindow is a top-level window featuring an optional menu and status bar. It may contain many sub-widgets, but exactly one central widget. If you don't want a menu or status bar, you can also use some other widgets (all which allow parent to be nil) as top-level windows.



MainWindow:getMenuBar

Parameters: none

Returns: MenuBar

MainWindow:setCentralWidget

Parameters: Widget

Returns: none

MainWindow:showStatusText

Displays the given text on the status line. When the optional duration is given, the text automatically disappears after the timeout.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The text to show
2	number	x	Duration in milli seconds

Returns

None

Class MenuBar

Inherits Class Frame

This class inherits all features of Frame. A MenuBar is automatically created when you create a MainWindow. There is no other way to create a MenuBar.

MenuBar:clear

Parameters: none

Returns: none

Remove all menues from the menu bar.

MenuBar:insertMenu

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	PopupMenu		The menu to insert into the MenuBar
2	number	x	Menu ID (default: autogenerated ID)
3	number	x	Position index within MenuBar

Returns

number (Menu ID)

MenuBar:isEnabled

Parameters: number (Menu ID)

Returns: boolean

MenuBar:removeMenu

Parameters: number (Menu ID)

Returns: none

MenuBar:setCallback

See Widget:setCallback. The following event and callback function is supported:

gui.event.Activated	function(self, menuId)	Parameters: MenuBar, number Returns: none

MenuBar:setEnabled

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Menu ID
2	boolean	x	Enabled (default true)

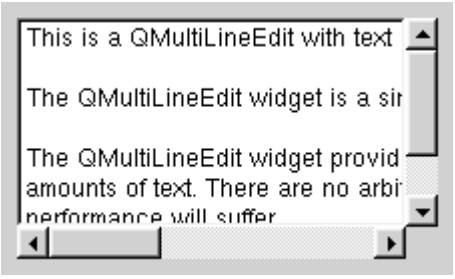
Returns

None

Class MultiLineEdit

Inherits Class Frame

This class inherits all features of Frame. It is a text editor control which allows the user to enter line breaks.



MultiLineEdit:getLine

Parameters: number (line number, starting with 1)

Returns: string

MultiLineEdit:getLineCount

Parameters: none

Returns: number

MultiLineEdit:getText

Parameters: none

Returns: string (the whole text)

MultiLineEdit:insertLine

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The text line
2	number	x	Line number, starting with 1; default: insert at end

Returns

None

MultiLineEdit:isEdited

Parameters: none

Returns: boolean

This flag becomes true, if the text is changed by the user or by a call to setText() or insertLine().

MultiLineEdit:setAlignment

Parameters: number (alignment code)

Returns: none

Use one of the following values: gui.align.Left, gui.align.Right or gui.align.HCenter

MultiLineEdit:setCallback

See Widget:setCallback(). The following events and callback functions are supported:

gui.event.Changed	function(self)	Parameters: MultiLineEdit Returns: none
gui.event.Return	function(self)	Parameters: MultiLineEdit Returns: none

MultiLineEdit:setEdited

Parameters: boolean (default true)

Returns: none

MultiLineEdit:setReadOnly

Parameters: boolean (default true)

Returns: none

MultiLineEdit:setText

Parameters: string

Returns: none

MultiLineEdit:setWordWrap

Parameters: boolean (optional, default true)

Returns: none

If true, the widget automatically breaks text lines, so the text fits into the width of the widget.

Class MyWidget

Implemented by QWidget

MyWidget inherits all features of Widget. It can be used to build custom controls, where all painting and interactivity is handled by Lua functions.

MyWidget:setAcceptFocus

Parameters: boolean (default true)

Returns: none

If true, this widget will accept keyboard focus by either "tabbing" or clicking on it. There can be widgets without focus (e.g. Label) or with focus (e.g. LineEdit).

MyWidget:setAutoBackground

Parameters: boolean (default true)

Returns: none

If true, the widget automatically fills its background with the preset (see Widget:setBgColor) background color. If false, you have to draw the background in your paint handler.

MyWidget:setCallback

See Widget:setCallback(). The following events and callback functions are supported:

gui.event.SizeHint	function(self) return width, heigth	Parameters: MyWidget Returns: number, number
gui.event.MousePressed	function(self, x, y, left, mid, right, shift, ctrl, alt)	Parameters: MyWidget, number, number, 6 x boolean Returns: none
gui.event.MouseReleased	function(self, x, y, left, mid, right, shift, ctrl, alt)	Parameters: MyWidget, number, number, 6 x boolean Returns: none
gui.event.MouseMoved	function(self, x, y, left, mid, right, shift, ctrl, alt)	Parameters: MyWidget, number, number, 6 x boolean Returns: none
gui.event.DblClicked	function(self, x, y, left, mid, right, shift, ctrl, alt)	Parameters: MyWidget, number, number, 6 x boolean Returns: none
gui.event.KeyPressed	function(self, keyCode, text, shift, ctrl, alt) return accepted	Parameters: MyWidget, number, string or nil, 3 x boolean Returns: boolean
gui.event.KeyReleased	function(self, keyCode, text, shift, ctrl, alt) return accepted	Parameters: MyWidget, number, string or nil, 3 x boolean Returns: boolean
gui.event.FocusIn	function(self)	Parameters: MyWidget Returns: none
gui.event.FocusOut	function(self)	Parameters: MyWidget Returns: none
gui.event.Paint	function(self, painter)	Parameters: MyWidget, Painter Returns: none
gui.event.Resized	function(self, w, h, oldW, oldH)	Parameters: MyWidget, number, number, number, number Returns: none
gui.event.Closing	function(self) return accept	Parameters: MyWidget Returns: boolean
gui.event.Showing	function(self)	Parameters: MyWidget Returns: none
gui.event.Hiding	function(self)	Parameters: MyWidget Returns: none

MyWidget:setMouseTracking

Parameters: boolean (default true)

Returns: none

If true, this widget sends mouse move events even if no mouse button is pressed. If false, events are only sent if the user started a mouse drag (i.e. pressed a mouse button) on top of the widget. Mouse tracking is initially disabled since it uses a lot of computing resources.

Class Object

This is the superclass of most classes which are created by the spec Library (see page 167). It implements the following methods which are also available in al subclasses.

Object:getAttr

With this function the attributes of this object can be accessed. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table...").

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute

Returns

Nr.	*Type*	*Opt.*	*Description*
1	nil | string | boolean | number | Record		The value of the attribute or nil, if the attribute is not defined.

`Object:getAttrs`

Parameters: none

Returns: Table[string (attribute name), value (like Object:getAttr)]

`Object:getInstanceName`

Parameters: none

Returns: string (name of object instance or empty)

`Object:setAttr`

This method allows to store custom values in CARA objects. Only attributes stored in Record objects (subclass of Object) are persistent i.e. saved and loaded with the Repository.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute
2	nil | string | boolean | number | Record		The value of the attribute

Returns

none

Class Painter

A painter is automatically created by a paint event on MyWidget or Printer:setup. You can access it within your paint callback function, but nowhere else (i.e. it becomes invalid after the event passed).

`Painter:drawContour`

Draws the given ContourPlot at the given point position. The plot scales into the area given by width and height.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	ContourPlot		
2	number		X position
3	number		Y position
4	number		Width
5	number		Height

Returns

None

Painter:drawEllipse

Draws an ellipse with the current pen and brush at center point x/y with extension width and height.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position
3	number		Width
4	number		Height

Returns

None

Painter:drawIcon

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Icon		
2	number		X position
3	number		Y position

Returns

None

Painter:drawImage

Draws the given image at the given point position. If width and height are left out, the image is painted with its original dimension. You can stretch the image to width and height if needed.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Image		
2	number		X position
3	number		Y position
4	number	x	Width
5	number	x	Height

Returns

None

Painter:drawLine

Draws a line from point x1/y1 to x2/y2 using the current pen.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X1 position
2	number		Y1 position
3	number		X2 position
4	number		Y2 position

Returns

None

Painter:drawPoint

Draws a single point at x/y using the current pen.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position

Returns

None

Painter:drawRect

Draws a rectangle with the current pen and brush, whose left upper edge is at point x/y, extending width to the right and height to the bottom.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position
3	number		Width
4	number		Height

Returns

None

Painter:drawSlice

Draws the given SlicePlot at the given point position. The plot scales into the area given by width and height.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	SlicePlot		
2	number		X position
3	number		Y position
4	number		Width
5	number		Height

Returns

None

Painter:drawText

Draws the text string with the current pen and font at point x/y, where y is at the base line and x at the left edge of the first glyph.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position
3	string		Text to draw

Returns

None

Painter:fillRect

Draws a rectangle with the given brush, whose left upper edge is at point x/y, extending width to the right and height to the bottom. Only fills the area without drawing a border. The brush is defined by the text string (format "#RRGGBB" or color-names).

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position
3	number		Width
4	number		Height
5	string		Fill color name

Returns

None

Painter:getBounding

This method accepts a text string and returns the width and height it will have when drawn on the screen using the current font.

Parameters

string

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		Width
2	number		Height

Painter:lineTo

This function draws a line with the current pen, starting from the last position set by a former lineTo or moveTo.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position

Returns

None

Painter:moveTo

This function sets the current positon to x/y without drawing anything. This is useful when calling lineTo afterwards.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position

Returns

None

Painter:setBrush

Parameter: string (optional)

Returns: none

This function sets the current brush color. All following drawing operations depending on brush are immediately affected. The color is coded using format "#RRGGBB" with R, G and B being hexadecimal numbers, or alternatively by naming an X-Window color like "white", "green", etc.. Omitting the parameter sets the brush to transparent.

Painter:setFont

This function sets the current font, which affects the following calls to drawText.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		Font name
2	number		Point size
3	boolean		Bold
4	boolean		Italic

Returns

None

Painter:setPen

This function sets the current pen color and width. All following drawing operations depending on pen are immediately affected. The color is coded as usual.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		Color name
2	number		Width

Returns

None

Painter:setZoomFactor

Parameters: number

Returns: none

This method sets a factor by which the x and y coordinates are stretched when drawing.

Class Peak

Inherits Class Object

A Peak is owned by a PeakList and represents a single ppm position. It thus has the same number of dimensions and atom types like the peaklist it belongs to. A Peak is uniquely identified by a ID number, which is automatically set when creating the Peak object. This number is visible in the CARA user interface and also in exported peaklists. Additional attributes are amplitude, volume, color and label. Each dimension can also have a number representing a spin assignment (this is an arbitrary number without any constraints).

Peak:getAmp

Parameters: Spectrum (optional)

Returns: Amplitude

Get the Amplitude of the peak, optionally aliased to a certain spectrum. If the spectrum has no alias, the default is used.

Peak:getAssig

Returns the assignment in all dimensions or nil, when it's not defined.

Parameters

None

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number | nil		Assigned Spin ID of dimension D1
n	number | nil		Assigned Spin ID of dimension Dn

Peak:getAttr

With this function the attributes of this object can be accessed. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table...").

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute

Returns

Nr.	*Type*	*Opt.*	*Description*
1	basic types, Record		The value of the attribute or nil, if the attribute is not defined.

See Also

setAttr

Peak:getColor

Parameters: none

Returns: number

This is the color also associated with XEASY peaks. Values 1 to 6 are priviledged in XEASY.

Peak:getGuesses

Parameters: none

Returns: Table[number (Guess ID), Guess]

Peak:getId

This function returns the identification number of the given object. This number is automatically assigned by CARA when creating the object. It is displayed in the user interface and saved in the CARA file. Identification numbers are greater than zero.

Parameters

None.

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		The ID number of the given object, unique within class/repository

Peak:getLabel

Parameters: none.

Returns: string

The label is a free form text string associated with a peak. It is written as a peak comment to XEASY peaklists.

Peak:getModel

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Spectrum	x	Optionally the Spectrum to which the PeakModel applies; if left out, the general default model is returned.

Returns

None

Peak:getPos

Gives the PPM position associated with a peek. When a spectrum is specified, the function returns the alias position (if available) or the default position.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Spectrum	x	If a Spectrum is given the method returns the alias position

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		PPM position in dimension D1
n	number		PPM position in dimension Dn

Peak:getVol

Get the Volume of the peak, optionally aliased to a certain spectrum. If the spectrum has no alias, the default is used.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Spectrum	x	If a Spectrum is given the method returns the alias volume

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		Volume

Peak:setAmp

See PeakList:setAmp.

Peak:setAssig

See PeakList:setAssig.

Peak:setAttr

With this function the attributes of this object can be written. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table..."). If the attribute is read-only, execution is aborted with an error. If the second parameter is omitted, the attribute is set to nil, i.e. it is removed from the object.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute
2	basic types, Record	x	The value to be assigned to the attribute.

Returns

None.

See Also

getAttr

Peak:setColor

See PeakList:setColor.

Peak:setGuess

See PeakList:setGuess.

Peak:setLabel

See PeakList:setLabel.

Peak:setModel

Each Peak is assigned to a PeakModel; the PeakModel is identified by an ID number.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		ID of the PeakModel to be set
2	Spectrum	x	Spectrum to which the PeakModel specifically applies or nil

Returns

None.

Peak:setPos

See PeakList:setPos.

Peak:setVol

See PeakList:setVol.

Class PeakList

Inherits Class Object

A PeakList is a container of Peaks. It can be read from or written to a file. PeakLists can also be part of a Repository, but mainly for backward compatibility with old XEASY. Only PeakLists belonging to a Repository have an ID number different to zero.

A PeakList gets its invariable number of dimensions and atom type per dimension when it is created with spec.createPeakList or loaded with spec.openPeakList. It can optionally have a batch list, which is simply an array of spectrum IDs (the spectra have to be resolved using Project:getSpectrum).

`PeakList:createBackcalc`

This method creates a Spectrum which is calculated from the given PeakList (see [3] section 4.5.1 for a description of the method).

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Spectrum		Reference Spectrum for Scales and difference calculations.
2	bool	x	true..calculation difference from reference Spectrum, false (default)..just calculate plain spectrum from PeakList
2	bool	x	true..do an exact calculation false (default)..do an approximation

Returns

Nr.	*Type*	*Opt.*	*Description*
1	Spectrum		Spectrum calculated from the PeakList

`PeakList:createGuess`

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Peak		The Peak for which the tentative assignment is given
2	number	x	Probability (0..1) of the tentative assignment (default 0)
3	number	x	ID of the tentative spin the peak dimension D1 belongs to
n	number	x	ID of the tentative spin the peak dimension Dn belongs to

Returns

Nr.	*Type*	*Opt.*	*Description*
1	Guess		The new Guess of the Peak

`PeakList:createModel`

Parameters: none

Returns: PeakModel

`PeakList:createPeak`

Creates a new peak at the given coordinates (which will be used as default). The peak automatically gets a unique ID number. It is an error to pass the wrong number of parameters.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		PPM position dimension D1
n	number		PPM position dimension Dn

Returns

Nr.	*Type*	*Opt.*	*Description*
1	Peak		The new Peak

PeakList:getAtomType

Parameters: number (Dimension)

Returns: string (atom type symbol)

PeakList:getAttr

With this function the attributes of this object can be accessed. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table...").

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute

Returns

Nr.	*Type*	*Opt.*	*Description*
1	basic types, Record		The value of the attribute or nil, if the attribute is not defined.

See Also

setAttr

PeakList:getBatchList

Parameters: none

Returns: Table[number (index 1..n), number (spectrum ID)]

This function returns the spectrum batch list associated with a peaklist as an array of spectrum IDs. These IDs should reference a valid spectrum, but don't have to (e.g. if the spectrum was deleted in mean time). The following sample converts all spectrum IDs in the batch list to real spectrum references:

```
batch = peakList:getBatchList()
local pro = cara:getProject()
local spec
-- remember: batch still contains spectrum IDs
for i= 1, table.getn( batch ) do
     spec = pro:getSpectrum( batch[ i ] ) -- try to get a reference
     if spec == nil then
          -- it was obviously a wrong ID
          error( string.format(
               "Invalid spectrum ID %d", batch[ i ] ) )
     end
     batch[ i ] = spec -- the ID is replaced by the reference
end
```

PeakList:getDimCount

Parameters: none

Returns: number (number of dimensions of PeakList)

PeakList:getHome

Parameters: none

Returns: number (Spectrum ID or 0)

Each PeakList can be associated with a specific spectrum. This is usually the spectrum the PeakList was created for.

PeakList:getId

This function returns the identification number of the given object. This number is automatically assigned by CARA when creating the object. It is displayed in the user interface and saved in the CARA file. Identification numbers are greater than zero.

Parameters

None.

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		The ID number of the given object, unique within class/repository

PeakList:getIsotope

Parameters: Dimension

Returns: string (the isotope label of the given dimension)

The PeakList usually inherits its dimension attributes from the home Spectrum for which it was created.

PeakList:getModel

Parameters: number (ID of the model)

Returns: PeakModel or nil

PeakList:getName

Parameters: none

Returns: string (the name associated with the peaklist)

PeakList:getPeak

Parameters: number (the peak ID number)

Returns: Peak or nil

If there is no peak known with the given ID, the function returns nil.

```
peak = cara:getProject():getPeakList( 1 ):getPeak( 53 )
```

PeakList:getPeaks

Parameters: none

Returns: Table[number (peak ID), Peak]

This function returns all peaks of the peaklist as a table, indexed by the peak ID number.

PeakList:removeGuess

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Peak		The Peak from which to remove the Guess
2	Guess		The Guess to remove from the given Peak

Returns

None

PeakList:removePeak

Parameters: Peak

Returns: none

This function removes the given Peak from the peaklist. The Peak is still alive afterwards, but no longer useful. It will be garbage collected as soon as it is no longer stored in a Lua variable.

PeakList:rotate

The dimension mapping of the PeakList can only be set as long as the PeakList doesn't belong to a Repository. There must be exactly one parameter for each dimension and the mappings must be disjoint.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Dimension		The new dimension mapping for dimension D1
n	Dimension		The new dimension mapping for dimension Dn

Returns
None

PeakList:saveToFile

Parameters: string (file path)

Returns: none

Use this function to write the peaklist to a file in XEASY format. When writing is unsuccessful, the script aborts with an error.

PeakList:setAmp

Use this function to set either the default (if spectrum is omitted) or an aliased amplitude of the given peak. The amplitude is set to 0 if the parameter is omitted.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Peak		The Peak whose amplitude is to be set
2	number	x	The new amplitude of the Peak (0 is default)
3	Spectrum	x	Set the amplitude at the aliassed position in the given Spectrum

Returns
None

PeakList:setAssig

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Peak		The Peak whose assignment is to be set
2	number		The Spin ID for dimension D1
n	number		The Spin ID for dimension Dn

Returns
None

PeakList:setAttr

With this function the attributes of this object can be written. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table..."). If the attribute is read-only, execution is aborted with an error. If the second parameter is omitted, the attribute is set to nil, i.e. it is removed from the object.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute
2	basic types, Record	x	The value to be assigned to the attribute.

Returns
None

See Also
getAttr

PeakList:setBatchList

Parameters: Table[number (array index), number (Spectrum ID)]

Returns: none

Use this function to set the spectrum batch list of a peaklist. The table is an array of spectrum IDs in the order you provide. The spectrum IDs should be valid, but don't have to.

PeakList:setColor

Assign a color to the Peak using a color code from the color table of the Repository. The color table is persistent and can be configured from the user interface of CARA (Explorer / Setup Peak Colors).

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Peak		The Peak whose color is to be set
2	number		The new color code of the Peak

Returns

None.

PeakList:setGuess

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Peak		The Peak whose Guess is to be set
2	Guess		The Guess to be set
3	number (0..1)	x	The new probability of the Guess, default is 0
4	number	x	The new assignment of the Guess in dimension D1
n	number	x	The new assignment of the Guess in dimension Dn

Returns

None.

PeakList:setHome

Parameters: Spectrum

Returns: none

Use this method to set the new home Spectrum of the PeakList. The method is only available for PeakLists which don't belong to a Repository.

PeakList:setLabel

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Peak		The Peak whose label is to be set
2	string	x	The new label of the Peak; default is an empty string

Returns

None.

PeakList:setName

Parameters: string (the new name of the PeakList)

Returns: none

PeakList:setPos

Use this function to set the PPM position of the given Peak. If Spectrum is omitted, the default position is set. Otherwise the alias position is set.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Peak		The Peak whose position is to be set
2	number		The new PPM position in dimension D1
n	number		The new PPM position in dimension Dn
n+1	Spectrum	x	The Spectrum in case of an alias position

Returns

None.

PeakList:setVol

Use this function to set either the default (if Spectrum is omitted) or an aliased volume of the given Peak. Volume is set to 0 when omitted.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Peak		The Peak whose volume is to be set
2	number	x	The new volume of the Peak; default is 0
3	Spectrum	x	The Spectrum in case of an alias volume

Returns

None.

Class PeakModel

Inherits Class Object

This class defines the model which is used for peak shapes in back-calculation.

PeakModel:getAmplitude

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Table[Dim, PPM]		The point p for which the amplitude is calculated
2	Table[Dim, PPM]		The reference point where the amplitude is maximum
3	number		The amplitude at the reference point

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		The amplitude at point p

PeakModel:getBalance

Parameters: Dimension

Returns: number

The model is a superposition of a Gauss and Lorentz shape. If balance is 1.0, then the shape is fully gaussian. If balance is 0.0, then the shape is fully lorentzian. See [3] Eq. 12.

PeakModel:getDimCount

Parameters: none

Returns: number (the number of dimensions of the model)

PeakModel:getGain

Parameters: Dimension

Returns: number

Gain is used to boost the peak width. See MonoScope for how it is used.

PeakModel:getId

Parameters: none

Returns: number (ID number identifying the model)

PeakModel:getMaxWidth

Parameters: Dimension

Returns: number (PPM)

This method calculates the maximum width per dimension. This is a convenience number depending on balance, gain and width used to display the base width marks on the MonoScope slices.

PeakModel:getTol

Parameters: Dimension

Returns: number

This is a factor (0..2) used to broaden the peak width when looking for the maximum amplitude.

PeakModel:getWidth

Parameters: Dimension

Returns: number (the PPM value set by PeakModel:setWidth)

PeakModel:setBalance

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Dimension		The dimension of the model for which the balance is set
2	number (0..1)		The balance.

Returns

None

The model is a superposition of a Gauss and Lorentz shape. If balance is 1.0, then the shape is fully gaussian. If balance is 0.0, then the shape is fully lorentzian. See [3] Eq. 12.

PeakModel:setGain

Parameters

Nr.	Type	Opt.	Description
1	Dimension		The dimension of the model for which the gain is set
2	number (0..1)		The gain.

Returns

None

Gain is used to boost the peak width. See MonoScope for how it is used.

PeakModel:setTol

Parameters

Nr.	Type	Opt.	Description
1	Dimension		The dimension of the model for which the tolerance is set
2	number (0..2)		The tolerance.

Returns

None

This is a factor (0..2) used to broaden the peak width when looking for the maximum amplitude.

PeakModel:setWidth

Parameters

Nr.	Type	Opt.	Description
1	Dimension		The dimension of the model for which the width is set
2	number (>0)		The width in PPM.

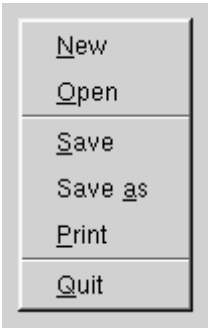
Returns

None

Class PopupMenu

Inherits Class Frame

This class inherits all features of Frame. A PopupMenu is created by gui.createPopupMenu. It can be inserted into a MenuBar or used as an independent popup menu, e.g. as a reaction when the user clicks the right mouse button. PopupMenus can be cascaded to build menu hierarchies. Menu items are identified by menu ID numbers, which can be defined by the programmer or automatically by the widget.



PopupMenu:clear

Parameters: none

Returns: none

Remove all menu items and sub-menus from this menu.

PopupMenu:getText

Parameters: number (menu ID)

Returns: string

PopupMenu:insertItem

Insert a new menu item at index or the end of the menu. Give it an explicit menu ID or accept the one automatically given to it by the widget.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		Menu text
2	number	x	Menu ID (default: autogenerated ID)
3	number	x	Position index within PopupMenu

Returns

number (Menu ID)

PopupMenu:insertSeparator

Parameters: none

Returns: none

Insert a menu separator line at the end of the menu.

PopupMenu:insertSubmenu

Insert a new sub-menu at index or the end of the menu. Give it an explicit menu ID or accept the one automatically given to it by the widget.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	PopupMenu		
2	string		Menu text
3	number	x	Menu ID (default: autogenerated ID)
4	number	x	Position index within PopupMenu

Returns

number (Menu ID)

PopupMenu:isChecked

Parameters: number (menu ID)

Returns: boolean

PopupMenu:isEnabled

Parameters: number (menu ID)

Returns: boolean

PopupMenu:popup

Use this function to open the PopupMenu at the given position (e.g. as reaction on a right-click on the mouse button).

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position

Returns

None

PopupMenu:removeItem

Parameters: number (menu id)

Returns: none

PopupMenu:setAccel

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Menu ID
2	string		Accelerator code, e.g. "CTRL+S"

Returns

None

PopupMenu:setCallback

See Widget:setCallback. The following events and callback functions are supported:

gui.event.Activated	function(self, menuId)	Parameters: PopupMenu, number Returns: none
gui.event.Showing	function(self, menuId)	Parameters: PopupMenu, number Returns: none

PopupMenu:setChecked

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Menu ID
2	boolean		Checked, default true

Returns

None

PopupMenu:setEnabled

Parameters

Nr.	Type	Opt.	Description
1	number		Menu ID
2	boolean		Enabled, default true

Returns

None

PopupMenu:setIcon

Parameters

Nr.	Type	Opt.	Description
1	number		Menu ID
2	Icon		see Class Icon

Returns

None

PopupMenu:setText

The text can contain shortcuts, e.g. "Do &This". The "T" is underlined as a shortcut. The user can trigger the shortcut pressing ALT+T when the menu is shown.

Parameters

Nr.	Type	Opt.	Description
1	number		Menu ID
2	string		Menu text

Returns

None

PopupMenu:setWhatsThis

Parameters

Nr.	Type	Opt.	Description
1	number		Menu ID
2	string		Text

Returns

None

Class Printer

This class represents the printing function of the operating system. It is also possible to print to a PDF or SVG. You can create a Printer by dlg.createPrinter and adjust the settings. Then call Printer:setup and pass a Lua function which does the painting.

Printer:abort

Parameters: none

Returns: boolean

Aborts the current print run. Returns true if the print run was successfully aborted.

Printer:getFromPage

Parameters: none

Returns: number

Returns the number of the first page in a range of pages to be printed (the "from page" setting). Pages in a document are numbered according to the convention that the first page is page 1. By default, this function returns a special value of 0, meaning that the "from page" setting is unset.

Printer:getNumCopies

Parameters: none

Returns: number

Returns the number of copies to be printed. The default value is 1.

Printer:getToPage

Parameters: none

Returns: number

Returns the number of the last page in a range of pages to be printed.

Printer:newPage

Parameters: none

Returns: boolean

Tells the printer to eject the current page and to continue printing on a new page. Returns true if this was successful; otherwise returns false.

Printer:setCreator

Parameters: string

Returns: none

Sets the name of the application that created the document. This function is only applicable to the X11 version of Qt.

Printer:setDocName

Parameters: string

Returns: none

Sets the document name. On X11, the document name is for example used as the default output filename.

Printer:setMinMax

Sets the page range to be from min to max.

Parameters

Nr.	Type	Opt.	Description
1	number		Min. page number (starting with 1)
2	number		Max. page number

Returns

None

`Printer:setOrientation`

Parameters: number (0..Portrait, 1..Landscape)

Returns: none

`Printer:setPageSize`

Parameters: number (0..A4, etc., see QPrinter::PageSize in [8])

Returns: none

`Printer:setup`

Parameters: function (does the paint work)

Returns: boolean

This method opens the print setup dialog. If the user confirms the dialog, the function handed in as a parameter is called and true is returned. If the user cancels the dialog, false is returned. The function which does the paint work is expected to have the following parameters:

Nr.	Type	Opt.	Description
1	Printer		The Printer currently executing "setup"
2	Painter		See Class Painter

Here is an example which prints a page number on three pages:

```
local printer = gui.createPrinter()
local function doPrint( printer, painter )
     for i = 1, 3 do
          painter:drawText(30,30,"Page "..i)
          if i < 3 then
               printer:newPage()
          end
     end
end
printer:setup( doPrint )
```

Class Project

Inherits Class Object.

A project is part of a repository and contains all relevant data of an NMR analysis process, i.e. spectra, spins, spin systems, etc. A project makes use of the configuration given by the repository, i.e. the spectrum and residue types.

Project:addCandidate

Use this function to associate the given SpinSystem with the given ResidueType as a candidate assignment. CARA then restricts the sequence mapping of the SpinSystem to residues of the given type. Each SpinSystem can have zero or more candidates.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	SpinSystem		
2	ResidueType		

Returns

None

Project:addPeakList

Parameters: PeakList

Returns: number (ID of PeakList within Project)

Project:addResidue

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		Short name of ResidueType
2	string	x	Name of the chain; may be empty
3	number	x	Number of Reqidue within the chain (not the same as ID)

Returns

Residue

Project:addSpectrum

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Spectrum		The spectrum to be added
2	string		Valid short name of the SpectrumType
3	number	x	Mapping of dimension D1 (default: auto-rotate)
n	number	x	Mapping of dimension Dn (default: auto-rotate)

Returns

number (ID of added Spectrum)

Project:addStructure

Parameters: none

Returns: Structure

Project:assignSpin

Parameters: Spin, SpinSystem

Returns: none

This function tries to assign the given Spin to the given SpinSystem. The function aborts the script with an error, if it would break consistency of the model.

Project:assignSystem

This function tries to assign the given SpinSystem to the Residue. For consistency reasons, all the SpinSystems of the fragment, this SpinSystem is part of, are assigned to their corresponding residues. The function aborts with an error, when the transaction would break model consistency.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	SpinSystem		
2	Residue		

Returns

None

Project:calcLevels

Parameters: number (ID of Spectrum of which levels shall be recalculated)

Returns: none

Project:createSpin

This function creates a new Spin of the given atom type at the PPM position. When Spectrum is omitted, the default position is set. Otherwise the alias position for the given Spectrum is set. The Spin automatically gets a unique ID number.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		Atom type or isotope symbol
2	number		PPM position of the spin
3	Spectrum	x	Hand in a Spectrum if the position is an alias

Returns

None

Project:createSystem

Parameters: none

Returns: SpinSystem.

This function creates a new SpinSystem and automatically gives it a unique ID number.

Project:getAttr

With this function the attributes of this object can be accessed. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table...").

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute

Returns

Nr.	Type	Opt.	Description
1	basic types, Record		The value of the attribute or nil, if the attribute is not defined.

See Also

setAttr

Project:getCalibration

Parameters

Nr.	Type	Opt.	Description
1	number		ID of the Spectrum of which the calibration is requested
2	number		Dimension of which the calibration is requested

Returns

number (PPM offset)

Project:getCombinedFragment

This function returns an array representing the concatenation of the fragments, to which the two given spin systems belong to. The original fragment are cut right of predecessor and left of successor and then glued as predecessor-successor. If both spin systems belonged to the same original fragment, it is returned and no splicing occurs.

Parameters

Nr.	Type	Opt.	Description
1	SpinSystem		The predecessor SpinSystem
2	SpinSystem		The successor SpinSystem

Returns

Table[number (array index), SpinSystem]

Project:getFragment

Parameters: SpinSystem

Returns: Table[number (array index), SpinSystem]

This function simply returns an array representing the spin system fragment, to which the given spin system belongs.

Project:getName

Parameters: none

Returns: string (the name of the project)

Project:getOrigin

For convenience a reference spatial position can be stored in Project.

Parameters

None

Returns

Nr.	Type	Opt.	Description
1	number		Coordinate in dimension x
2	number		Coordinate in dimension y
3	number		Coordinate in dimension z

Project:getPeakList

Parameters: number (ID of PeakList)

Returns: PeakList or nil

This function tries to find a PeakList with the given ID number. If it finds one, it will be returned.

```
peakList = cara:getProject():getPeakList( id )
if peakList == nil then
     dlg.showError( "Select Peaklist", "Unknown peaklist" )
     return
end
```

Project:getPeakLists

Parameters: none

Returns: Table[number (peaklist ID), PeakList]

This function returns all PeakLists of the Project in an array indexed by the ID of the PeakList.

Project:getResidue

Parameters: number (residue number)

Returns: Residue or nil.

This function tries to find a residue with the given ID number. If it finds one, it will be returned. In contrast to other ID numbers, the ID number of the residue corresponds to its ordering within the sequence. But notice: residue numbers can have any value (even negative) and can also have gaps. For iterating along the residues use Residue:getPred and Residue:getSucc instead.

Project:getSequence

Parameters: none

Returns: Table[number (residue number), Residue]

This function returns the sequence of the project as an array indexed by the residue number. Please notice, that residue numbers can have gaps and be of any value. Iterate through this array e.g. using forall of the Lua library or a simple for loop (see example).

```
s = cara:getProject():getSequence()
for a,b in pairs( s ) do
     print( a.." "..b:getType():getShort() )
end
```

Project:getSpectra

Parameters: none

Returns: Table[number (spectrum ID), Spectrum]

This function returns all spectra of the project in an array indexed by the spectrum ID.

Project:getSpectrum

Parameters: number (Spectrum ID)

Returns: Spectrum or nil

This function tries to find a Spectrum with the given ID number. If it finds one, it is returned.

Project:getSpin

Parameters: number (Spin ID)

Returns: Spin or nil

This function tries to find a Spin with the given ID number. If it finds one, it is returned.

Project:getSpinLinks

Parameters: none

Returns: Table[number (array index), Spectrum]

This function returns all SpinLinks of the Project in an array.

```
l = cara:getProject():getSpinLinks()
for a,b in pairs( l ) do print( b:getLeft().." "..b:getRight() ) end
```

Project:getSpins

Parameters: none

Returns: Table[number (Spin ID), Spin]

This function returns all Spins of the Project in an array indexed by the Spin ID.

```
s = cara:getProject():getSpins()
for a,b in pairs( s ) do print( a.." "..b:getLabel() ) end
```

Project:getStructure

Parameters: number (ID of the Structure)

Returns: Structure or nil

Project:getStructures

Parameters: none

Returns: Table[number (Structure ID), Structure]

Project:getSystem (also getSpinSystem)

Parameters: number (SpinSystem ID)

Returns: SpinSystem or nil

This function tries to find a SpinSystem with the given ID number. If it finds one, it will be returned.

Project:getSystems (also getSpinSystems)

Parameters: none

Returns: Table[number (spin system ID), SpinSystem]

This function returns all SpinSystem of the Project in an array indexed by the SpinSystem ID.

Project:getTolerance

Parameters: string (atom type symbol)

Returns: number (matching tolerance in PPM)

This is the tolerance with used by the spin matching algorithm (see [3] Eq. 6).

Project:linkSpins

This functions links two arbitrary Spins and returns the corresponding SpinLink. If the Spins were already linked, the existing SpinLink is returned.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Spin		First Spin to link
2	Spin		Second Spin to link

Returns

SpinLink

Project:linkSystems

Use this function to combine spin systems to fragments. The function maintains and checks the consistency of the model. If both SpinSystems were already assigned to a residue, the fragment is only built when their sequence assignments are continuous. If only one of the spin systems was assigned, the function assigns the other spin system accordingly. If this is not possible, the function aborts with an error.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	SpinSystem		Predecessor SpinSystem
2	Spin		Successor SpinSystem

Returns

None

Project:matchResidue

This function calculates a match of the given SpinSystem to the given Residue (see [3] section 4.2.2). Only Spins labeled with the given offset (default 0) are used for the match. If a Spectrum is provided, its alias spin positions are used (default positions otherwise).

The returned rating represents the "fitness" of the match, ranging from 0 to the number of involved Spins (given by the weight). The zero count represents the number of Spins with a potential match to the values of the Residue, but beeing out of its deviation. Excluded is true, if the SpinSystem has a candidate set and the Residue is not part of it. Assigned is true, if the Residue already is assigned to a SpinSystem.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	SpinSystem		SpinSystem to match with Residue
2	Residue		
3	number	x	Offset (default 0)
4	Spectrum	x	

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		Rating, fitness of the match; range 0 to weight
2	number		Weight
3	number		Zero count
4	boolean		Excluded
5	boolean		Assigned

Project:matchSpin

This function calculates a match of the given Spin (PPM position) to the given SpinSystem. Only Spins with the same Atom type are matched. If a Spectrum is provided, its alias Spin positions are used (default positions otherwise).The match is influenced by the tolerance value corresponding to the Atom type (see Project:getTolerance). The rating is a value between 0 and the number of matching Spins (given by weight). Zero represents the number of Spins with equal atom type beeing out of the tolerance (and thus not contributing to weight or rating).

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Spin		Spin to match to SpinSystem
2	SpinSystem		
3	Spectrum	x	

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		Rating
2	number		Weight
3	number		Zero count

Project:matchSystems

This function does a full cross correlation of the Spins of both SpinSystem. The match is rated using the tolerance value corresponding to the atom types of the compared Spins. If a Spectrum is provided, its alias Spin positions are used (default positions otherwise). The rating is a value between 0 and the number of matching Spins (given by weight). Zero represents the number of Spins with equal atom type or label beeing out of the tolerance (and thus not contributing to weight or rating).

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	SpinSystem		
2	SpinSystem		
3	boolean	x	If true use only atom types (not labels) for matching; if false only spins with equal labels are correlated (considering 0 and -1 offsets)
4	Spectrum	x	

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		Rating
2	number		Weight
3	number		Zero count

Project:removeCandidate

This function removes the given ResidueType from the assignment candidate list of the given SpinSystem. See also Project:addCandidate.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	SpinSystem		
2	ResidueType		

Returns

None

Project:removePeakList

Parameters: PeakList

Returns: none

Project:removeSpin

Parameters: Spin

Returns: none

This function tries to remove the Spin from the Project. Consistency rules demand that the Spin is not part of an assignment (SpinSystem, etc.). Otherwise the function is aborted with an error. The removed Spin is still alive but useless; it will be deleted by the garbage collector.

Project:removeStructure

Parameters: Structure

Returns: none

Project:removeSystem (also removeSpinSystem)

Parameters: SpinSystem

Returns: none

This function tries to remove the SpinSystem from the Project. Consistency rules demand that the SpinSystem is not part of an assignment (Spin, Residue, etc.). Otherwise the function is aborted with an error. The removed SpinSystem is still alive but useless; it will be deleted by the garbage collector.

Project:renameSpectrum

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Spectrum		The Spectrum to rename (must belong to the Project)
2	string		The new name (may not be empty; must be unique in Project)

Returns

None.

Project:setAttr

With this function the attributes of this object can be written. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table..."). If the attribute is read-only, execution is aborted with an error. If the second parameter is omitted, the attribute is set to nil, i.e. it is removed from the object.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute
2	basic types, Record	x	The value to be assigned to the attribute.

Returns

None.

See Also

getAttr

Project:setCalibration

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Valid Spectrum ID
2	number		Valid dimension in Spectrum
3	number		New PPM offset

Returns

None

Project:setLabel

This function tries to set the label of the given Spin. The label must obey the CARA spin label syntax (see [3] Eq. 4). When left out, it defaults to the empty label. If the Spin belongs to a SpinSystem, the label must be acceptable by the SpinSystem. In case of a consistency violation the execution is aborted with an error.

Parameters

Nr.	Type	Opt.	Description
1	Spin		
2	string	x	Spin label

Returns

None

Project:setLinkParams

Parameters

Nr.	Type	Opt.	Description
1	SpinLink		
2	Spectrum or nil	x	Optionally a SpinLink can have individual parameters per Spectrum; if Spectrum is nil, the default parameters of the SpinLink are set.
3	number	x	Rate (default 0)
4	number (0..255)	x	Color code (default 0)
5	boolean	x	Visible (default true)

Returns

None

Project:setLocation

For convenience a spatial position can be stored directly in a Spin object. See also Structure on page 137 or Conformer on page 44.

Parameters

Nr.	Type	Opt.	Description
1	Spin		
2	number	x	Spatial position dimension x
3	number	x	Spatial position dimension y
4	number	x	Spatial position dimension z
5	number	x	Position deviation (error radius)

Returns

None

Project:setOrigin

For convenience a spatial reference position can be stored in the Project object. See also Structure:setOrigin.

Parameters

Nr.	Type	Opt.	Description
1	number	x	Spatial position dimension x
2	number	x	Spatial position dimension y
3	number	x	Spatial position dimension z

Returns

None

Project:setShift

This function sets the default or alias (if spectrum provided) shift of the given spin.

Parameters

Nr.	Type	Opt.	Description
1	Spin		
2	number		PPM shift
3	Spectrum	x	

Returns

None

Project:setSystemType

This function sets the spin system type of the given spin system. This is useful for a homonuclear, sequential assignment strategy.

Parameters

Nr.	Type	Opt.	Description
1	SpinSystem		
2	SystemType		

Returns

None

Project:setTolerance

Sets the matching tolerance of the given atom type. See also Project:getTolerance.

Parameters

Nr.	Type	Opt.	Description
1	string		Atom type symbol
2	number		PPM tolerance

Returns

None

Project:setValue

Use this function to set the (random coil) PPM values associated with the given Residue. These values are used by Project:matchResidue, comparing the Spins with the random coil values. Mean and deviation can be omitted, in which case the value for the given atom label is removed from the residue.

Parameters

Nr.	Type	Opt.	Description
1	Residue		
2	string		Atom label
3	number	x	Mean value (default "undefined")
4	number	x	Deviation value

Returns

None

Project:unassignSpin

Parameters: Spin

Returns: none

Removes the given Spin from its previously assigned SpinSystem, if it has one. Otherwise the execution is aborted with an error.

Project:unassignSystem

Parameters: SpinSystem

Returns: none

Unassigns the given SpinSystem from the previously assigned Residue. For consistency reasons, the whole fragment, to which this SpinSystem belongs is unassigned. If there was no assignment, the execution is aborted with an error.

Project:unlinkSpins

If the two Spins were linked, the function removes the corresponding SpinLink from the project. Nothing happens, when the Spins were not linked.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Spin		
2	Spin		

Returns

None

Project:unlinkSystems

This function unlinks the two SpinSystems, i.e. the fragment is cut between the SpinSystems. The Residue assignments of the SpinSystem remain the same after the cut. If the SpinSystems were not linked, execution is aborted with an error.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	SpinSystem		
2	SpinSystem		

Returns

None

Class ProtonList

Inherits Class Object

This class can be used to load and save XEASY proton list files. This feature allows custom selections of the Spins to be written to the file which are not supported by the user interface (e.g. using spins with -1 offset from neighbour SpinSystems when the corresponding local Spin is missing). The following example shows how to open and iterate a protonlist, whose path can be selected in a file dialog.

```
pl = spec.openProtonList( dlg.getOpenFileName() )
for i = 1, pl:getCount() do
     print( pl:getAtom( i ) )
end
```

ProtonList:getAtom

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Array index, starting with 1

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		Spin ID
2	number		PPM shift
3	string		Atom label
4	number		SpinSystem ID

ProtonList:getCount

Parameters: none

Returns: number (the number of atoms i.e. entries in the list)

ProtonList:isValid

Parameters: number (array index)

Returns: boolean

An entry in the proton list is considered invalid if its Spin ID equals to -9999 and its shift value equals to 999. This is just by convention, originally defined in XEASY and DYANA.

ProtonList:saveToFile

Parameters: string (file path)

Returns: none

This function tries to write the ProtonList to the file with the given path. If the file cannot be written, the function is aborted with an error. Notice that existing files with the same path will be overwritten.

ProtonList:setAtom

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Array index
2	number		Spin ID
3	number		PPM shift
4	string		Atom label
5	number		SpinSystem ID

Returns

None

Class PushButton

Implemented by QPushButton

PushButton conceptually inherits all features of Class Button. You can set a Lua function as a callback, to be executed when a user presses the button.

`PushButton:setDefault`

Parameters: boolean (default true)

Returns: none

Sets this button to be the default button of the dialog box, i.e. its callback is executed when the user presses the RETURN key.

`PushButton:setFlat`

Parameters: boolean (default true)

Returns: none

If true, the button is drawn without a border, but highlighted when the mouse hovers over it.

`PushButton:setOn`

Parameters: boolean (default true)

Returns: none

Set the button to the "pressed" state. Only useful for toggle buttons (see PushButton:setToggleButton).

`PushButton:setPopup`

Parameters: PopupMenu

Returns: none

Set the popup menu to be opened, when the user presses this button.

`PushButton:setToggleButton`

Parameters: boolean (default true)

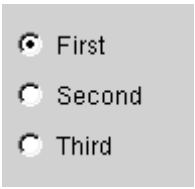
Returns: none

If true, the button has to states, either up or down. If false, the button has no states, i.e. goes automatically up if the user releases the mouse.

Class RadioButton

Implemented by QRadioButton

This class conceptually inherits all features of Class Button. RadioButtons are useful as child widgets of ButtonGroup, where they can represent mutually exclusive options.



RadioButton:isChecked

Parameters: none

Returns: boolean

RadioButton:setChecked

Parameters: boolean (default true)

Returns: none

When this button is part of a ButtonGroup, the other buttons of the group are automatically unchecked.

Class Record

Inherits Class Object

A record is a flexible, persistent data structrure exclusively available to Lua programmers. They can be used to build hierarchies or networks of data objects, which are automatically saved and loaded with the CARA repository. You can use records to build your own "repositories", containing which ever objects you want to manage.

Each record is identified by an OID (object identity), which is a number greater than zero, automatically and invariably set by CARA when creating the record (see Repository:createRecord).

As with all other classes you can set and read arbitrary attributes on records. All attributes (also the ones of objects of other classes) can contain a reference to a Record. Technically, an OID is stored in an attribute. CARA does the conversion between OIDs and references automatically, invisible to the programmer (invalid OIDs are mapped to nil). In fact you normally should never get in contact with OIDs, but they are there anyway.

Record:getAttr

With this function the attributes of this object can be accessed. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table...").

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute

Returns

Nr.	*Type*	*Opt.*	*Description*
1	basic types, Record		The value of the attribute or nil, if the attribute is not defined.

See Also

setAttr

Record:getAttrs

Parameters: none

Returns: Table[string (attribute name), basic type or Record (value)]

Use this function to get a table containing all attributes (i.e. fields) of this Record.

Record:getId

Parameters: none

Returns: number (object ID)

Record:setAttr

With this function the attributes of this object can be written. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table..."). If the attribute is read-only, execution is aborted with an error. If the second parameter is omitted, the attribute is set to nil, i.e. it is removed from the object.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute
2	basic types, Record	x	The value to be assigned to the attribute.

Returns

None.

See Also

getAttr

Class Repository

Inherits Class Object

There is exactly one Repository in each running instance of CARA. This Repository is accessible to Lua by means of the global variable called `cara`. Whenever the user creates or opens another Repository, this variable is automatically updated accordingly.

In the following example we get a reference to a Project called "test".

```
p = cara:getProject( "test" )
```

Repository:createProject

Parameters: string (name of Project)

Returns: Project

Aborts with error if name is not unique.

Repository:createRecord

Parameters: none

Returns: Record

This function creates a new persistent Record within the current Repository. You can set or read attributes of this Record. It is automatically saved with the Repository and available next time you open it. CARA manages Record identity using **O**bject **ID**entities (OID), which are positive numbers starting from one. Whenever you "store" a Record in any attribute, its OID is actually stored. But you don't even notice this, since CARA does the translation from OID to Record reference (and vice versa) automatically.

Repository:getAttr

With this function the attributes of this object can be accessed. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table...").

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute

Returns

Nr.	*Type*	*Opt.*	*Description*
1	basic types, Record		The value of the attribute or nil, if the attribute is not defined.

See Also

setAttr

Repository:getAuthor

Parameters: none

Returns: string

This function returns the author attribute of the repository (identical to cara:getAttr("Author"));

Repository:getExperiment

This method makes the path simulation engine accessible to Lua. The simulation is done for the given ResidueType. If the latter is nil, a cross-product of the SpectrumType dimension labels is calculated.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	SpectrumType		
2	ResidueType	x	Default nil

Returns

Nr.	*Type*	*Opt.*	*Description*
1	Experiment		

Repository:getProject

Parameters: string (project name, optional)

Returns: Project

This function tries to find a Project with the given name (case sensitive). If it finds one, it will be returned (nil otherwise). The name parameter can be left out in case where the Repository only contains one Project.

Repository:getProjects

Parameters: none

Returns: Table[string (project name), Project]

Returns an array containing all Projects of the Repository, indexed by Project name.

Repository:getRecord

Parameters: number (OID)

Returns: Record or nil

This function simply returns nil, if a Record with the given OID is not part of the Repository (e.g. when you removed it before).

Repository:getRecords

Parameters: none

Returns: Table[number (OID), Record]

Returns a list of all Records of the Repository.

Repository:getResidueType

Parameters: string (short name)

Returns: ResidueType or nil

Repository:getResidueTypes

Parameters: none

Returns: Table[string (short name), ResidueType]

Returns an array containing all ResidueType of the Repository, indexed by short name.

Repository:getScript

Parameters: string (name of script)

Returns: string (Lua source code)

Repository:getScriptNames

Parameters: none

Returns: Table{ number (array index), string (script name)]

Repository:getSpectrumType

Parameters: string (name)

Returns: SpectrumType or nil

Repository:getSpectrumTypes

Parameters: none

Returns: Table[string (name), SpectrumType]

Returns a table containing all spectrum types of the repository, indexed by name.

Repository:getSystemTypes

Parameters: none

Returns: Table[number (ID of SystemType), SystemType]

Returns an array containing all system types of the repository, indexed by ID number.

Repository:new

Parameters: string (optional, path to template)

Returns: none

This method loads an empty Repository, optionally initialized with a template.

Repository:open

Parameters: string (file path)

Returns: none

This method loads the Repository specified by the file path.

Repository:removeRecord

Parameters: Record

Returns: none

Removes the given Record from the Repository. Nothing happens, if the Record was not part of the Repository. When a Record is removed, it is still alive but useless (subject to garbage collection).

Repository:save

Parameters: string (optional, new file path)

Returns: none

This method saves the Repository to a file. If the Repository was already loaded from or saved to a file the parameter can be left out. At any time you can hand in a different file path to save the Repository to another file.

Repository:setAttr

With this function the attributes of this object can be written. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table..."). If the attribute is read-only, execution is aborted with an error. If the second parameter is omitted, the attribute is set to nil, i.e. it is removed from the object.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute
2	basic types, Record	x	The value to be assigned to the attribute.

Returns

None.

See Also

getAttr

`Repository:touch`

Parameters: none

Returns: none

This method sets the dirty flag of the Repository. Usually this flag is set automatically whenever you change its data (either by user interface or by Lua).

Class Residue

Inherits Class Object

`Residue:getAttr`

With this function the attributes of this object can be accessed. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table...").

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute

Returns

Nr.	*Type*	*Opt.*	*Description*
1	basic types, Record		The value of the attribute or nil, if the attribute is not defined.

See Also

setAttr

`Residue:getChain`

Parameters: none

Returns: string (name of chain)

`Residue:getId`

This function returns the identification number of the given object. This number is automatically assigned by CARA when creating the object. It is displayed in the user interface and saved in the CARA file. Identification numbers are greater than zero.

Parameters

None.

Returns

Nr.	Type	Opt.	Description
1	number		The ID number of the given object, unique within class/repository

Residue:getNr

Parameters: none

Returns: number

Residue:getPred

Parameters: none

Returns: Residue or nil.

This function returns the predecessor of the Residue in the sequenc (to N-terminus), if it has one.

Residue:getSucc

Parameters: none

Returns: Residue or nil.

This function returns the successor of the Residue in the sequence (to C-terminus), if it has one.

Residue:getSystem

Parameters: none

Returns: SpinSystem or nil.

This function returns the assigned SpinSystem (see Project:assignSystem), if it has one.

Residue:getType

Parameters: none

Returns: ResidueType (never nil)

Residue:getValue

Parameters

string (atom label)

Returns

Nr.	Type	Opt.	Description
1	number		Mean value
2	number		Deviation value

Residue:setAttr

With this function the attributes of this object can be written. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table..."). If the attribute is

read-only, execution is aborted with an error. If the second parameter is omitted, the attribute is set to nil, i.e. it is removed from the object.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute
2	basic types, Record	x	The value to be assigned to the attribute.

Returns

None.

See Also

getAttr

Class ResidueType

Inherits Class Object

This class is used to model the basic bilding blocks of biomolecules like i.e. the amino or nucleic acids. In CARA the biomolecules are represented by sequences of ResidueTypes.

`ResidueType:getAtom`

Parameters: string (atom symbol)

Returns: Atom or nil.

`ResidueType:getAtomGroup`

Parameters: string (atom group symbol)

Returns: AtomGroup or nil

`ResidueType:getAtomGroups`

Parameters: none

Returns: Table[string (atom group symbol), AtomGroup]

`ResidueType:getAtoms`

Parameters: none

Returns: Table[string (atom symbol), Atom]

`ResidueType:getAttr`

With this function the attributes of this object can be accessed. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table...").

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute

Returns

Nr.	*Type*	*Opt.*	*Description*
1	basic types, Record		The value of the attribute or nil, if the attribute is not defined.

See Also
setAttr

ResidueType:getId

This function returns the identification number of the given object. This number is automatically assigned by CARA when creating the object. It is displayed in the user interface and saved in the CARA file. Identification numbers are greater than zero.

Parameters
None.

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		The ID number of the given object, unique within class/repository

ResidueType:getLetter

Parameters: none.

Returns: string (the single letter name of the ResidueType)

ResidueType:getName

Parameters: none.

Returns: string (the full name of the ResidueType)

ResidueType:getShort

Parameters: none.

Returns: string (the three letter code of the ResidueType, also used as unique ID)

ResidueType:getSystemType

Parameters: none.

Returns: SystemType or nil.

ResidueType:setAttr

With this function the attributes of this object can be written. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table..."). If the attribute is read-only, execution is aborted with an error. If the second parameter is omitted, the attribute is set to nil, i.e. it is removed from the object.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute
2	basic types, Record	x	The value to be assigned to the attribute.

Returns
None.

See Also
getAttr

ResidueType:setValue

Parameters

Nr.	Type	Opt.	Description
1	Atom		
2	number	x	Mean value (default "undefined")
3	number	x	Deviation value

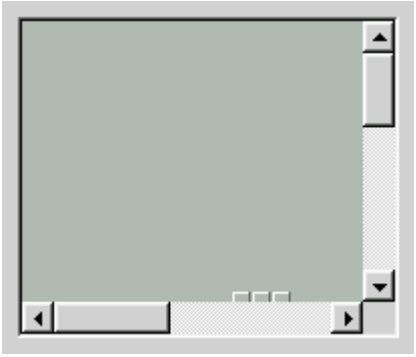
Returns

None

Class ScrollView

Inherits Class Frame

ScrollView inherits all features of Frame. It is a container for child widgets, i.e. restricts their visibility to the size of the ScrollView. The user can controll the visible detail by moving the scroll bars.



ScrollView:addChild

Parameters

Nr.	Type	Opt.	Description
1	Widget		
2	number	x	X position, default 0
3	number	x	Y position, default 0

Returns

None

ScrollView:center

Adjusts the scroll bars so the given position becomes the new center of the ScrollView (if possible).

Parameters

Nr.	Type	Opt.	Description
1	number		X position
2	number		Y position

Returns

None

ScrollView:ensureVisible

Ensures that the given position is part of the visible area. Otherwhise the scroll bars are appropriately adjusted.

Parameters

Nr.	Type	Opt.	Description
1	number		X position
2	number		Y position

Returns

None

ScrollView:moveChild

Move the given child widget to the new position.

Parameters

Nr.	Type	Opt.	Description
1	Widget		
2	number		X position
3	number		Y position

Returns

None

ScrollView:removeChild

Parameters: Widget

Returns: none

ScrollView:resizeContents

Sets the size of the scroll pane. If the pane becomes larger than the size of the ScrollWidget, scroll bars are automatically shown (and vice versa).

Parameters

Nr.	Type	Opt.	Description
1	number		Width
2	number		Height

Returns

None

ScrollView:scrollBy

Parameters

Nr.	Type	Opt.	Description
1	number		X offset
2	number		Y offset

Returns

None

ScrollView:scrollTo

Parameters

Nr.	Type	Opt.	Description
1	number		X position
2	number		Y position

Returns
None

ScrollView:setCallback

See Widget:setCallback(). The following events and callback functions are supported:

gui.event.Scrolled	function(self, x, y)	Parameters: ScrollView, number, number Returns: none

Class SlicePlot

A SlicePlot is used to draw slices of to the given Buffer to a Canvas or a Painter. It is created by dlg.createSlice. It is - like an Image - an autonomous object which can be drawn on a Canvas or Painter with the drawContour method. The object can be reused for different drawings. Changes made to SlicePlot do not affect already commited drawings (see Canvas:commit).

```
spec = cara:getProject():getSpectrum( 3 )
buf = spec:getSlicePpm( 1, 10, 7, 128, 116 )
c = dlg.createSlice( 1, buf )
cv = dlg.createCanvas()
cv:begin()
cv:drawSlice( c, 30, 40, 200, 200 )
cv:setPen( "blue", 1 )
cv:drawRect( 30, 40, 200, 200 )
cv:commit()
```

SlicePlot:getDimension

Parameters: none

Returns: number (dimension 1 or 2)

SlicePlot:getMinMax

Parameters
None

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		Minimum amplitude
2	number		Maximum amplitude

SlicePlot:recalcMinMax

Parameters: none

Returns: none

Let the slice automatically determine the minimal and maximal amplitude (from the given Buffer).

SlicePlot:setAutoScale

If auto-scale is true, the diagram automatically scales the maximum and minimum amplitudes to the given screen space. If auto-center is true, the zero-line is kept in the middle of the given screen space. If auto-scale is false, the diagram is scaled according to the values given by setMinMax.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	boolean	x	Do autoscale, default true
2	boolean	x	Do auto-center, default false

Returns

None

SlicePlot:setBuffer

Parameters: Buffer

Returns: none

This function aborts with an error, if the given Buffer is not 1D.

SlicePlot:setColor

Parameters: string (color code)

Returns: none

Set the color used to draw the diagram. The code has the usual format ("#RRGGBB", etc.).

SlicePlot:setMinMax

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Minimum amplitude
2	number		Maximum amplitude

Returns

None

SlicePlot:toPoint

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Origin (in points)
2	number		Width (in points)
3	number		PPM position

Returns

number (point position of given ppm value)

SlicePlot:toPpm

Parameters

Nr.	Type	Opt.	Description
1	number		Origin (in points)
2	number		Width (in points)
3	number		Position (in points)

Returns

number (ppm value of given point position)

Class Spectrum

Inherits Class Object

This class represents an NMR spectrum living in a file of the file system. It is an abstract concept, independent of the specific spectrum file formats (i.e. XEASY, Bruker, etc.). Spectrum data is used by means of Buffer objects, which represent planes or slices cut out of the Spectrum in arbitrary direction and orientation. The CARA spectrum engine guarantees, that cutting is efficient in terms of disc access, regardless of the choosen direction or orientation.

Spectrum:create1dProjection

This method creates a new Spectrum which is a projection of the original Spectrum (from which the method is called). The new Spectrum is only a kind of "filter", i.e. no copy of the data is created.

Parameters

Nr.	Type	Opt.	Description
1	number		Dimension of original Spectrum to be used in the projection
2	table[number (dimension), number (PPM position)]	x	Position in original Spectrum where the projection is located (default is 0 in all dimensions). Usually an explicit position is necessary in all dimensions but the projected one.

Returns

Spectrum

Spectrum:create2dProjection

This method creates a new Spectrum which is a projection of the original Spectrum (from which the method is called). The new Spectrum is only a kind of "filter", i.e. no copy of the data is created.

Parameters

Nr.	Type	Opt.	Description
1	number		Dimension of original Spectrum to be used as D1 in the projection
2	number		Dimension of original Spectrum to be used as D2 in the projection
3	table[number (dimension), number (PPM position)]	x	Position in original Spectrum where the projection is located (default is 0 in all dimensions). Usually an explicit position is necessary in all dimensions but the projected one.

Returns

Spectrum

Spectrum:createRotation

This method creates a new Spectrum which is a rotated version of the original Spectrum (from which the method is called). The new Spectrum is only a kind of "filter", i.e. no copy of the data is created.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	table[number (from dimension), number (to dimension)]		Dimension mapping, i.e. how the dimensions of the new Spectrum will be rotated compared to the original one.

Returns

Spectrum

Spectrum:getAt

This function returns the amplitude at the given index position (starting with 1). For each dimension of the Spectrum an index must be provided. When the position is out of valid range, the script is aborted with an error.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Sample index in dimension D1 of the Spectrum
n	number		Sample index in dimension Dn of the Spectrum

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		Amplitude at the given sample index.

See Also

Spectrum:getDimCount, Spectrum:getSampleCount

Spectrum:getAtomType

Parameters: number (dimension)

Returns: string (atom symbol)

See also SpectrumType:getAtomType.

Spectrum:getAtPoint

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	table[number (dimension), number (PPM position)]		PPM point within Spectrum at which the amplitude is requested
2	boolean	x	Enable folding (default is false); if true, the PPM point may be outside of the spectral width; otherwise the execution is aborted with an error.

Returns

number (amplitude)

Spectrum:getAtPpm

This function returns the amplitude at the given PPM position. For each dimension a position parameter has to be provided. The position can be out of the spectral width. In this case the folded value is returned.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		PPM value for dimension D1 of the spectrum.
n	number		PPM value for dimension Dn of the spectrum.

Returns

number (amplitude)

See Also

Spectrum:getFolding

Spectrum:getAttr

With this function the attributes of this object can be accessed. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table...").

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute

Returns

Nr.	*Type*	*Opt.*	*Description*
1	basic types, Record		The value of the attribute or nil, if the attribute is not defined.

See Also

setAttr

Spectrum:getDimCount

Parameters: none

Returns: number (of dimensions)

Spectrum:getFilePath

Parameters

None

Returns

Nr.	*Type*	*Opt.*	*Description*
1	string		The path in the file system where the spectrum (e.g. the param file, depending on the spectrum format) resides

Spectrum:getFolding

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Dimension

Returns

Nr.	Type	Opt.	Description
1	string		The symbol representing the folding type of the dimension (empty, "RSH" or "TPPI").

Spectrum:getFreq

Parameters

Nr.	Type	Opt.	Description
1	number		Dimension 1..d
2	number		Sample index 1..n

Returns

Nr.	Type	Opt.	Description
1	number		PPM position corresponding to the given sample index

See Also

Spectrum:getDimCount

Spectrum:getHisto

This method calculates two histograms, one for the positive and one for the negative amplitudes of the Spectrum.

Parameters

Nr.	Type	Opt.	Description
1	number		Number of bins for positive amplitudes
2	number		Number of bins for negative amplitudes
3	number	x	Maximum amplitude (default: calculated from spectrum)
4	number	x	Minimum amplitude (default: calculated from spectrum)

Returns

Nr.	Type	Opt.	Description
1	Table[number (array index), number (frequency)]		The histogram of the positive amplitudes
2	Table[number (array index), number (frequency)]		The histogram of the negative amplitudes

Spectrum:getId

This function returns the identification number of the given object. This number is automatically assigned by CARA when creating the object. It is displayed in the user interface and saved in the CARA file. Identification numbers are greater than zero.

Parameters

None.

Returns

Nr.	Type	Opt.	Description
1	number		The ID number of the given object, unique within class/repository

Spectrum:getIndex

Parameters

Nr.	Type	Opt.	Description
1	number		Dimension
2	number		PPM position

Returns

Nr.	Type	Opt.	Description
1	number		Sample index corresponding to the given PPM position

See Also

Spectrum:getDimCount

Spectrum:getIsotope

Parameters: number (dimension)

Returns: string (isotope label)

See also SpectrumType:getIsotope.

Spectrum:getLabel

Parameters

Nr.	Type	Opt.	Description
1	number		Dimension

Returns

Nr.	Type	Opt.	Description
1	string		The label associated with the given dimension

Spectrum:getLevels

Parameters

None

Returns

Nr.	Type	Opt.	Description
1	number		Positive maximum amplitude
2	number		Positive noise level (i.e. mean amplitude)
3	number		Negative maximum (i.e. minimum) amplitude
4	number		Negative noise level

Spectrum:getName

Returns

Nr.	Type	Opt.	Description
1	string		The name of this spectrum (usually unique within project)

Spectrum:getPeakWidth

Parameters: number (dimension)

Returns: number (PPM peak width)

Spectrum:getPlanePpm

Use this method to cut out a plane from a spectrum with n dimensions. The plane can have whatever size or direction needed. The given PPM ranges can be reversed to flip the plane along its axis. If the range is outside the spectral width, the buffer is filled with folded amplitudes. Please note that in all spectrum dimensions but the two covered by the plane, the PPM ranges should in fact be PPM points (i.e. the first and second number of the range are identical).

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Spectrum dimension mapped to Buffer dimension x
2	number		Spectrum dimension mapped to Buffer dimension y
3	number		PPM range first position in spectrum dimension D1
4	number		PPM range second position in spectrum dimension D1
n	number		PPM range first position in spectrum dimension Dn
n+1	number		PPM range second position in spectrum dimension Dn

Returns

Nr.	*Type*	*Opt.*	*Description*
1	Buffer		The 2D Buffer containing the resulting plane

See Also

Spectrum:getDimCount

Spectrum:getPlaneRange

This method is similar to Spectrum:getPlanePpm, but accepts two tables containing the ppm ranges. Each table contains a ppm value for each dimension of the spectrum (represented by index 1..n). The first table contains the first-values, the second the second-values (e.g. the range of dimension D2 would be given by first[2] and second[2]). If folded=false, the parts out of spectrum are filled with zeroes. Optionally a scale reduction can be requested by specifying the maximum number of points per Buffer dimension.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Spectrum dimension mapped to Buffer dimension x
2	number		Spectrum dimension mapped to Buffer dimension y
3	table[number (dimension), number (PPM)]		Table containing the first position of the PPM range in all spectrum dimensions
4	table[number (dimension), number (PPM)]		Table containing the second position of the PPM range in all spectrum dimensions
5	boolean	x	Enable folding (default false)
6	number	x	Number of required points in Buffer dimension x (default all)
7	number	x	Number of required points in Buffer dimension y (default all)

Returns

Buffer

Spectrum:getPpmRange

Parameters

number (dimension)

Returns

Nr.	Type	Opt.	Description
1	number		PPM value at sample index 1 (middle of sample)
2	number		PPM value at sample index n (middle of sample)

See Also

Spectrum:getDimCount, Spectrum:getSampleCount

Spectrum:getRfFreq

Parameters

Nr.	Type	Opt.	Description
1	number		Dimension

Returns

Nr.	Type	Opt.	Description
1	number		The spectrometer frequency in MHz for the given dimension.

Spectrum:getSample (also getBioSample)

Parameters: none

Returns: BioSample or nil

Spectrum:getSampleCount

Parameters: number (dimension)

Returns: number (Number of samples i.e. FFT points in the given dimensions)

Spectrum:getSlicePpm

Use this method to cut out a slice from a Spectrum with n dimensions. The slice can have whatever size or direction needed. The given PPM ranges can be reversed to flip the slice along its center axis. If the range is outside the spectral width, the buffer is filled with folded amplitudes. Please note that in all Spectrum dimensions but the one covered by the slice, the PPM ranges should in fact be PPM points (i.e. the first and second number of the range are identical).

Parameters

Nr.	Type	Opt.	Description
1	number		Spectrum dimension mapped to Buffer dimension
2	number		PPM range first position in spectrum dimension D1
3	number		PPM range second position in spectrum dimension D1
n	number		PPM range first position in spectrum dimension Dn
n+1	number		PPM range second position in spectrum dimension Dn

Returns

Nr.	Type	Opt.	Description
1	Buffer		The 1D Buffer containing the resulting plane

See Also

Spectrum:getDimCount

Spectrum:getSliceRange

This method is similar to Spectrum:getSlicePpm, but accepts two tables containing the ppm ranges. Each table contains a ppm value for each dimension of the spectrum (represented by index 1..n). The first table contains the first-values, the second the second-values (e.g. the range of dimension D2 would be given by first[2] and second[2]). If folded=false, the parts out of spectrum are filled with zeroes. Optionally a scale reduction can be requested by specifying the maximum number of points per Buffer dimension.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Spectrum dimension mapped to Buffer dimension
2	table[number (dimension), number (PPM)]		Table containing the first position of the PPM range in all spectrum dimensions
3	table[number (dimension), number (PPM)]		Table containing the second position of the PPM range in all spectrum dimensions
4	boolean	x	Enable folding (default false)
5	number	x	Number of required points in Buffer dimension (default all)

Returns

Buffer

Spectrum:getThreshold

Parameters: none

Returns: number (threshold, usually the mean amplitude of the spectrum, used by auto contour)

Spectrum:getType

Parameters: none

Returns: SpectrumType or nil

The return value is not nil for all spectra belonging to a Project.

Spectrum:setAttr

With this function the attributes of this object can be written. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table..."). If the attribute is read-only, execution is aborted with an error. If the second parameter is omitted, the attribute is set to nil, i.e. it is removed from the object.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute
2	basic types, Record	x	The value to be assigned to the attribute.

Returns

None.

See Also

getAttr

Spectrum:setPeakWidth

Parameters

Nr.	Type	Opt.	Description
1	number		dimension
2	number		The new peak width (>=0.0)

Returns

none

Spectrum:setThreshold

Parameters: number (new threshold)

Returns: none

Class SpectrumType

Inherits Class Object

A SpectrumType specifies the characteristics of a Spectrum. It defines the number, order and atom types of dimensions. For each dimension there optionally exists a list of labels of spins to be expected. The corresponding NMR experiment is modelled by its procedure steps. Each step can be seen as a kind of query for the atoms of an arbitrary molecule, i.e. the SpectrumType can decide itself, which atoms of a given molecule (modelled by Class ResidueType) will potentially be visible in the spectrum.

SpectrumType:getAtomType

Parameters: number (dimension 1..n)

Returns: string (atom symbol)

SpectrumType:getAttr

With this function the attributes of this object can be accessed. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table...").

Parameters

Nr.	Type	Opt.	Description
1	string		The name of the attribute

Returns

Nr.	Type	Opt.	Description
1	basic types, Record		The value of the attribute or nil, if the attribute is not defined.

See Also

setAttr

SpectrumType:getDimCount

Parameters: none

Returns: number (dimension count)

SpectrumType:getDimName

Parameters: number (dimension 1..n)

Returns: string

SpectrumType:getIsotope

Parameters: number (dimension)

Returns: string (isotope symbol)

SpectrumType:getKeyLabel

Parameters: number (dimension 1..n)

Returns: string (spin label syntax)

If the given dimension of the spectrum type has a single label or a single final label among other draft labels, it is returned. If there is more than one final label, nothing is returned.

SpectrumType:getLabels

Parameters: number (dimension 1..n)

Returns: Table[number (index), string (spin label syntax)]

SpectrumType:getName

Parameters: none

Returns: string

SpectrumType:getStep

Parameters: number (step number 1..n)

Returns: Table[atom: string, hops: number, repeat: boolean, mean: number or nil, dev: number or nil, dim: number or nil, name: string]

This function returns a procedure step as a Lua table with name indizes. Example:

```
st = cara:getSpectrumType( "HNCA" )
t = st:getStep( 1 )
print( t[ "atom" ] )    -- "official" notation
print( t.hops )         -- more handier notation
```

SpectrumType:getStepCount

Parameters: none

Returns: number (number of procedure steps)

SpectrumType:getStepNr

Parameters: number (Dimension 1..n)

Returns: number (corresponding step number)

SpectrumType:isNoesy

Parameters: number (dimension 1..n)

Returns: boolean

If the step with the given number is declared as NOESY (i.e. selects all spins of its atom type instead of following the magnetization transfer along bonds), this function returns true.

SpectrumType:setAttr

With this function the attributes of this object can be written. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table..."). If the attribute is read-only, execution is aborted with an error. If the second parameter is omitted, the attribute is set to nil, i.e. it is removed from the object.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute
2	basic types, Record	x	The value to be assigned to the attribute.

Returns

None.

See Alsodo

getAttr

Class Spin

Inherits Class Object

A Spin is a one-dimensional signal, visible in a Spectrum. It is yet an abstract concept, different from the concept of peaks used in other applications so far. CARA calculates the Spins visible in a given Spectrum on the fly (using the information contained in SpectrumTypes and ResidueTypes), i.e. they are not dedicated to a certain spectrum (as it was the case in most earlier applications). Even though you can set an individual position for each spectrum if needed (called *spin alias*). The identity of the Spin remains the same, even if the positions may vary with the spectra.

Spin:getAtomType

Parameters: none

Returns: string (atom type symbol)

Spin:getAttr

With this function the attributes of this object can be accessed. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table...").

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute

Returns

Nr.	*Type*	*Opt.*	*Description*
1	basic types, Record		The value of the attribute or nil, if the attribute is not defined.

See Also

setAttr

Spin:getId

This function returns the identification number of the given object. This number is automatically assigned by CARA when creating the object. It is displayed in the user interface and saved in the CARA file. Identification numbers are greater than zero.

Parameters

None.

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		The ID number of the given object, unique within class/repository

Spin:getIsotope

Parameters: none

Returns: string (isotope label)

Spin:getLabel

Parameters: none

Returns: string (spin label syntax)

Spin:getLink

Parameters: Spin

Returns: SpinLink or nil

Spin:getLinks

Parameters: none

Returns: Table[number (array index), SpinLink]

Spin:getLocation

Parameters

none

Returns

Nr.	Type	Opt.	Description
1	number		Spatial dimension x
2	number		Spatial dimension y
3	number		Spatial dimension z
4	number		Position deviation (error radius)

Spin:getShift

Parameters: Spectrum (optional)

Returns: number (PPM shift)

Spin:getShifts

Parameters: none

Returns: Table[number (Spectrum ID), number (PPM shift)]

Spin:getSystem

Parameters: none

Returns: SpinSystem or nil.

Spin:setAttr

With this function the attributes of this object can be written. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table..."). If the attribute is read-only, execution is aborted with an error. If the second parameter is omitted, the attribute is set to nil, i.e. it is removed from the object.

Parameters

Nr.	Type	Opt.	Description
1	string		The name of the attribute
2	basic types, Record	x	The value to be assigned to the attribute.

Returns

None

See Also

getAttr

Class SpinLink

Inherits Class Object

A SpinLink relates two arbitrary Spins and is thus used to model NOESY constraints (i.e. NOEs). For all intra-residual Spin CARA knows which of them belong toghether to form a peak in a certain Spectrum (this is automatically calculated on the fly by means of SpectrumTypes and ResidueTypes). CARA does not try to predict long-range spin relations, but uses the SpinLinks instead created by the user (or a Lua script). The user creates SpinLinks e.g. by picking a peak in HomoScope or PolyScope.

SpinLink:getAttr

With this function the attributes of this object can be accessed. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table...").

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute

Returns

Nr.	*Type*	*Opt.*	*Description*
1	basic types, Record		The value of the attribute or nil, if the attribute is not defined.

See Also

setAttr

SpinLink:getCode

Parameters: Spectrum (optional)

Returns: number (color code)

SpinLink:getLeft

Parameters: none

Returns: number (Spin ID)

SpinLink:getParams

Parameter

Spectrum (optional)

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		Rating
2	number		Code (0..255)
3	boolean		Visible

SpinLink:getRating

Parameters: Spectrum (optional)

Returns: number (Rating)

SpinLink:getRight

Parameters: none

Returns: number (Spin ID)

SpinLink:isVisible

Parameters: Spectrum

Returns: boolean

SpinLink:setAttr

With this function the attributes of this object can be written. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table..."). If the attribute is read-only, execution is aborted with an error. If the second parameter is omitted, the attribute is set to nil, i.e. it is removed from the object.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute
2	basic types, Record	x	The value to be assigned to the attribute.

Returns

None.

See Also

getAttr

Class SpinSystem

Inherits Class Object

A SpinSystem represents a group of Spins belonging to the same molecule. It is the purpose of CARA to support the user in detecting Spins, aggregating them in SpinSystems and finally assigning the SpinSystems to Residues (by first concatenating SpinSystems to fragments).

SpinSystem:getAttr

With this function the attributes of this object can be accessed. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table...").

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute

Returns

Nr.	*Type*	*Opt.*	*Description*
1	basic types, Record		The value of the attribute or nil, if the attribute is not defined.

See Also

setAttr

SpinSystem:getCandidates

Parameters: none

Returns: Table[number (array index), ResidueType]

SpinSystem:getId

This function returns the identification number of the given object. This number is automatically assigned by CARA when creating the object. It is displayed in the user interface and saved in the CARA file. Identification numbers are greater than zero.

Parameters

None.

Returns

Nr.	Type	Opt.	Description
1	number		The ID number of the given object, unique within class/repository

SpinSystem:getPred

Parameters: none

Returns: SpinSystem or nil.

SpinSystem:getResidue

Parameters: none

Returns: Residue or nil.

SpinSystem:getSpins

Parameters: none

Returns: Table[number (Spin ID), Spin]

SpinSystem:getSucc

Parameters: none

Returns: SpinSystem or nil.

SpinSystem:getSystemType

Parameters: none

Returns: SystemType or nil.

SpinSystem:isAcceptable

Parameters: string (spin label syntax).

Returns: boolean

Use this function to check, whether a Spin with the given label would be acceptable within the SpinSystem.

SpinSystem:setAttr

With this function the attributes of this object can be written. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table..."). If the attribute is read-only, execution is aborted with an error. If the second parameter is omitted, the attribute is set to nil, i.e. it is removed from the object.

Parameters

Nr.	Type	Opt.	Description
1	string		The name of the attribute
2	basic types, Record	x	The value to be assigned to the attribute.

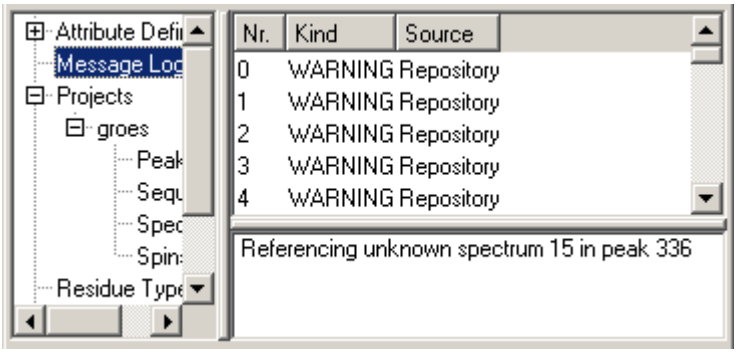
Returns
None.

See Also
getAttr

Class Splitter

Implemented by QSplitter

Splitter conceptually inherits all featurs of Class Frame. It is used to organize its sub-widgets into panes, whose borders can be interactively moved by the user.



`Splitter:setOrientation`

Parameters: number (possible values: gui.Horizontal or gui.Vertical)

Returns: none

Class Structure

Inherits Class Object

This class allows CARA to also store structure data. CARA itself does not use this data. But it is possible to run external tools from Lua which output structure data. A Structure consists of Conformers and Locations. A Conformer is a calculated version of the Structure. A Location represents the spatial position of an atom. A Structure has an origin to which all other positions are relative.

`Structure:addConformer`

Parameters: none

Returns: Conformer

`Structure:getConformer`

Parameters: number (ID of Conformer)

Returns: Conformer or nil

`Structure:getConformers`

Parameters: none

Returns: Table[number (ID of Conformer), Conformer]

Structure:getCoord

Parameters

Nr.	Type	Opt.	Description
1	number		ID of spatial position (usually corresponds to the Spin ID)

Returns

Nr.	Type	Opt.	Description
1	number		Coordinate in dimension x
2	number		Coordinate in dimension y
3	number		Coordinate in dimension z
4	number		Deviation of coordinate (error radius around position)

Structure:getCoords

Parameters: none

Returns: Table[number (array index), number (ID of coordinate)]

Structure:getId

Parameters: none

Returns: number (ID of Structure)

Structure:getName

Parameters: none

Returns: string (name of Structure)

Structure:getOrigin

The origin is the reference position of the Structure.

Parameters

None

Returns

Nr.	Type	Opt.	Description
1	number		Coordinate in dimension x
2	number		Coordinate in dimension y
3	number		Coordinate in dimension z

Structure:removeConformer

Parameters: Conformer

Returns: None

Structure:setCoord

Parameters

Nr.	Type	Opt.	Description
1	number		ID of coordinate (usually corresponds to the Spin ID)
2	number		Coordinate in dimension x
3	number		Coordinate in dimension y
4	number		Coordinate in dimension z
5	number	x	Deviation of coordinate (error radius around position, default 0.0)

Returns

None

Structure:setName

Parameters: string (New name of Structure)

Returns: none

Structure:setOrigin

Parameters

Nr.	Type	Opt.	Description
1	number		Coordinate in dimension x
2	number		Coordinate in dimension y
3	number		Coordinate in dimension z

Returns

None

Class SystemType

Inherits Class Object

A SystemType is used to classify SpinSystems and ResidueTypes. According to the process of *sequential assignment* [7], the type of many spin systems can be recognized as *AMX* etc. just by looking at the spectrum. A SpinSystem is thus assigned to the corresponding SystemType. CARA has algorithms to match SpinSystem fragments to the sequence by SystemTypes.

SystemType:getAttr

With this function the attributes of this object can be accessed. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table...").

Parameters

Nr.	Type	Opt.	Description
1	string		The name of the attribute

Returns

Nr.	Type	Opt.	Description
1	basic types, Record		The value of the attribute or nil, if the attribute is not defined.

See Also

setAttr

SystemType:getId

This function returns the identification number of the given object. This number is automatically assigned by CARA when creating the object. It is displayed in the user interface and saved in the CARA file. Identification numbers are greater than zero.

Parameters

None.

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		The ID number of the given object, unique within class/repository

SystemType:getName

Parameters: none

Returns: string (name of system type)

SystemType:setAttr

With this function the attributes of this object can be written. These are the attributes also visible from within the CARA user interface (using "Edit Attributes..." or "Open Object Table..."). If the attribute is read-only, execution is aborted with an error. If the second parameter is omitted, the attribute is set to nil, i.e. it is removed from the object.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The name of the attribute
2	basic types, Record	x	The value to be assigned to the attribute.

Returns

None.

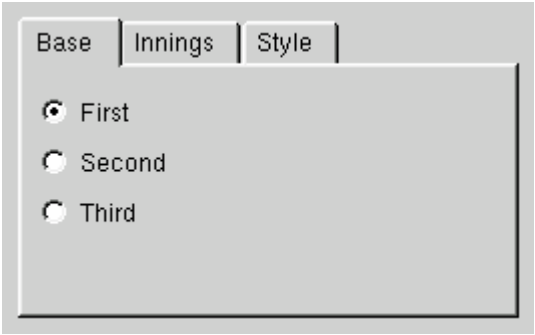
See Also

getAttr

Class TabWidget

Implemented by QTabWidget

TabWidget inherits all features of Widget. It is used to organize its sub-widgets, i.e. to group them into pages. The user can select the page displayed by clicking on the corresponting tab header.



TabWidget:addTab

Parameters

Nr.	Type	Opt.	Description
1	QWidget		The widget to add in a new page
2	string	x	The tab label of the new page

Returns

number (the index of the tab in the tab bar)

TabWidget:getCurrentPage

Parameters: none

Returns: Widget or nil

TabWidget:setCallback

See Widget:setCallback(). The following events and callback functions are supported:

gui.event.Changed	function(self, page)	Parameters: TabWidget, Widget Returns: none

TabWidget:setMargin

Parameters: number (points)

Returns: none

TabWidget:setTabEnabled

Parameters

Nr.	Type	Opt.	Description
1	QWidget		The tab page of the widget (index can be used alternatively)
2	boolean	x	Enabled, default true

Returns

None

TabWidget:setTabPosition

Parameters: number (use TabWidget.Top or TabWidget.Bottom)

Returns: none

TabWidget:setTitle

The string can contain accelerator keys, e.g. "&Title". The widget switches to this page, when the user presses ALT+T.

Parameters

Nr.	Type	Opt.	Description
1	QWidget		The tab page of the widget
2	string		New title

Returns
None

`TabWidget:showPage`

Parameters: Widget

Returns: none

Class TextView

Inherits Class ScrollView

TextView inherits all features of ScrollView. It is used to display formated HTML text to the user.

`TextView:getText`

Parameters: none

Returns: string (HTML text)

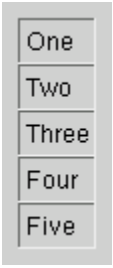
`TextView:setText`

Parameters: string (HTML text)

Returns: none

Class VBox

This class inherits all features of HBox. It is used to vertically align its child-widgets.



Class Widget

Implemented by QWidget, Inherits QObject

This is the root class of all GUI components of the CARA Object Model. It is implemented using QWidget (see page 22).

`Widget:clearFocus`

Parameters: none

Returns: none

This function removes the focus from this Widget.

`Widget:close`

Parameters: boolean (default true)

Returns: boolean

Closes this Widget and deletes it, if the parameter is true. If the parameter is false, the widget stays in memory. Returns true, when the Widget was really closed, false otherwise.

NOTE: when you call this function with a false parameter on a top level window (besides Dialog), you are responsible to later call destroy(), otherwise the hidden window stays in memory forever.

Widget:destroy

Parameters: none

Returns: none

The function deletes this widget. It disappears from screen (if it didn't already disappear before). The Lua reference is still available but no longer useful.

Widget:getBgColor

Parameters: none

Returns: string (color code, see http://www.w3.org/TR/SVG/types.html#ColorKeywords)

Widget:getCaption

Parameters: none

Returns: string

Widget:getData

Parameters: none

Returns: Table

Use this function to access the data table associated with this widget. Each widget has its own data table which you can use to save arbitrary data in the widget (i.e. references to the fields of a dialog).

You can also use the data attribute with the same effect (see page 20).

Widget:getFgColor

Parameters: none

Returns: string (color code, see http://www.w3.org/TR/SVG/types.html#ColorKeywords)

Widget:getFont

Parameters

None

Returns

Nr.	*Type*	*Opt.*	*Description*
1	string		Font name
2	number		Point size
3	boolean		Bold
4	boolean		Italic

Widget:getFrameGeometry

This function returns the position and size of this widget relative to its parent. The function considers the complete widget, including frame and decorations.

Parameters

None

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position
3	number		Width
4	number		Height

Widget:getGeometry

This function returns the position and size of the contents pane of this widget relative to its parent (not considering frame and decorations).

Parameters

None

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position
3	number		Width
4	number		Height

Widget:getMaximumSize

Parameters

None

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		Width
2	number		Height

Widget:getMinimumSize

Parameters

None

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		Width
2	number		Height

Widget:getParentWidget

Parameters: none

Returns: Widget or nil

Each Widget can have a parent Widget, in which it is contained. For some Widgets (e.g. Grid, HBox, VBox) the parent is optional, in which case the Widget becomes an independent top level window.

Widget:getSize

Parameters

None

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		Width
2	number		Height

Widget:getTopLevelWidget

Parameters: none

Returns: Widget or nil

Widgets form a hierarchy. Top level Widgets are the ones which don't have a parent by themselves, but which may have many child Widgets. When you call this function on a child Widget, it returns the root Widget.

Widget:hasFocus

Parameters: none

Returns: boolean

Widget:hide

Parameters: none

Returns: none

Use this function to make a widget disappear from the screen without deleting it.

NOTE: when you call this function on a top level window (besides Dialog), you are responsible to later call destroy(), otherwise the hidden window stays in memory forever.

Widget:isActiveWindow

Parameters: none

Returns: boolean

Returns true if this Widget is in the active window, i.e. the window that has keyboard focus. When popup windows are visible, this function returns true for both the active window and the popup.

Widget:isDesktop

Parameters: none

Returns: boolean

Returns true if the Widget is a desktop Widget, otherwise false. A desktop Widget is also a top-level Widget.

Widget:isEnabled

Parameters: none

Returns: boolean

Returns true if the Widget is enabled, or false if it is disabled.

Widget:isFocusEnabled

Parameters: none

Returns: boolean

Returns true if the Widget accepts keyboard focus, or false if it does not.

Widget:isHidden

Parameters: none

Returns: boolean

Returns true if the Widget is explicitly hidden, or false if it is visible or would become visible if all its ancestors became visible.

Widget:isMaximized

Parameters: none

Returns: boolean

Returns true if this Widget is top-level and maximized, or else false. Note that due to limitations in some window-systems, this does not always report correct results.

Widget:isMinimized

Parameters: none

Returns: boolean

Returns true if this Widget is top-level and minimized (iconified), or else false.

Widget:isModal

Parameters: none

Returns: boolean

Returns true, if this is a modal Widget, e.g. a Dialog.

Widget:isPopup

Parameters: none

Returns: boolean

Returns true, if this is a popup Widget, e.g. a PopupMenu.

Widget:isTopLevel

Parameters: none

Returns: boolean

Returns true if the Widget is top-level, otherwise false. A top-level Widget usually has a frame and a caption (title bar). Popup and desktop widgets are also top-level Widgets.

Top-level widgets can have a parent widget. It will then be grouped with its parent: deleted when the parent is deleted, minimized when the parent is minimized etc. If supported by the window manager, it will also have a common taskbar entry with its parent. Dialog and MainWindow widgets are by default top-level, even if a parent widget is specified in the create function.

Widget:isVisible

Parameters: none

Returns: boolean

Returns true if the widget itself is visible, or else false. Calling show() sets the widget to visible status if all its parent widgets up to the toplevel widget are visible. If an ancestor is not visible, the widget won't become visible until all its ancestors are shown. Calling hide() hides a widget explicitly. An explicitly hidden widget will never become visible, even if all its ancestors become visible. Iconified top-level widgets also have hidden status and isMinimized() returns true. Windows that live on another virtual desktop (on platforms that support this concept) also have hidden status.

This function returns true if the widget currently is obscured by other windows on the screen, but would be visible if moved.

Widget:lower

Parameters: none

Returns: none

Lowers the Widget to the bottom of the parent widget's stack. If there are siblings of this Widget that overlap it on the screen, this Widget will be obscured by its siblings afterwards

Widget:mapFromGlobal

Translates the global screen coordinates to Widget coordinates.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position

Widget:mapFromParent

Translates the parent coordinates to Widget coordinates.

Parameters

Nr.	Type	Opt.	Description
1	number		X position
2	number		Y position

Returns

Nr.	Type	Opt.	Description
1	number		X position
2	number		Y position

Widget:mapToGlobal

Translates the Widget coordinates to global screen coordinates.

Parameters

Nr.	Type	Opt.	Description
1	number		X position
2	number		Y position

Returns

Nr.	Type	Opt.	Description
1	number		X position
2	number		Y position

Widget:mapToParent

Translates the Widget coordinates to parent coordinates.

Parameters

Nr.	Type	Opt.	Description
1	number		X position
2	number		Y position

Returns

Nr.	Type	Opt.	Description
1	number		X position
2	number		Y position

Widget:move

Move this Widget to the given position relative to its parent (or the screen, when there is no parent).

Parameters

Nr.	Type	Opt.	Description
1	number		X position
2	number		Y position

Returns

None

Widget:raise

Parameters: none

Returns: none

Raises this Widget to the top of the parent widget's stack. If there are any siblings of this Widget that overlap it on the screen, it will be visually put in front of its siblings.

Widget:resize

Resize the widget to the given size. The size is adjusted if it is outside the minimum or maximum widget size.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Width
2	number		Height

Returns

None

Widget:setActiveWindow

Parameters: none

Returns: none

Sets the top-level Widget containing this Widget to be the active window. An active window is a visible top-level window that has the keyboard input focus.

This function performs the same operation as clicking the mouse on the title bar of a top-level window. On X11, the result depends on the window manager. If you want to ensure that the window is stacked on top as well, call raise() in addition. Note that the window has be to visible, otherwise this function has no effect.

On Microsoft Windows, if you are calling this when the application is not currently the active one then it will not make it the active window. It will flash the task bar entry blue to indicate that the window has done something. This is due to Microsoft not allowing an application to interrupt what the user is currently doing in another application.

Widget:setBgColor

Parameters: string (color code, see http://www.w3.org/TR/SVG/types.html#ColorKeywords)

Returns: none

Set the background color of this Widget.

Widget:setCallback

Use this function to associate a Widget event (e.g. Clicked for a PushButton) with a Lua function to be called when the event happens. Not all Widgets send the same events. Please check the descriptions of the individual widgets.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Event ID
2	function or nil		

Returns

None

Example

The following example opens a top-level widget and installs a paint callback, which draws a contour plot. The plot scales with the size of the window.

```
w = gui.createWidget()
w:setCaption( "Plot Test" )
w:setBgColor( "black" )
w:show()
w:getData().spec = cara:getProject():getSpectrum( 3 )
w:getData().buf = w:getData().spec:getPlanePpm(
     1, 2, 10, 7, 128, 116 )
w:getData().c = dlg.createContour( w:getData().buf )
w:setCallback( gui.event.Paint,
     function( self, p )
          local w, h = self:getSize()
          p:drawContour( self:getData().c, 0, 0, w, h )
     end )
w:update()
```

Widget:setCaption

Parameters: string

Returns: none

Set the window title of this Widget. Has no effect, if the Widget doesn't display a title bar.

Widget:setEnabled

Parameters: boolean (optional, default true)

Returns: none

Enables Widget input events if true, otherwise disables input events. An enabled Widget receives keyboard and mouse events; a disabled widget does not. Note that an enabled Widget receives keyboard events only when it is in focus.

Some widgets display themselves differently when they are disabled. For example a button might draw its label grayed out. Disabling a widget implicitely disables all its children. Enabling respectively enables all child widgets unless they have been explicitly disabled.

Widget:setFixedSize

Sets this Widget to a fixed size, i.e. it will no longer dynamically adapt its size to its parent.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Width
2	number		Height

Returns

None

Widget:setFocus

Parameters: none

Returns: none

Gives the keyboard input focus to the widget.

Widget:setFont

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		Font name
2	number		Point size
3	boolean		Bold
4	boolean		Italic

Returns

None

Widget:setMaximumSize

After calling this function the Widget will no longer automatically increase its size above the given values. Wenn it is a top-level Widget, the user cannot extend the size of the window above the given values.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Width
2	number		Height

Returns

None

Widget:setMinimumSize

After calling this function the Widget will no longer automatically decrease its size below the given values. Wenn it is a top-level Widget, the user cannot reduce the size of the window below the given values.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Width
2	number		Height

Returns

None

Widget:setUpdatesEnabled

Parameters: boolean (optional, default true)

Returns: none

Enables Widget updates if true, or disables Widget updates if false. Calling update() has no effect if updates are disabled. Paint events from the window system are processed normally even if updates are disabled.

This function can be used to disable updates for a short period of time, for instance to avoid screen flicker during a series of changes.

Widget:show

Parameters: none

Returns: none

Makes the Widget and its child Widgets visible.

Widget:showFullScreen

Parameters: none

Returns: none

Shows the Widget in full-screen mode. Calling this function has no effect if the Widget is not top-level. To return from full-screen mode, call showNormal().

Full-screen mode works fine under Windows, but depends on window manager capabilities under X11.

Widget:showMaximized

Parameters: none

Returns: none

Shows the Widget maximized. Calling this function has no effect if the Widget is not top-level.

On X11, this function may not work properly with certain window managers.

Widget:showMinimized

Parameters: none

Returns: none

Shows the Widget minimized and shown as an icon. Calling this function has no if the Widget is not top-level.

Widget:showNormal

Parameters: none

Returns: none

Restores the Widget after it has been maximized or minimized. Calling this function has no effect if the Widget is not top-level.

Widget:update

Use this function to update the given rectangle of the Widget. It will be redrawn asynchronously. If the parameters are left out, the whole area of the widget will be redrawn.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number	x	X position
2	number	x	Y position
3	number	x	Width
4	number	x	Height

Returns
None

Widget:updateAll

Parameters: none

Returns: none

This function has the same effect as update() without parameters.

Widget:updateGeometry

Parameters: none

Returns: none

Notifies the layout system that this Widget has changed geometry.

Class WidgetStack

Inherits Class Frame

This class inherits all features of Frame. It is used to organize its child-widgets as a stack, while only the top most Widget is visible.

WidgetStack:addWidget

Parameters: Widget

Returns: none

Adds a Widget to the stack. It is initially hidden.

WidgetStack:getTopWidget

Parameters: none

Returns: Widget

WidgetStack:setTopWidget

Parameters: Widget

Returns: none

Puts the given Widget on top of the stack and thus makes it visible.

Library dlg

dlg.beginProgress

This function opens the progress dialog, showing a progress bar to the user and giving him the possibility to cancel the execution. Use ist whenever the execution of a function is expected to last for more than a few seconds.

This function can be called recursively by more than one function. The first caller determines the title of the dialog.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string	x	Sets the caption of the progress dialog
2	string	x	Sets the label text above the progress bar
3	string	x	Sets the text of the cancel button

Returns

None

Example

```
dlg.beginProgress( "Testing Progress..." )
max = 10000000
for i = 1, max do
     if math.mod( i, 1000 ) == 0 then
          dlg.updateProgress( 100 * i / max )
     end
end
dlg.endProgress()
```

See Also

endProgress, updateProgress

dlg.createCanvas

This function opens a new Canvas window and returns the object reference to the caller. You can open as many canvasses as you need. If you save the reference in a global variable, the canvas can even be reused by different scripts. Please note that the reference becomes invalid as soon as the user closes the Canvas window.

Parameters

None

Returns

Nr.	*Type*	*Opt.*	*Description*
1	Canvas		A new Canvas ready to be drawn on

See Also

Class Canvas

dlg.createContour

Parameters: Buffer

Returns: ContourPlot

This function aborts with an error, if the given Buffer is not 2D.

dlg.createIcon

Parameters: string (XPM format, see https://en.wikipedia.org/wiki/X_PixMap)

Returns: Icon

The following example script opens a Canvas and displays the following icon: 

```
xpm = [[/* XPM */
"16 16 7 1"
"# c #000000"
"b c #ffffff"
"e c #000000"
"d c #404000"
"c c #c0c000"
"a c #ffffc0"
". c None"
"................"
".........#......"
"......#.#a##...."
".....#b#bbba##.."
"....#b#bbbabbb#."
"...#b#bba##bb#.."
"..#b#abb#bb##..."
".#a#aab#bbbab##."
"#a#aaa#bcbbbbbb#"
"#ccdc#bcbbcbbb#."
".##c#bcbbcabb#.."
"...#acbacbbbe..."
"..#aaaacaba#...."
"...##aaaaa#....."
".....##aa#......"
".......##......."
]]
icon = dlg.createIcon( xpm )
cv = dlg.createCanvas()
cv:begin()
cv:drawIcon( icon, 30, 30 )
cv:commit()
```

Please note that the above format is critical, also the /* XPM */ and the quotation marks. Otherwise the icon cannot be loaded and the function is aborted with an error.

dlg.createSlice

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number		Direction of slice: 1..x, 2..y
2	Buffer	x	The Buffer must have one dimension

Returns

SlicePlot

dlg.doscript

This function tries to comile and execute the script referenced by name. When there are compilation or runtime errors, the execution of the caller is aborted with an error message printed to the terminal.

Take care that you don't call scripts recursively, otherwise a stack overflow error occurs.

In contrast to `require`, this function always executes the referenced script (not only the first time).

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		Name of the script to be executed. Must be a script within the repository currently open

Returns

None

See Also

require, dofile in Lua base library [1]

dlg.endProgress

This function closes the progress dialog. You have to explicitly call it after a beginProgress, unless the user pressed *Cancel*, in which case endProgress is internally called and the execution of the script is aborted with an error message.

This function can be called recursively by more than one function. It should be called as many times as beginProgress.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	boolean	x	Kills the progress dialog, regardless whether beginProgress was called recursively.

Returns

None

See Also

dlg.beginProgress, dlg.updateProgress

dlg.getCurrentDir

This function gives access to the application wide variable, which is automatically updated to the directory where the user selected the last file from.

Parameters

None

Returns

Nr.	*Type*	*Opt.*	*Description*
1	string		The directory last set by CARA or setCurrentDir

See Also

dlg.setCurrentDir

dlg.getDecimal

This function displays a modal input dialog, where the user can enter a decimal number. The number entered by the user is returned to the calling function.

Parameters

Nr.	Type	Opt.	Description
1	string	x	The title of the input dialog
2	string	x	An optional message text displayed on the dialog
3	number	x	The default value to be displayed on the dialog

Returns

Nr.	Type	Opt.	Description
1	number or nil		The number entered by the user or nil if she pressed cancel

See Also

dlg.getText, dlg.getInteger, dlg.getSymbol

dlg.getInteger

This function displays a modal input dialog, where the user can enter an integer number. The number entered by the user is returned to the calling function.

Parameters

Nr.	Type	Opt.	Description
1	string	x	The title of the input dialog
2	string	x	An optional message text displayed on the dialog
3	number	x	The default value to be displayed on the dialog

Returns

Nr.	Type	Opt.	Description
1	number or nil		The number entered by the user or nil if he pressed cancel

Example

```
local id = dlg.getInteger( "Select Peaklist",
     "Please enter a valid peaklist ID:" )
if id == nil then
     return
end
```

See Also

dlg.getText, dlg.getDecimal, dlg.getSymbol

dlg.getOpenFileName

This function displays a modal file selector dialog. The user can navigate the file system and select a file. The dialog starts at the directory given by getCurrentDir.

Parameters

Nr.	Type	Opt.	Description
1	string	x	The title of the file dialog
2	string	x	The filter pattern to restrict the selection to files of a certain ending (optional all files)

Returns

Nr.	Type	Opt.	Description
1	string or nil		The path to the file the user selected or nil if he pressed cancel

See Also

dlg.getSaveFileName, getCurrentDir

dlg.getSaveFileName

This function displays a modal file selector dialog. The user can navigate the file system, select the target directory and specify a file name. The dialog starts at the directory given by getCurrentDir.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string	x	The title of the file dialog
2	string	x	The filter pattern to restrict the selection to files of a certain ending (optional all files)

Returns

Nr.	*Type*	*Opt.*	*Description*
1	string or nil		The path to the file the user selected or nil if he pressed cancel

See Also

dlg.getOpenFileName, dlg.getCurrentDir

dlg.getSymbol

This function displays a modal input dialog, where the user can select a text string from a list. The string selected by the user is returned to the calling function.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The title of the input dialog
2	string		An optional message text displayed on the dialog
3	string	x	The first string to be displayed in the list, where the user can select from
n	string	x	The last string to be displayed in the list, where the user can select from

Returns

Nr.	*Type*	*Opt.*	*Description*
1	string or nil		The string selected by the user or nil if he pressed cancel

See Also

dlg.getInteger, dlg.getDecimal, dlg.getText

dlg.getText

This function displays a modal input dialog, where the user can enter a text string. The string entered by the user is returned to the calling function.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string	x	The title of the input dialog
2	string	x	An optional message text displayed on the dialog
3	string	x	The default string value to be displayed on the dialog

Returns

Nr.	*Type*	*Opt.*	*Description*
1	string or nil		The string entered by the user or nil if he pressed cancel

See Also

dlg.getInteger, dlg.getDecimal, dlg.getSymbol

dlg.loadIcon

Parameters

Nr.	Type	Opt.	Description
1	string		File path
2	string	x	Format; if the format is not specified (which is the default), the loader probes the file for a header to guess the file format.

Returns

Icon

dlg.loadImage

Use this function to load a bitmap image from a file. The function can automatically recognize the format of the image file. You can support the guess by providing a format symbol like "JPEG". The following formats are supported: PNG, BMP, XBM, XPM, PNM and JPEG.

Parameters

Nr.	Type	Opt.	Description
1	string		File path
2	string	x	Format; if the format is not specified (which is the default), the loader probes the file for a header to guess the file format.

Returns

Image

dlg.logError

This function accepts a message argument and puts it as error in the message log. The user can check the message log by selecting the corresponding category in the CARA explorer.

Parameters

Nr.	Type	Opt.	Description
1	string		Message to be displayed in the log

Returns

None

See Also

dlg.logInfo, dlg.logWarning

dlg.logInfo

This function accepts a message argument and puts it as information in the message log. The user can check the message log by selecting the corresponding category in the CARA explorer.

Parameters

Nr.	Type	Opt.	Description
1	string		Message to be displayed in the log

Returns

None

See Also

dlg.logError, dlg.logWarning

dlg.logWarning

This function accepts a message argument and puts it as warning in the message log. The user can check the message log by selecting the corresponding category in the CARA explorer.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		Message to be displayed in the log

Returns

None

See Also

dlg.logInfo, dlg.logError

dlg.openAttributeEditor

Use this function to open a dialog where the dynamic attributes of the object can be set by the user.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Object		The object for which the attribute dialog shall be displayed
2	string	x	Class to use for attribute names; default determines the class from the given object; usually you don't have to set this parameter; keep in mind that dynamic attributes are only supported for the classes displayed in the CARA Explorer.

Returns

None

dlg.popCursor

Parameters: none

Returns: none

This function restores the cursor to the one befor calling dlg.pushCursor or dlg.pushHourglass.

dlg.pushCursor

Parameters: number (cursor ID)

Returns: none

Use this function to set the cursor to one of the following IDs:

1..Arrow, 2..alternative Arrow, 3..Cross, 4..Wait, 5..I-Beam, 6..vertical Resizer, 7..horizontal Resizer, 8..north-east Resizer, 9..south-east Resizer, 10..all direction Resizer, 11..vertical Splitter, 12..horizontal Splitter, 13..Hand.

dlg.pushHourglass

Parameters: none

Returns: none

Shortcut to display the Wait cursor.

dlg.require

This function tries to compile and execute the script referenced by name. When there are compilation or runtime errors, the execution of the caller is aborted with an error message printed to the terminal.

Take care that you don't call scripts recursively, otherwise a stack overflow error occurs.

In contrast to doscript, this function executes the referenced script only once (and again whenever it changed). This function thus is suited to decompose large applications into separate scripts (which can be seen as "script libraries").

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		Name of the script to be executed. Must be a script in the repository currently open.

Returns

None

See Also

doscript, require in Lua base library [1]

dlg.selectColor

Parameters: string (optional, color code)

Returns: string or nil (color code)

Use this dialog to interactively select a color. You can preset the selected color by providing a parameter. If the dialog is canceled by the user, the return value is nil.

dlg.selectFont

Use this dialog to interactively select a font. You can preset the selected font by providing all four parameters (either all or none). If the dialog was canceled by the user, all four return values are nil.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string	x	Font name
2	number	x	Point size
3	boolean	x	Bold
4	boolean	x	Italic

Returns

Nr.	*Type*	*Opt.*	*Description*
1	string	x	Font name
2	number	x	Point size
3	boolean	x	Bold
4	boolean	x	Italic

dlg.setCurrentDir

This function allows to explicitly preset the application wide variable, which is automatically updated to the directory where the user selected the last file from.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		A string representing a directory

Returns

None

See Also

dlg.getCurrentDir

dlg.showError

This function displays a modal error dialog to the user. The dialog can have one to three buttons, whose labels can be specified as parameters of the function (default is one OK button).

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The title of the dialog
2	string		The message text to be displayed on the dialog
3	string	x	The label text on button 1 (left)
4	string	x	The label text on button 2 (middle)
5	string	x	The label text on button 3 (right)

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		The butten pressed by the user: 1, 2 or 3

Example

```
dlg.showError( "Select Peaklist", "Unknown peaklist" )
```

See Also

dlg.showInfo, dlg.showWarning

dlg.showInfo

This function displays a modal information dialog to the user. The dialog can have one to three buttons, whose labels can be specified as parameters of the function (default is one OK button).

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The title of the dialog
2	string		The message text to be displayed on the dialog
3	string	x	The label text on button 1 (left)
4	string	x	The label text on button 2 (middle)
5	string	x	The label text on button 3 (right)

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		The butten pressed by the user: 1, 2 or 3

See Also

dlg.showError, dlg.showWarning

dlg.showWarning

This function displays a modal warning dialog to the user. The dialog can have one to three buttons, whose labels can be specified as parameters of the function (default is one OK button).

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		The title of the dialog
2	string		The message text to be displayed on the dialog
3	string	x	The label text on button 1 (left)
4	string	x	The label text on button 2 (middle)
5	string	x	The label text on button 3 (right)

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		The butten pressed by the user: 1, 2 or 3

See Also

dlg.showError, dlg.showInfo

dlg.updateProgress

Call this function to update the progress bar. Whenever you call this function (and only then), the system checks whether the user pressed Cancel. In this case, the execution is aborted and an error is thrown.

This function can be called recursively by more than one function. The most recent caller determines the progress value and label text.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	number	x	A percent value (0..100) expressing the progress of the calculation, i.e. the elapsed time.
2	string	x	Sets the label text above the progress bar

Returns

None

See Also

dlg.beginProgress, dlg.endProgress

Library gui

gui.createButtonGroup

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Widget		Parent Widget
2	string	x	Title
3	number	x	Column count, default 1

Returns

ButtonGroup

gui.createCheckBox

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Widget		Parent Widget
2	string	x	Label

Returns

CheckBox

gui.createComboBox

Parameters: Widget (parent)

Returns: ComboBox

gui.createDialog

Parameters: Widget or nil (parent)

Returns: Dialog

gui.createFrame

Parameters: Widget or nil (parent)

Returns: Frame

gui.createGrid

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Widget or nil		Parent Widget
2	number		Column count
3	boolean	x	Vertical orientation, default true

Returns

Grid

gui.createGroupBox

Parameters: Widget (parent)

Returns: GroupBox

gui.createHBox

Parameters: Widget or nil (parent)

Returns: HBox

gui.createLabel

Parameters

Nr.	Type	Opt.	Description
1	Widget		Parent Widget
2	string	x	Text

Returns

Label

gui.createLineEdit

Parameters: Widget (parent)

Returns: LineEdit

gui.createListView

Parameters: Widget (parent) or nil

Returns: ListView

gui.createLuaEdit

Parameters: Widget (parent, optonal)

Returns: LuaEdit

gui.createMainWindow

Parameters: none

Returns: MainWindow

gui.createMultiLineEdit

Parameters: Widget (parent)

Returns: MultiLineEdit

gui.createPopupMenu

Parameters: Widget (parent)

Returns: PopupMenu

gui.createPrinter

Parameters: none

Returns: Printer

gui.createPushButton

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Widget		Parent Widget
2	string	x	Caption

Returns

PushButton

gui.createRadioButton

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Widget		Parent Widget
2	string	x	Caption

Returns

RadioButton

gui.createScrollView

Parameters: Widget (parent) or nil

Returns: ScrollView

gui.createSplitter

Parameters: Widget (parent)

Returns: Splitter

gui.createTabWidget

Parameters: Widget (parent) or nil

Returns: TabWidget

gui.createTextView

Parameters: Widget (parent)

Returns: TextView

gui.createVBox

Parameters: Widget (parent) or nil

Returns: VBox

gui.createWidget

Parameters: Widget (parent) or nil

Returns: MyWidget

gui.createWidgetStack

Parameters: Widget (parent)

Returns: WidgetStack

gui.getCursorPos

This function returns the current position of the mouse cursor in global coordinates (i.e. 0/0 is at the upper left corner of the screen). Use Widget:mapFromGlobal to project the position on a window.

Parameters

None

Returns

Nr.	*Type*	*Opt.*	*Description*
1	number		X position
2	number		Y position

Library spec

spec.composeLabel

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		Atom label
2	number		Offset
3	number		State, one of Draft..0, Assigned..1, Finalized..2)

Returns

string (spin label syntax)

spec.createExperiment

This function creates a new Experiment object. The path table is not calculated unless both spectrum and residue type are specified (can also be done later).

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	SpectrumType	x	
2	ResidueType	x	

Returns

Experiment

spec.createPeakList

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	string		Atom type of dimension D1
n	string		Atom type of dimension Dn
n+1	Spectrum	x	Home Spectrum of PeakList; alternatively this method can also be called with a Spectrum argument only, in which case the dimensions and atom types are taken from the Spectrum

Returns

PeakList

spec.createProtonList

Parameters: number (atom count)

Returns: ProtonList

Use this function to create a ProtonList with a given number of entries. This number cannot be changed afterwards. The atoms are initialized to "invalid" (but legal) values.

spec.createRepository

Parameters: none

Returns: Repository

spec.decomposeLabel

Parameters

string (spin label syntax)

Returns

Nr.	*Type*	*Opt.*	*Description*
1	string		Atom label
2	number		Offset
3	numbe		State, one of Draft..0, Assigned..1, Finalized..2

spec.openPeakList

Parameters: string (file path to XEASY peak list), Spectrum (optional, used as home)

Returns: PeakList

This function aborts with an error, if the PeakList cannot be loaded.

```
-- Load a Peaklist with a File Selector
pl = spec.openPeakList( dlg.getOpenFileName( "Load Peaklist",
     "Peaklist (*.peaks)" ) )
```

spec.openProtonList

Parameters: string (file path)

Returns: ProtonList

Tries to parse a ProtonList from a file with the given path. The function aborts with an error, when the file could not be opened or had a syntax error.

spec.openSpectrum

Parameters: string (file path to spectrum)

Returns: Spectrum

This function aborts with an error, if the Spectrum cannot be loaded. The format is automatically determined from the file ending. CARA 1.9.1 supports XEASY, Bruker, Felix, NMR Pipe and Sparky spectrum formats, as well as its own formats (Sitar [6] and CARA [3] section 7.5).

spec.saveBuffer

This function stores a Buffer to a file using the CARA spectrum file format. The maximum and minimum amplitudes can explicitly be set for the purpose of clipping unused intensities (e.g. the water line) to save the bits for the interesting amplitude range.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Buffer		The Buffer to be saved to file.
2	string		File path
3	number	x	Format code (see also [3] section 7.5): 0...symmetric 8bit log-compressed 1...adaptive 8bit uncompressed 2...adaptive 16bit uncompressed (Default) 3...floating point IEEE 32bit uncompressed 4...symmetric 8bit gauss-compressed 5...adaptive 8bit log-compressed
4	number	x	Maximum amplitude (default: calculated maximum amplitude)
5	number	x	Minimum amplitude (default: calculated minimum amplitude)

Returns

None

spec.saveSpectrum

This function stores a whole Spectrum to a file using the CARA spectrum file format. The maximum and minimum amplitudes can explicitly be set for the purpose of clipping unused intensities (e.g. the water line) to save the bits for the interesting amplitude range.

Parameters

Nr.	*Type*	*Opt.*	*Description*
1	Spectrum		The Spectrum object to be saved.
2	string		File path
3	number	x	Format code (see also [3] section 7.5): 0...symmetric 8bit log-compressed 1...adaptive 8bit uncompressed 2...adaptive 16bit uncompressed (Default) 3...floating point IEEE 32bit uncompressed 4...symmetric 8bit gauss-compressed 5...adaptive 8bit log-compressed
4	number	x	Maximum amplitude (default: calculated maximum amplitude)
5	number	x	Minimum amplitude (default: calculated minimum amplitude)

Returns
None

Library xml

`xml.createDocument`

Parameters: string (optional, root element name)

Returns: DomDocument

Use this function to create a new XML tree as a DOM (Document Object Model, see http://www.w3.org/DOM/) document. The optional parameter controls the name of the root element (see DomDocument:getDocumentElement). When it is left out, the root element gets the name "untitled".

`xml.openDocument`

Parameters: string (file path)

Returns: DomDocument

With this function you can open an XML file and parse its contents to a DomDocument. The function aborts with an error if it could not open the file or there were parsing errors. No consistency checks are made besides the "well formdness" check of the XML syntax. Syntax errors are reported to the shell where CARA was started from.

`xml.parseDocument`

Parameters: string (XML syntax)

Returns: DomDocument

This function creates a DomDocument by parsing the XML string given as parameter. If the string has syntax errors, the function aborts with an error message. No consistency checks are made besides the "well formdness" check of the XML syntax. Syntax errors are reported to the shell where CARA was started from.